

Embedding Requirements within the Model Driven Architecture

By

Ali Fouad BA (Hons.), MSc

Member, British Computer Society; IEEE Computer Society;
Association for Computing Machinery.

A thesis submitted in partial fulfilment of the requirements of
Bournemouth University for the degree of Doctor of Philosophy (Ph.D)

Principal Supervisor: Dr. Keith Thomas Phalp
Associate Supervisor: Dr. John Mathenge Kanyaru

Software Systems Research Centre
Bournemouth University
Bournemouth, UK

September 2011

Copyright Statement

This copy of the thesis has been supplied on condition that anyone who consults it is understood to recognise that its copyright rests with its author and due acknowledgement must always be made of the use of any material contained in, or derived from, this thesis.

Keywords

Software Development; Requirements Engineering; Specification; Model Driven Architecture; MDA; Model Driven Development; MDD; Model Driven Engineering; MDE; Computation Independent Model; CIM; Role Activity Diagram; RAD; Transformations.

The ability to efficiently design appropriate computer systems and enable them to evolve over their lifetime depends on the extent to which... knowledge can be captured (Greenspan et al. 1982).

Abstract

Name of Author: Ali Fouad

Thesis Title: Embedding Requirements within the Model Driven Architecture

The Model Driven Architecture (MDA) is offered as one way forward in software systems modelling to connect software design with the business domain. The general focus of the MDA is the development of software systems by performing transformations between software design models, and the automatic generation of application code from those models. Software systems are provided by developers, whose experience and models are not always in line with those of other stakeholders, which presents a challenge for the community. From reviewing the available literature, it is found that whilst many models and notations are available, those that are significantly supported by the MDA may not be best for use by non technical stakeholders. In addition, the MDA does not explicitly consider requirements and specification.

This research begins by investigating the adequacy of the MDA requirements phase and examining the feasibility of incorporating a requirements definition, specifically focusing upon model transformations. MDA artefacts were found to serve better the software community and requirements were not appropriately integrated within the MDA, with significant extension upstream being required in order to sufficiently accommodate the business user in terms of a requirements definition. Therefore, an extension to the MDA framework is offered that directly addresses Requirements Engineering (RE), including the distinction of analysis from design, highlighting the importance of specification. This extension is suggested to further the utility of the MDA by making it accessible to a wider audience upstream, enabling specification to be a direct output from business user involvement in the requirements phase of the MDA. To demonstrate applicability, this research illustrates the framework extension with the provision of a method and discusses the use of the approach in both academic and commercial settings. The results suggest that such an extension is academically viable in facilitating the move from analysis into the design of software systems, accessible for business use and beneficial in industry by allowing for the involvement of the client in producing models sufficient enough for use in the development of software systems using MDA tools and techniques.

List of Contents

Copyright Statement.....	2
Keywords	3
Abstract	5
List of Contents	6
List of Figures	9
List of Tables.....	11
Published Materials.....	12
Acknowledgements	13
Declaration of Original Authorship	14
List of Acronyms.....	15
1.0 Introduction	18
1.1 Rationale.....	19
1.2 Scope and Aims.....	21
1.3 Report Structure	23
2.0 Literature Review (State of Art)	24
2.1 The MDA Prescription	24
2.1.1 Viewpoints.....	25
2.1.2 Standards	26
2.1.3 Transformations.....	28
2.1.4 Specification and the CIM.....	31
2.2 The Business Perception	33
2.2.1 Business Process Nature.....	34
2.2.2 Workflow Management.....	35
2.2.3 Business Process Modelling	37
2.2.4 Role Activity Diagram (RAD)	39
2.2.5 Human Interaction Management Systems (HIMS).....	41
2.2.6 Goal Modelling.....	41
2.2.7 Sketch Recognition.....	44
2.3 Alternative Approaches.....	45
2.3.1 Value Chain	45
2.3.2 Balanced Scorecard	46
2.3.3 Enterprise Modelling	47
2.3.4 Software Process Modelling	49
2.3.5 Process Programming	52
2.3.6 Functional Modelling	53
2.3.7 Business Rules	56
2.3.8 Software Product Lines	58
2.3.9 Agile Methods	59
2.3.10 Service Oriented Architecture (SOA)	60
2.4 Summary.....	61
3.0 Research Overview	63
3.1 Ontology & Epistemology.....	64
3.2 Research Paradigm	64
3.3 Research Methodology.....	66
3.3.1 Phenomenology	66
3.3.2 Ethnography.....	67
3.3.3 Grounded Theory.....	67
3.3.4 Template Analysis.....	67
3.3.5 Surveying.....	68
3.3.6 Interviewing.....	68

3.3.7 Case Study	68
3.4 Knowledge.....	69
3.4.1 MDA Evaluation.....	70
3.4.2 Investigate Requirements Solution.....	71
3.4.3 Investigate Potential Extension Mechanisms	72
3.4.4 Verify Extension Mechanisms	72
3.4.5 Software Support for Research.....	73
3.4.6 Ethical, Health & Safety and Risk.....	74
4.0 Adequacy of the CIM.....	75
4.1 Examination of the CIM definition within the MDA and the appropriation of it as an interface with the business user for defining requirements in MDA notations.....	75
4.1.1 The Connection between the MDA and Business.....	75
4.1.2 The Sufficiency of the CIM at Delivering Requirements to the MDA.....	78
4.2 Extending the Model Driven Architecture with pre-CIM.....	87
4.3 Summary.....	89
5.0 Situating Requirements within the CIM	91
5.1 Discovery of how other modelling techniques which are accessible to the business user, might be integrated with the MDA in terms of method and notation, with the focus on transformation and traceability	91
5.1.1 PIM Support for Requirements (CIM-to-PIM)	91
5.1.2 CIM Support for Requirements (CIM-to-CIM)	113
5.2 Implications	119
5.3 Summary.....	122
6.0 Extended MDA	124
6.1 Extending the MDA	124
6.2 Importance of Specification	126
6.3 xMDA Framework	128
6.4 xMDA Application.....	130
6.4.1 Moving from Analysis into Specification	131
6.4.2 Accounting for Specification.....	132
6.4.3 Moving from Specification into Design.....	133
6.4.4 Class Discovery, Transformation and Platform Information.....	134
6.4.5 Iteration.....	134
6.5 Summary.....	134
7.0 xMDA Illustration.....	136
7.1 Order Processing Worked Example	136
7.2 Environment RAD.....	137
7.3 Shared RAD	138
7.4 Machine RAD.....	139
7.5 Class Discovery.....	141
7.5.1 Entity Classes	141
7.5.2 Interface Classes	142
7.5.3 Control Classes.....	142
7.5.4 Tri-Step Analysis.....	143
7.6 Transformation and Platform Information	144
7.6.1 Transformation Information	144
7.6.2 Platform Information	145
7.6.3 Transformation Rules	149
7.7 Summary.....	151
8.0 Moving from Analysis to Design via xMDA.....	153
8.1 Academic Application.....	153
8.2 Thematic Analysis	154
8.3 Methods	155
8.3.1 Process Oriented Systems Design (POSD)	156
8.3.2 SystemRAD	157

8.3.3 xMDA Method	157
8.4 Discussion	158
8.4.1 Tool Support	158
8.4.2 Enterprise and Distributed Processes	159
8.4.3 Design Architecture	160
8.4.4 Object Orientation	160
8.5 Summary	161
9.0 xMDA and The Club Company	163
9.1 Commercial Application	163
9.1.1 Environment RAD	165
9.1.2 Shared RAD	166
9.1.3 Machine RAD	168
9.1.4 Tri-Step Analysis	169
9.1.5 Transformation and Platform Information	171
9.1.6 Discussion	181
9.2 QVT Application	183
9.2.1 Transformation Declaration	184
9.2.2 Role2Class	184
9.2.3 IndependentActivity2Operation	186
9.2.4 Interaction2Operation	186
9.2.5 Prop2Class	187
9.2.6 rad2umlcd QVT-Relations	189
9.3 Tool Application	190
9.3.1 M1 : Machine RAD XMI	191
9.3.2 Java Transformation Rules	193
9.3.3 M1 : Class Diagram XMI	194
9.3.4 Comparative	196
9.3.5 Elaboration	198
9.3.6 Extension	199
9.4 Summary	201
10.0 Conclusions	203
10.1 Contributions	206
10.2 Related Work	207
References	209
Appendix I	228
Appendix II	233
Appendix III	265
Appendix IV	277
Appendix V	282
Appendix VI	308
Appendix VII	316
Glossary	321

List of Figures

figure 2.1.1.1, MDA viewpoint abstractions (Source: developed from Brown (2004a)).....	25
figure 2.1.3.1, elements of PIM-to-PSM transformation (Source: OMG (2003b)).....	29
figure 2.3.2.1, perspectives and aspects related to MEMO framework (Source: Frank (2002)).....	48
figure 3.1.1, overview of the research process (Source: adapted from Shelmerdine (2010)).....	63
figure 3.2.1, experimental learning cycle (Source: adapted from Kolb et al. (1979)).....	64
figure 3.2.2, framework for integrating research perspectives (Source: adapted from Braa and Vidgen (1999)).....	65
figure 4.1.1.1, technological development of Software Engineering and human interactivity (Source: developed from Brown (2004a)).....	76
figure 4.1.2.1.1, Class Diagram relating to the sample case study (Source: Wa and Leong (2004)).....	81
figure 4.1.2.2.1, Activity Diagram relating to the sample case study (Source: adapted from Wa and Leong (2004)).....	85
figure 4.2.1, the Extended Model Driven Architecture (xMDA), including pre-CIM activity (Source: developed from Bray (2004), Brown (2004a), Jackson (1995), OMG (2003b)).....	88
figure 5.1.1.1, the RAD metamodel (Source: developed from Badica et al. (2005), OMG (2006a), Ould (2004c)).....	92
figure 5.1.1.9.1, UML Activity Diagram transformation into RIVA RAD.....	109
figure 5.1.1.10.1, RAD fragment for jukebox example and associated RUD fragment.....	110
figure 5.1.1.10.2, RUD fragments combined via model merging.....	111
figure 5.1.2.1, CIM of a travel reservation system represented in BPMN (Source: adapted from Silver (2008d)).....	114
figure 5.1.2.2, travel reservation system represented in a Use Case.....	115
figure 6.1.1, MDA Viewpoints (Source: developed from OMG (2003b)).....	124
figure 6.1.1, requirements and specification included as part of the CIM definition.....	126
figure 6.2.1, systems of prime concern and development activities (Source: developed from Jackson (1995)).....	127
figure 6.3.1, xMDA framework (Source: developed from OMG (2003b)).....	129
figure 6.3.2, the integration of the xMDA with Jackson's systems of prime concern (Source: developed from Gunter et al. (2000), Jackson (1995)).....	130
figure 6.4.1.1, systems of prime concern and development activities (Source: developed from Jackson (1995)).....	132
figure 6.4.1.3, Environment RAD of Specification.....	132
figure 6.4.2.1, Shared RAD of Specification.....	133
figure 6.4.3.1, Machine RAD of Specification.....	133
figure 7.2.1, Environment RAD for the Order Processing example.....	137
figure 7.3.1, Shared RAD for the Order Processing example.....	138
figure 7.4.1, Machine RAD for the Order Processing example.....	140
figure 7.6.1.1, SimpleRAD metamodel (Source: developed from Badica et al. (2005), OMG (2006a), Ould (2004c)).....	145
figure 7.6.2.1, SimpleRAD and SimpleUML metamodels related by the rad2umlcd transformation (Source: SimpleUML developed from Appukuttan et al. (2003b), FT (2007), Jos and Anneke (2003), OMG (2008b)).....	146
figure 7.6.2.1.1, UML Class Diagram for the Order Processing example.....	148
figure 8.1.1, orthogonal notations and the software process (Source: Phalp (2002)).....	153
figure 9.1.1.1, Environment RAD for the <i>Follow Up Call</i> process.....	166
figure 9.1.2.1, Shared RAD for the <i>Follow Up Call</i> process.....	167
figure 9.1.3.1, Machine RAD for the <i>Follow Up Call</i> process.....	168
figure 9.1.5.1.1, UML Use Case Diagram for the <i>Follow Up Call</i> process Machine RAD.....	172
figure 9.1.5.2.1.1, UML Activity Diagram for the Check Prospect part of the <i>Follow Up Call</i> process Machine RAD.....	174

figure 9.1.5.2.2.1, UML Activity Diagram for the Upload Prospect part of the <i>Follow Up Call</i> process Machine RAD	176
figure 9.1.5.2.3.1, UML Activity Diagram for the Blow Out Prospect part of the <i>Follow Up Call</i> process Machine RAD	178
figure 9.1.5.3.1, UML Class Diagram for the <i>Follow Up Call</i> process Machine RAD	180
figure 9.2.2.1, a RAD role to a UML class relation.....	185
figure 9.2.3.1, a RAD independent activity to a UML operation relation	186
figure 9.2.4.1, a RAD interaction to a UML operation relation	187
figure 9.2.5.1, a RAD prop to a UML class relation.....	188
figure 9.3.1, the rad2umlcd QVT transformation pattern (Source: developed from FT (2007), Ignjatovic (2006), Koch (2006), Koch et al. (2006), Kusel et al. (2009), OMG (2006a, 2007c, 2008b), Peltier et al. (2000), Sheena et al. (2003)).....	190
figure 9.3.2, the extended rad2umlcd QVT transformation pattern (Source: developed from FT (2007), Ignjatovic (2006), Koch (2006), Koch et al. (2006), Kusel et al. (2009), OMG (2006a, 2007c, 2008b), Peltier et al. (2000), Sheena et al. (2003)).....	191
figure 9.3.1.1, VCLL representation of the <i>Follow Up Call</i> process Machine RAD.....	192
figure 9.3.2.1, Java extract showing how Pool2Class (p,c) and SubProcess2Operation (sp,o) relations are implemented in the PPT (Source: by permission from the PPT source files relating to VIDE (2010a)).....	194
figure 9.3.3.1, UML Class Diagram for the <i>Follow Up Call</i> process Machine RAD created from the application of QVT transformation rules in Java (first-cut)	195
figure 9.3.3.2, UML Class Diagram for the <i>Follow Up Call</i> process Machine RAD created manually from the application of transformation rules	195
figure 9.3.4.1, VCLL representation of the <i>Follow Up Call</i> process Machine RAD created for best representation after application of the rules encoded in the PPT.....	197
figure 9.3.4.2, UML Class Diagram for the <i>Follow Up Call</i> process Machine RAD created for best representation from the application of the rules encoded in the PPT	197
figure 9.3.5.1, UML Class Diagram for the <i>Follow Up Call</i> process Machine RAD created from the application of QVT transformation rules in Java (modified).....	199
figure 9.3.6.1, Java extract showing how the Role2Class (r, c), IndependentActivity2Operation (ia, o), Interaction2Operation (i, o), and Prop2Class (p, c) relations could be implemented in Java (Source: developed from the PPT source files relating to VIDE (2010a)).....	201

List of Tables

table 2.2.6.1, four phases of requirements driven design (Source: adapted from Castro et al. (2002))	43
table 3.2.1, comparison of positivist and interpretivist methods (Source: adapted from Gill and Johnson (1997)).....	65
table 4.1.2.1, original requirements relating to the sample case study (Source: adapted from Wa and Leong (2004)).....	80
table 4.1.2.1.1, requirements derived from reverse engineering the Class Diagram of the sample case study.	83
table 4.1.2.1.1, analysis of the number of requirements identified from the Class Diagram	84
table 4.1.2.2.1, requirements derived from reverse engineering the Activity Diagram of the sample case study.	86
table 4.1.2.2.2, analysis of the number of requirements identified from the Activity Diagram.....	87
table 5.1.2.1, travel reservation system represented in a Use Case Description.....	116
table 7.5.1, stereotype descriptions for class discovery (Source: developed from Cox and Phalp (2007))	141
table 7.5.4.1, potential design classes derived from a Tri-Step analysis of the Machine RAD for the Order Processing example	143
table 7.6.1.1, the four-layered architecture of the OMG (Source: developed from FT (2007), Kusel et al. (2009), OMG (2006a, 2007c), Peltier et al. (2000), Sheena et al. (2003), Thiemann (2009)).....	144
table 7.6.2.1, initial transformation rules to map from the RAD to the UML Class Diagram	147
table 7.6.3.1, complete set of initial transformation rules to map from the RAD to the UML	150
table 9.1.4.1, potential design classes derived from a Tri-Step analysis of the Machine RAD for the <i>Follow Up Call</i> process.	170
table 9.2.1, initial transformation rules to map from the RAD to the UML Class Diagram	183
table 9.2.6.1, complete set of QVT-Relations defined by the rad2umlcd transformation.....	189

Published Materials

- Fouad, A., Phalp, K., Kanyaru, J. M., and Jeary, S., 2011. Embedding requirements within the Model Driven Architecture. *Software Quality Journal*, 19 (2), 411 - 430.
- Fouad, A., Phalp, K., Jeary, S., and Kanyaru, J. M., 2009, 6 - 8th April 2009. *The consideration of a requirements phase in the Model Driven Architecture*. Paper presented at the Software Quality in the 21st Century, Software Quality XVII, 17th International Software Quality Management Conference (SQM), Southampton, UK.
- Jeary, S., Fouad, A., and Phalp, K., 2008, 30th June - 4th July 2008. *Extending the Model Driven Architecture with a pre-CIM level*. Paper presented at the 1st International Workshop on Business Support for MDA (MDABIZ), co-located with Tools Europe, Zurich, Switzerland.

Acknowledgements

I would like to take this opportunity to acknowledge the kind support and assistance that I received from everyone during the formulation and production of this research.

I would particularly like to thank Bournemouth University for providing me with the opportunity to study under the studentship scheme, my project supervisors, Dr. Keith Thomas Phalp and Dr. John Mathenge Kanyaru, who supported, motivated and encouraged me throughout my studies and supported my initial application, and School Research Administrator, Naomi Bailey. With her advice and guidance, I was able to produce this research report to a good standard of quality and style.

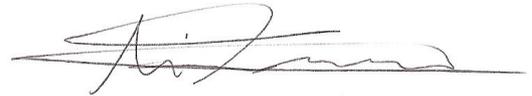
Special thanks are given to the management and staff at The Club at Meyrick Park who participated by providing me with vital research input for the content production of this paper and ensuring that time could be afforded for effective participation.

I would also like to express my appreciation to Dr. Jonathan Vincent, Jo Sawyer, Dr. Frank Milsom, Dr. Michael Jones, Dr. Hameed Mughal, Steve Bond, Siya Bhalla and Amy Louise Harris for their support in the early days of the project, and Martyn Ould (Venice Consulting), Andrew Watson (Object Management Group), Tracy Hammond (Massachusetts Institute of Technology), Scott W. Ambler (IBM Rational), Dr. Sherry Jeary, Dr. Lai Xu, Dr. Vegard Engen, Dr. Dong Ling Tong, Frank Grimm, Zoe Louise Andrews, Malik Saleh and Alexandra Nicole Turner-Piper for their input during its production, especially Keith Harrison-Broninski, CTO Role Modellers (www.rolemodellers.com) and Christie Dewell, Membership Manager, The Club at Meyrick Park (www.theclubcompany.com).

Very Special thanks are given to my friends and family. Without their advice and support, this thesis would not have been possible.

Declaration of Original Authorship

It is hereby declared that the work contained within this thesis has not been submitted to meet the requirements for an award at this, or any other, education institution, other than that which this submission is made for. The thesis contains no material previously published or written by another person, except where due reference is made. This work has not been previously presented, nor has been submitted for presentation (in full or in part) except for the academic publications outlined on page twelve.



Signed: _____

Date: 4th September 2011

List of Acronyms

3GL	Third Generation Language.
AMDD	Agile Model Driven Development.
API	Application Programming Interface.
ASSIST	A Shrewd Sketch Interpretation and Simulation Tool.
ATL	ATLAS Transformation Language.
B-SCP	Business Strategy Context Process.
BBPF	Basic Business Process Flow.
BM	Behavioural Model.
BPD	Business Process Diagram.
BPDL	Business Process Developing Life Cycle.
BPDM	Business Process Definition Metamodel.
BPEL	Business Process Execution Language.
BPEL4WS	Business Process Execution Language for Web Services.
BPM	Business Process Management.
BPMN	Business Process Modelling Notation.
BPMS	Business Process Management System.
BPR	Business Processes and Requirements.
BSC	Balanced Scorecard.
CASE	Computer Aided Software Engineering.
CIM	Computation Independent Model.
COTS	Common-Off-The-Shelf.
CSP	Communicating Sequential Processes.
CTO	Chief Technical Officer.
CWM	Common Warehouse Metamodel.
DDL	Data Definition Language.
DDM	Domain Description Model.
DFD	Data Flow Diagram.
DSL	Domain Specific Language.
DSM	Domain Specific Modelling.
EJB	Enterprise JavaBeans.
ERD	Entity Relationship Diagram.
FBCM	Fact Based Collaboration Modelling
GORE	Goal-Oriented Requirements Engineering.

GUI	Graphical User Interface.
GUIDE	Goal, Use, Investment, Deliverables, Experience/Environment.
HIM	Human Interaction Management.
HIMS	Human Interaction Management System.
HTTP	Hypertext Transfer Protocol.
ICAM	Integrated Computer Aided Manufacturing.
ICOM	Inputs, Controls, Outputs and Mechanisms.
IDEF	Integration DEFinition.
IDL	Interface Definition Language.
IPSE	Integrated Project Support Environment.
IRM	Initial Requirements Model.
ISO	International Organisation for Standardisation.
IT	Information Technology.
LOTOS	Language of Temporal Ordering Specification.
LSS	Lean Six Sigma.
MDA	Model Driven Architecture.
MDABIZ	Business Support for MDA.
MDD	Model Driven Development.
MDE	Model Driven Engineering.
MDSEE	Model Driven Software Engineering Environment.
MEMO	Multi-perspective Enterprise Modelling.
MIT	Massachusetts Institute of Technology.
MOF	Meta Object Facility.
OCL	Object Constraint Language.
OMG	Object Management Group.
ORMSC	Object and Reference Model Subcommittee.
PD	Problem Domain.
PIM	Platform Independent Model.
PML	Process Modelling Language.
POSD	Process Oriented Systems Design.
PPT	PIM Prototyping Tool.
PSM	Platform Specific Model.
QVT	Query / View / Transformation.
QVT-R	QVT-Relations.
RAD	Role Activity Diagram.
RE	Requirements Engineering.
REBNITA	Requirements Engineering for Business Need and IT Alignment

RM-ODP	Reference Model of Open Distributed Processing.
RML	Requirements Modelling Language.
RUD	Role Utility Diagram.
SADT	Structured Analysis and Design Technique.
SEAM	Systemic Enterprise Architecture Method.
SOA	Software Oriented Architecture.
SOAP	Simple Object Access Protocol.
SPL	Software Product Line.
SQL	Structured Query Language.
SQM	Software Quality Management.
STRIM	Systematic Technique for Role and Interaction Modelling.
UCDML	Use Case Description Mark-up Language.
UML	Unified Modelling Language.
VCLL	VIDE CIM Level Language.
VIDE	Visualise All Model Driven Programming.
WS-Policy	Web Service-Policy.
WSDL	Web Service Definition Language.
xMDA	eXtended Model Driven Architecture.
XMI	XML Metadata Interchange.
XML	eXtensible Mark-up Language.
XSLT	eXtensible Stylesheet Language Transformation.
YAWL	Yet Another Workflow Language

Chapter 1

Introduction

The MDA (OMG 2003b) is an approach to software development in which application code is proposed to be automatically generated from design models. The analysis and design phases of the MDA are known as the Computation Independent Model (CIM) and the Platform Independent Model (PIM) respectively (OMG 2003b). Prior experience on the Requirements Engineering unit of the Computing Masters Framework at Bournemouth University demonstrated that there may be a lack of emphasis put on the construction of the CIM within the MDA. The CIM was exposed as not being considered integral in most MDA implementations. In RE, the area identifying a defined problem is known as the problem domain (PD), to which solution systems are built to remedy that problem. Requirements are the desired effects that the solution software system is to provide within the PD (Bray 2002). The OMG describe the CIM as “the environment of the system, and the requirements for the system” (OMG 2003b) and it is therefore used to address *all* issues relating to the PD and requirements definition (Blanc 2009; OMG 2003b; Slack 2008).

Understanding the requirements of stakeholders is a difficulty within software systems development (Kappelman et al. 2006). It is identified that “social and organisation factors influence system requirements” (Sommerville 2004) and therefore, solution systems need to reflect company strategies and processes, available resources and the environment in which the problem exists. Through the extension of RE upstream, the alignment of business strategy with Information Technology (IT) is enabled by allowing for organisations to define business processes in terms of their strategic value and then be reflected in the technology (Beeson et al. 2002; Bleistein et al. 2006). Requirements modelling languages are “fundamentally different from programming and specification languages whose subject matter (software systems) is man-made, bounded and objectively known” (Greenspan et al. 1994). It is suspected that techniques natural to the fields of RE and Business Process Management (BPM) are not appropriately addressed by the MDA. Business Analysts tend to define processes informally, using simple flowcharting notations, whereas Software Engineers take such informal process notations and add further detail and abstractions to suit the engineering need. Development within the MDA involves the transformation of source models into target models, typically in the area of design (Sheena et al. 2003). To facilitate the connection between business analysis and software design, the MDA would ideally support the definition of workflows via domain specific modelling (DSM) techniques that are transferable to modelling techniques used in software development (Celms et al. 2003). The objective is for application code to be developed or generated from models that are directly informed by business.

1.1 Rationale

The sooner you start, the longer it takes (Brooks 1975).

This was Fred Brooks' vision, which still holds true today, in that time is well spent in defining requirements in the development process. If time is not invested, then more time and money is wasted in fixing problems, rewriting or maintaining erroneous code for missed or incorrectly elicited requirements (Brooks 1975; Greenspan et al. 1994; Kleppe et al. 2003; Sommerville 2004; STSC 2003; Wiegiers 2000). As computer hardware technology evolves, the requirements relating to produced software applications increase, along with an obligation for software developers to ensure that the quality of those software systems stay in line with company strategies (Beeson et al. 2002; Bleistein et al. 2006). In reality, root factors with relation to the differing ideals of software developers and business consumers create a gap in understanding and poorly defined requirements have been seen to lead to a multitude of failed projects (Bray 2002; Coughlan and Macredie 2002; Greenspan et al. 1982; Kanyaru 2006). Requirements are defined by non technical stakeholders and interpreted by technically minded developers; there is no traceability mechanism between the two (Ample 2007; Gotel and Finkelstein 1994). Communication theory relates that because these two stakeholders are from different backgrounds and have differing knowledge levels, a "lack of comprehension" (Lautenbacher et al. 2007) in design can be experienced leading to erroneous systems being developed, supported by Coughlan and Macredie (2002). This is due to differing terminologies, levels of granularity, varied models, approaches, tools and methodologies (Brahe and Bordbar 2006). It is common knowledge that companies establish, and implement strategy and that required software systems should be in line with such strategies. However, it is difficult for software developers to fully understand and implement such strategies as they are not business users. It is equally difficult for business users to develop and communicate strategies in technical terms as they are not software developers. Since "communication does not depend on what is transmitted, but on what happens to the person that receives it" (Cockburn 2007), a communication gap between the business and software analyst is highlighted.

To address this communication gap, much academic research has been directed at the MDA. The Object Management Group (OMG) provide a list of "committed companies" (OMG 2007a) regarding the application of the MDA, but they themselves are suggested to neglect the creation and transformation of the CIM (Ambler 2007; Kabanda and Adigun 2006; Karow and Gehlert 2006; Phalp et al. 2007). The specification of software systems is defined as the interface between the environment and the machine (Gunter et al. 2000; Jackson 1995), accounting for requirements and environmental concerns. The importance of PD analysis and specification is somewhat dampened by placing all related concerns under the CIM. Currently, the CIM "merely informs the decision makers about the system's context but does not influence design decisions in a functional describable way" (Karow and Gehlert 2006).

The conceptual framework prescribes an approach that should be simple for non-technical stakeholders to provide and produce artefacts that are understandable in nature to that stakeholder (Slack 2008; Soley 2006). However, the PIM and Platform Specific Model (PSM) of the MDA are complex in nature as they must contain enough detail to generate the associated code. Even some of the best developers, who might have many years in the field of Software Engineering, may have had little or no formal training in software modelling with significant costs being associated with facilitating such training in becoming a proficient modeller (Berrisford 2004; Cook 2004a; Lavagno and Mueller 2006). In literature, a good deal of attention is placed on how following the MDA can be beneficial to the development of software systems and the stakeholders involved (Brown 2004a; Hofstader 2006; Kleppe et al. 2003; Meservy and Fenstermacher 2005; OMG 2003b, 2010), but not much has been offered regarding the real costs associated with creating such systems, re-engineering models from legacy systems and the additional training that is implicated. Investment could help to ensure that business users and software developers can indeed communicate ideas coherently and that business users are not duped into thinking that the developers know what they mean by models they create at the CIM level, resulting in a project failure due to such misunderstandings (Lavagno and Mueller 2006).

The MDA vision has yet to be applied to concepts in RE (Ambler 2007; Kabanda and Adigun 2006; Karow and Gehlert 2006; Phalp et al. 2007). To help alleviate concerns, visualisation tools could be produced based upon MDA concepts and applied to RE with a business user interface, in effect allowing the business user to produce specification prototypes at the CIM level and ultimately authorise one for which software models could be built upon by developers, who might then be able to proceed to develop the system from the business user specification. A prototype of key functions is a good starting point, to “prove the architecture” (Hofstader 2006). One solution might be to establish a best practice framework based on context, allowing for user interaction to fix any errors, or limit the system to only containing distinctive components (Adler 2001). To adequately address this, it is necessary to understand the underlying impact of visualisations resulting from such a framework on the relationships between the visualisations, the underlying models, and the associated business paradigms. Therefore, an evaluation of tools and techniques within academia is presented as part of the preliminary research in Chapter 2.0. Several techniques are evaluated in terms of RE and the MDA.

The scientific motivation of this study is that RE techniques can interface well with business users but are not explicitly considered for use within the CIM phase of the MDA. The early identification of requirements models and correct transcription into the MDA is proposed to enable the alignment of software developer understanding with that of the business user, incorporating both business strategy and process in the development of software systems. Embedding requirements within the MDA is an important contribution in the field that would hopefully bring the added benefit of quality and consistency via tool support since “a human reading a paper model may be forgiving – an automated transformation tool is not” (Kleppe et al. 2003). If implemented correctly, the approach may facilitate a relatively low cost, simple method with the

potential to decrease the likelihood of project failure, provide confidence in produced products and ultimately, an overall reduction in cost and production schedule, all with a positive bearing on software quality assurance by making the MDA more accessible to business users.

1.2 Scope and Aims

In this section, the scope and aims relating to this research are outlined. Each aim is related to the detail in answering the research question below, which is the driving focus for the project:

To what extent can the MDA incorporate a requirements definition created by business user involvement within the CIM phase of the MDA to be practical in the development of software systems?

The thesis addresses this research question by targeting four aims in the context of RE directed at the notion that the current MDA definition is unsuitable for successful application in the business environment; requiring significant extension to achieve that. Each aim is discussed in turn within the subsequent paragraphs.

Aim 1: To examine the definition of the CIM within the MDA and consider the appropriation of it as an interface with the business user for defining requirements in MDA notations.

The notion of *learning to walk before you run* is one that can relate to the MDA. This is because the majority developers have not yet even begun sketching in the Unified Modelling Language (UML), let alone developed the art of creating sophisticated models using such tools that are required by the MDA (Uhl and Ambler 2003). In industry, "UML compliance is not as important as the business value" (Staron and Wohlin 2006), however, compliance is essential to transformations of the MDA. Very little industry-wide information is available to support the use of the MDA beyond the academic domain (Mattsson et al. 2009) and generally, those that do, report on implementations that have been employed and described according to particular Software Engineering situations. Whilst there are many attempts to facilitate the application of the MDA with explicit attention to the CIM (Casallas et al. 2005; Debevoise and Smith 2009; Garrido et al. 2007; Kherraf et al. 2008; Leonardi and Mauco 2004; Martin and Loos 2008; Osis et al. 2007; Poernomo et al. 2008; Rech and Schmitt 2008; Rodriguez et al. 2007a; VIDE 2009; Zhang et al. 2008), they appear not to be implemented to any substantial extent in defining software systems; *significant* literature is unavailable in supporting a successful implementation of a requirements definition within the MDA. This aim looks to examine the extent to which notations of the MDA are sufficient for capturing business requirements within the software process and how accessible they are to the business user in defining requirements.

Aim 2: To discover how other modelling techniques which are accessible to the business user, might be integrated with the MDA in terms of method and notation, with the focus on transformation and traceability.

Analysis relating to discoveries made in the previous aim may suggest that MDA techniques are somewhat inaccessible to business users because generic languages such as the Business Process Modelling Notation (BPMN) and the UML are not necessarily applicable to every PD (Jouault and Kurtev 2006; Mattsson et al. 2009; Rombach 1988). Therefore, there may be a need for the MDA to be open to the usage of any number and combination of alternate tools and techniques in defining the requirements of a software system. This aim investigates proposed solution mechanisms to integrate requirements with the MDA, specifically in terms of CIM-to-PIM and CIM-to-CIM transformations. For example, to discover how other non-software related methods, such as the Role Activity Diagram (RAD), might be better at successfully capturing requirements, defining specifications and contributing in support of existing software models from a RE standpoint within the CIM and PIM phases of the MDA. The selection of notations will be guided by the literature and comparable to those utilised by MDA phases.

Aim 3: To extend the framework of the MDA to account for specification within the CIM.

It is suggested that the definition of the MDA does not include any "precise rules or guidelines explaining how Software Engineers can use" the CIM, PIM and PSM (Garrido et al. 2007), supported by Kim (2008), Wood (2005). Neither does it give consideration to the advantages that the Business Analyst and Software Engineer may gain from specific guidelines for accessibility to the architecture in terms of both framework and method. In support of findings made from previous aims, it is suggested that, by extending the definition of the MDA framework to facilitate RE, benefits would be gained with regards to the overall user experience and the quality of developed systems. Aim 3 is directed at the justification and description of mechanisms to extend the framework of the MDA, by fusing RE techniques with those of the MDA to form a bridge between business and software use. The extension is to be produced with recommendations that are proposed to enhance both Business Analyst and Software Engineer understanding of CIM development and facilitate an unambiguous specification at both levels, thereby embedding requirements within the MDA.

Aim 4: To determine the academic and commercial value of extended mechanisms.

The final research question is driven by the necessity to establish the value and accuracy of any extensions suggested in resolution of the previous aim. Since the CIM is the founding phase of the MDA, it is important to know that it is formed in the correct perspective. Such perspective is relative to the modeller and that which is required to be modelled (Brown 2004b). Findings from previous aims may support the argument that the correct perspective of the CIM ought to account for Jackson's specification (Jackson 1995); a detailed definition based on the foundations of requirements elicitation. The important discovery is a solution that is

not only directly useful to the Business Analyst in being adaptable to RE techniques of elicitation, but also valuable to the Software Engineer in the construction of the PIM in an appropriate modelling notation (such as the UML) for use within the MDA. In order that any extension to the MDA might facilitate real system implementations motivated directly from the input of the Business Analyst, the rigour of the mechanisms described in part of the resolution of aim 3 is to be explored in both academic and commercial settings, determining whether they are viable in comparison with alternative techniques, accessible to business use and applicable to commercial processes and MDA tools and techniques. This is imperative to underpin the worth of the research in support of findings made in achieving aims 1 to 3.

Although fertile ground for investigation, issues relating to the real cost of MDA implementations, the applicability of MDA tools, MDA alternatives and vendor lock-in within the MDA are considered to be beyond the scope of this investigation because they are on the fringe of the area to which this research gives focus. Chapter 10.0 gives direction to follow-up work relating to this research and discusses these areas in further detail. It is important to also consider that the scope and methodology pertaining to this research is orchestrated with direct relation to time and cost constraints imposed on the project. A research overview is provided in Chapter 3.0 to that effect.

1.3 Report Structure

This research begins with an examination of the current state of the art with respect to aligning the needs of Requirements and Software Engineering. A research overview is discussed and a proposed methodology determined in Chapter 3.0. Chapters 4.0 and 5.0 outline discoveries made in consideration of aims 1 and 2 and provide discussion concerning those discoveries. Chapters 6.0 and 7.0 demonstrate how aim 3 is achieved by introducing an enhancement to the MDA via extension mechanisms to support RE techniques. Chapters 8.0 and 9.0 are concerned with assessing the application of the proposed enhancements in an academic and commercial context, thereby addressing the last aim. In Chapter 10.0, final conclusions are drawn in relation to the value of the research and any difficulties encountered, with due consideration to the project direction and scope.

Chapter 2

Literature Review (State of Art)

In 2003, the MDA emerged from the international trade association, the OMG. Administrators of the UML with affiliates around the world, the OMG describe a framework which implements a separation of “business oriented decisions from platform decisions” (Brown 2004a), supported by Blanc (2009), Kabanda and Adigun (2006), Slack (2008). The MDA utilises models as integral artefacts for development and deployment, and transformations between those models. A model is a high-level abstraction of a software system, below the model is the implementation, which is used to interface with hardware components via the operating system. Since business logic is defined in business models, the objective of current research is in looking at ways to transfer business logic into that of IT; ensuring that business logic is represented concisely and consistently in sync with the business model (Koehler et al. 2002). In this section, the natural fault-line between software development in the MDA and the business perspective of such development is addressed by examining the available literature and discussing any inconsistencies uncovered.

2.1 The MDA Prescription

The MDA is presented as a way forward in the development and implementation of software systems (OMG 2003b). The key principles for the MDA are the integral use of models, transformations between model abstraction layers, the description of such models via metamodels and associated industry standards, which are geared to the preservation and leverage of existing technologies (Slack 2008). Model manipulation and transformation is *vital* for the MDA to reach its full potential (Appukuttan et al. 2003b; Ignjatovic 2006). Grounded in software knowledge, the MDA provides a “conceptual framework and set of standards” (Thangaraj 2004) for a particular software development style (OMG 2003b). Much has been written about the success and benefit of the MDA (Brown 2004a; Hofstader 2006; Kleppe et al. 2003; Meservy and Fenstermacher 2005; OMG 2003b, 2010), but strong cause for concern regarding MDA application is raised in other literature (Berrisford 2004; Brown 2008; Cook 2004a; Haan 2011; McNeile 2003; Thomas 2004). Therefore, care and scepticism is suggested to be applied regarding claims made of the realisable value of the MDA before it can be established as the new frontier for software development.

2.1.1 Viewpoints

Within the MDA are several different viewpoints, each an abstraction of the previous. They are known as the Computation Independent Model (CIM), Platform Independent Model (PIM) and Platform Specific Model (PSM). These are the prescribed models to be used within the MDA (OMG 2003b). Transformations and mappings are used to translate a model “from one level of abstraction to another” (Brown 2004a) until code is generated for the designated platform defined by the PSM. Figure 2.1.1.1 has been developed as part of this research to facilitate the understanding of these viewpoints, extending a similar model that abstracts from assembly languages to the MDA in Brown (2004a).

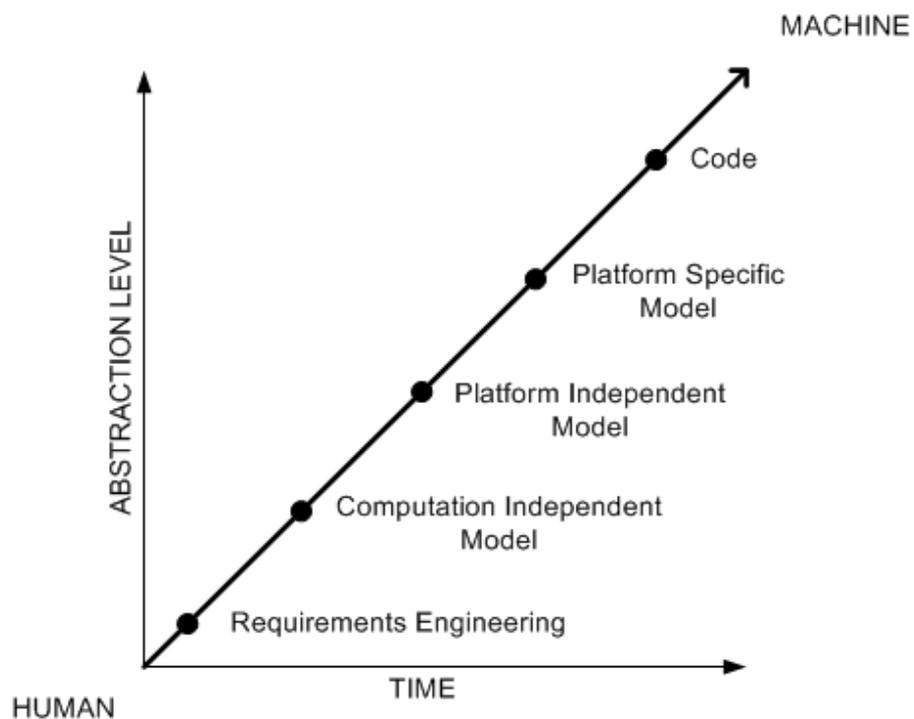


figure 2.1.1.1, MDA viewpoint abstractions (Source: developed from Brown (2004a)).

The essence of MDA is the independence and separation of technologically specific platforms (beds of functionality) (Blanc 2009; Brown 2004a; Kabanda and Adigun 2006; Slack 2008).

One of the best ways to combat complexity of software development is through the use of abstraction, problem decomposition, and the separation of concerns (Sendall and Kozaczynski 2003).

With focus on the PIM, a system can theoretically be developed without the definition of a particular platform. This enables models to be transformed into implementations of *any* required technological platform,

hardware or software. By following the MDA, investment returns are expected to flow “from the reuse of application and domain models across the software lifespan” (Blanc 2009); the biggest challenge of the architecture being how interoperable solutions can be made (Blanc 2009).

Whilst the MDA concentrates on models of those viewpoints and transformations between them, no phase appears to give sufficient consideration to RE (Ambler 2007; Kabanda and Adigun 2006; Karow and Gehlert 2006; Phalp et al. 2007). The separation of interest ought to put “the Business Analyst in a unique and potentially powerful position within an organisation” (Slack 2008) because through clear and concise CIM definitions they could affect design and implementation. Indeed, the significance that is placed on upstream stakeholders and related models reveals an expectation that MDA tools should be directed at the level of the Business Analyst. An analysis of tool support was conducted by Phalp et al. (2007), finding good support for stakeholders at the PIM and PSM level. However, support for the CIM level processes was inadequate and CIM-to-PIM transformations were unavailable (Phalp et al. 2007). One important consideration is that “there is no single model of a process” (Ould 2004c), implying that multiple CIM abstractions may be required to fully realise a complete business process. It is also difficult to know that the CIM which is the subject of scrutiny, is formed in the correct perspective, since perspective is relative to the modeller and what is required to be modelled (Berrisford 2004; Brown 2004b). Further to this, every model is an incomplete *representation* of some reality (Berrisford 2004; Thomas 2004). Representation is emphasised here since no model can ever equal the reality of the situation (without becoming that reality). The acceptance that a model can equal code is something that is being neglected in research into transformations (Brown 2004a) and could ultimately demonstrate an inherent MDA flaw. If a model (albeit CIM, PIM or PSM) could truly represent code, then the need for that code (or subsequent modelling phase), could be questionable. The MDA definition of the CIM does not prescribe any particular abstraction or guidance on abstraction and therefore neglects these considerations (Garrido et al. 2007; Kim 2008; Wood 2005). Furthermore, since concern in modelling is directed importantly at semantic, rather than syntactic issues, a difficulty is presented in ensuring the alignment of understanding between the Business Analyst and Software Engineer; hence, figure 2.1.1.1 extends the MDA viewpoints with the inclusion of RE as a phase prior to the CIM to interface the business user with the architecture.

2.1.2 Standards

The standards described by the OMG to form the basis of the MDA include the Meta Object Facility (MOF) (OMG 2006a), XML Metadata Interchange (XMI) (OMG 2007b), Common Warehouse Metamodel (CWM) (OMG 2003a), Object Constraint Language (OCL) (OMG 2006b) and Query / View / Transformation (QVT) (OMG 2008b). The OMG also offer support for definitions in the BPMN (OMG 2005, 2008a) and the UML (OMG 2007c). There are two key issues surrounding the UML. It is firstly, too convoluted for non-technical users to understand and implement and secondly, driven by the need to design complete systems upfront

before any coding can take place; this hinders design and implementation agility (Ford 2009). The UML and MOF standards do not really facilitate other mechanisms, such as DSM tools, to be implemented, which limits the scope of the MDA (Cook 2004a, 2004b). Such tools are useful across many disciplines and facilitate understanding between involved parties. Moreover, the UML does not have sufficient *precision* to enable a complete PSM, let alone code generation (Meservy and Fenstermacher 2005). Elements of some languages simply cannot translate from UML (e.g. a UML class does not translate into a C# class as the UML does not allow for *properties* in the way that C# does. Similarly with Java interfaces, *static fields* are allowed but with UML this is not a supported provision (Cook 2004a; Meservy and Fenstermacher 2005). A domain specific schema for that language would be more effective than the MOF standard as this also restricts usage in industry (Celms et al. 2003; Cook 2004a; Frank 2002) and it is for that reason the MOF is said not to be supported by Microsoft (Cook 2004a). The only example of MDA in action is from the J2EE platform for PIM-to-PSM mappings (Cook 2004a), which is maybe why vendors (such as Microsoft) bend the rules on the usage of the UML to, for example, support .NET mappings. Being grounded by the UML and MOF inspires companies to mutate UML, stretch the logic or add completely new elements to the UML toolkit at the PIM level. As previously noted, the PIM is defined so that it is independent of any hardware or software platform. UML is being mutated across the board to enable fluid transformations and mappings (Ambler 2007; Cook 2004a; Koch 2006; Tratt 2005). This in turn can cause models to be locked into the application program that produced the original PIM (known as vendor lock-in) and it is not a simple process of transferring a PIM created in one program into another program to create the PSM (see Berrisford (2004) for further discussion surrounding this concern). Once an understanding is gained on how a particular software program might adapt UML paradigms, catastrophic consequences might be realised if the code is generated in an entirely different program that generates a PSM from an incorrect (or seemingly correct) PIM. Furthermore, if such ambiguities remain hidden, it could be far into the implementation phase before any mistake is realised. It therefore holds that whilst being useful, the "UML and MDA code generators are... not the panaceas that some would have us believe" (Thomas 2004).

A number of software development styles have embraced MDA techniques to differing degrees. They include, but are not limited to, Agile Development, Service Oriented Architecture (SOA), Extreme Programming and The Rational Unified Process. An example is given in Thangaraj (2004) of Cancer Bioinformatics Infrastructure Objects, whereby instrument data is made available via services provided following a MDA. UML models are encoded in XMI files and are then manipulated by open source tools to generate Java, SOAP, HTTP and PERL APIs (Thangaraj 2004). A further example combines the MDA with an optimisation framework for the rapid construction of e-business software systems, which characterises a knowledge structure for the reusability of knowledge gained on prior projects (Yoda 2001).

2.1.3 Transformations

Model transformations are defined as those which take “one or more source models as input and produce one or more target models as output, following a set of transformation rules” (Sendall and Kozaczynski 2003). In the MDA, one or more PSM is commonly created from a PIM via transformation, which is the “key technology in the realisation of the MDA vision” (Appukuttan et al. 2003b). The UML is more amenable to software developers and tool support than BPM, and therefore, transformation is viewed to connect business process techniques with model driven development (Macek and Richta 2009). The vision is to automate much of these transformation processes in order that the benefits are reaped. Further to this, “with a large repository of model transformation descriptions at ones disposal, it follows that it may be desirable to combine existing transformations to build new, composite ones, since it is sometimes easier to compose components rather than build something from basic particles” (Sendall and Kozaczynski 2003). A Model Driven Software Engineering Environment (MDSEE) can be implemented to support developers of the MDA with model and metamodel access; model transformation; simulation; process; and project definition (Blanc 2009). The emphasis of the MDSEE should be on evolving, *living* models and metamodels to support the life cycle of models (Blanc 2009). Examples of the MDSEE include ModelBus and Praxis (Blanc 2009).

There are two different approaches to an MDA transformation. They are either conducted *manually*, using profiles, patterns and markings to provide additional details or *automatically* (where the PIM is considered computationally complete). A typical manual transformation would involve taking a PIM and adding platform specific detail to it which can in turn evolve into fully executable code. *Markings* are used to trace information within a transformation and a set of *marks* can be contained within a *marked* model (OMG 2003b). With automatic transformations, the PSM may appear transparent to the user (McNeile 2003) since the PIM is transformed directly into Code, this is because there is no requirement on the user to ever adapt the PSM since the PIM and transformation is computationally complete. This is illustrated in the Eclipse modelling tool, where it appears that the PSM is not represented, and therefore brings into question the necessity of the PSM phase in such an implementation. However, a transformation language known as Operational QVT is specified and can be used with models of the Eclipse Modelling Framework to cater for “model modification and transformation” (Boyko et al. 2009). A transformation record may also be resultant that details which PIM elements mapped to which PSM elements (OMG 2003b). The essential elements of PIM-to-PSM transformation are illustrated in figure 2.1.3.1.

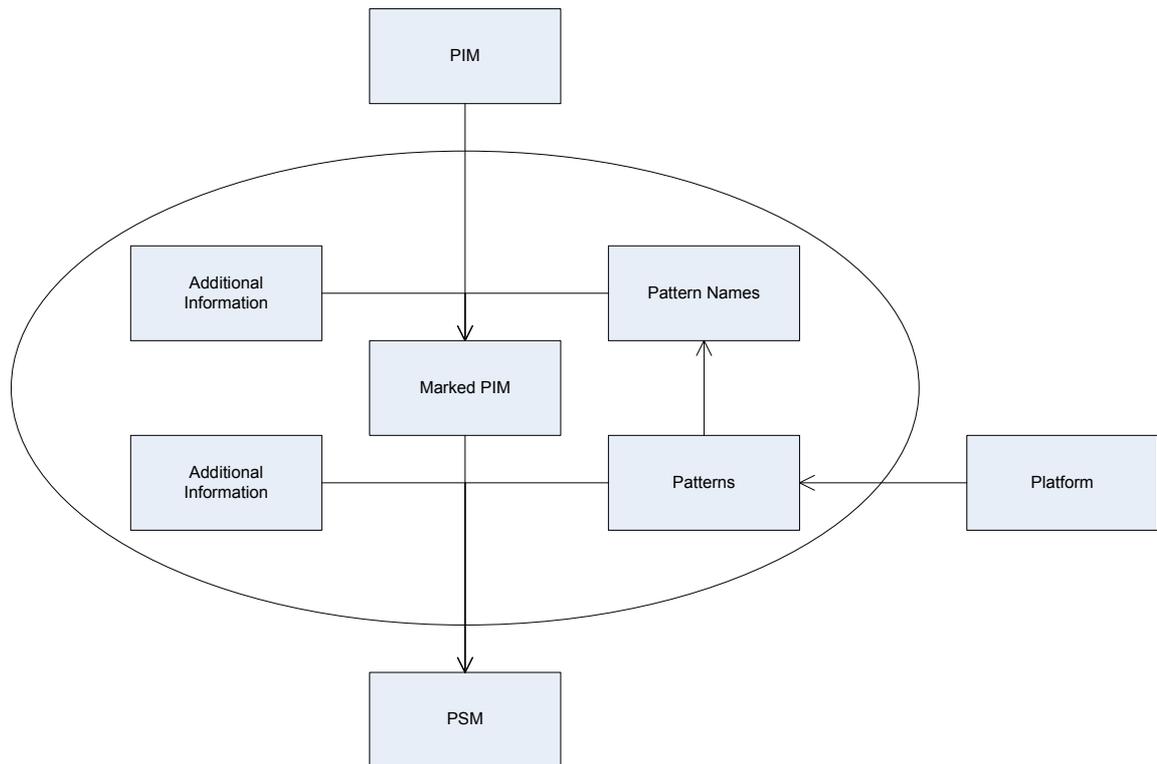


figure 2.1.3.1, elements of PIM-to-PSM transformation (Source: OMG (2003b)).

In Sheena et al. (2003), effort is made to extend the transformation mechanisms outlined by the OMG standards whereby a pattern-based model refactoring technique to describe UML transformations at the meta-level is offered. This technique is suggested to promote transformation reusability at the meta-level and align both MDA and QVT (Sheena et al. 2003). Here, patterns are used to raise the level of abstraction on transformation so that families of pattern-based transformations can be described for a model set, and not just simply directly on a model-to-model basis. In the approach, a Source Pattern is offered as a metamodeling extension on the UML metamodel that supports the Transformation Pattern; The Transformation Pattern is a metamodel supporting the transformation between Source and Target Patterns; and the Target Pattern is an extension on the UML metamodel that supports the Transformation Pattern (Sheena et al. 2003).

The two transformation approaches highlighted two distinct points of view regarding transformation implementation. First termed by Stephen Mellor, who was also present at the birth of the *Agile Manifesto*, they are the *Elaborationists* and the *Translationists* (McNeile 2003). Elaboration is demonstrated when it is the intent of the modeller or developer to produce a template code structure at the PSM level as a result of transformation, whereby it might be manually updated, and/or tweaked before the final code is generated. Even then, the final code may also be tweaked. This creates a problem of model synchronicity, with the solution being to ensure that the models and code develop together, so if the code is tweaked, the PSM is also

tweaked, and so on (McNeile 2003). This is important since “out of date documentation is worse than none at all because it actively misleads” (Ford 2009). *Translationists* believe that all of the detail required for a transformation should be included in the automated transformation and therefore there should be no requirement to ever see the PSM, since the transformation will produce the output as required. Any changes to be made need only be made at the PIM level or with associated transformation rules. In the ethos of Agile methods, models are considered equal to code, and therefore conform to the *Translationist* ideal (Mellor 2004). Whilst this divide exists, the development methodology appears to not yet be mature enough to facilitate mainstream translational MDA. For now, the majority of MDA implementations remain in elaboration.

It is highlighted that “the same approaches that enable transformation of a PIM to a PSM can be used to transform any model into another, related model” (OMG 2003b), which makes it all the more intriguing to consider why the MDA gives focus to PIM-to-PSM transformations, and not vice versa, or ones involving the CIM. In fact, it is suggested that such transformations may not be possible (Koch 2006; Meservy and Fenstermacher 2005). Even though the RE inclusion appears an important aspect of software development, much is left open regarding upstream transformations (Kherraf et al. 2008). It is thought here that, without traceability mechanisms between the CIM and PIM, each downstream transformation would result in a loss in upstream richness with important requirements becoming lost in translation. If a move in this direction were to be made within the MDA, a better understanding of legacy systems may be made because technical documentation could be deconstructed and presented in a format for which the Business Analyst has a greater appreciation of, and vice versa, thereby bridging the gap between business and technological processes. There are two viewpoints when specifically considering upstream transformations. The first is where business process models could include enough detail to generate code directly (Havey 2007). Such transformations are commonplace in the SOA. The second, being more typical of the MDA, is where business process models could be transformed into computational models with traceability mechanisms to ensure originating requirements are not lost, which can then facilitate the addition of detail and transformation into downstream models or code. It is agreed that design cannot be viewed as “a mere transformation of analysis models into software solutions... the input of a transformation has to be complete and all information must be significant for the transformation... by resembling the structure of real-world perceptions without regarding software quality requirements, the transformation results are potentially insufficient and unusable in the design process” (Karow and Gehlert 2006). Therefore, focus ought to be on the second viewpoint of upstream transformations as they address the difference between what is computational and what is not; code not being produced directly from analysis models, rather such models are used to influence computational ones.

2.1.4 Specification and the CIM

The MDA is the OMG's proposed solution to issues of portability, interoperability and reusability in the development of software systems (OMG 2003b). In RE, specification is used to reconcile the differences between what the business user requirements are via analysis and what is required by design in the software process. However, the MDA has no explicit mechanism to capture specification or account for collaborative interactions which are natural to the business process. Specification is rooted in the behaviour of a system which does not yet exist (Bray 2002), therefore the concept of specifying an entirely new system encourages the designer to be more creative and inventive when producing a specification that will influence design. This allows certain decisions to be made about the requirements, the vision the designer has of the new system, so long as the client requirements have ultimately been met by the solution system (and the client is happy about decisions made, and perhaps therein lays a problem).

In the UML, specification is, at best, delivered as a Use Case diagram and/or Use Case Description representing some form of business activity. "Use Cases are part of UML and offer a foundation or starting point for using models" (Hansz and Fado 2003). In a Use Case specification, components are broken down into the relevant *users* (or actors) of the system and the *tasks* completed during system interaction (Stevens and Pooley 2000). Use Case diagrams are useful in RE, mainly to capture which actors are required to interact with the system, and for each, which specific tasks are required and, which tasks form part interactions (Stevens and Pooley 2000). They are a common technique for specification because actors are presented in the system context via natural English (Kanyaru and Phalp 2005). However, others argue that they "are commonly expressed as hard-to-read text documents, containing a mix of natural language statements, semi-standard names and expressions and raw cross references" (Golbaz et al. 2008). With a Use Case, it is very difficult (if not impossible) to show relationships and dependencies between event flows (Kanyaru 2006), non-deterministic and parallel processes, choices that may be made to determine process flow, the order of which occurrences may happen, and events which contain loops. The Use Case actor is a simple notion which is not rich enough to represent other important business notions, such as a role. The sequencing of numbered events in Use Case descriptions only shows one distinct instance in which the events may occur and prescribes a specification that is to some degree set in stone. It is important to understand that the order in which events may occur for any relevant actor may not be the only way for that actor to complete the task. A number of *candidate* designs may be produced and it is the role of the developer to distinguish the difference between design and requirements issues, focusing on specification (Stevens and Pooley 2000). This illustrates that there is an overall loss of *richness* of information and a lack of control constructs in the Use Case diagram alone. This limitation has led some to augment Use Case descriptions with state based information (in the form of pre and post states for every event) (Kanyaru 2006; Kanyaru and Phalp 2005; Kanyaru and Phalp 2009). The concept of Use Case descriptions is to show a simplified, more practical description of the PD. However, it is suggested here that the addition of state based information could complicate the process

and increase ambiguity. In reality, business users have little time or inclination to spend viewing and reviewing model complexities and communication is suggested to be enhanced via informal approaches (Phalp 1998). It is highlighted that “the invention of the behaviour is part of the specification task” (Bray 2002). This behavioural description is commonly the natural language based rules of Use Case descriptions, of which there are already many different methods and templates (Phalp et al. 2011), which could lead to further ambiguity and misunderstanding. Use Cases can lead to a specification which is divorced from the system architecture, focussing rather on producing Use Cases in the relevant context rather than the object oriented nature or view of system design. This can lead to problems later in the design and implementation stages whereby, it can be difficult for developers to modify or add additional functionality to systems as they have not been designed in a nature that promotes it; Use Cases have an entirely more *functional* view in comparison with other methods. Furthermore, when using Use Cases to model the specification of a new system, it can, in practice, be very easy to miss important requirements (Stevens and Pooley 2000) and it may be very easy for an inept developer to invent requirements, which of course might get overlooked in the development process. A solution proposed to “overcome most... deficiencies” (Issa et al. 2005) of the Use Case is provided in Issa et al. (2005) by defining the Use Case via a metamodel; which is an amalgamation of available... techniques (textual descriptions / tabular descriptions / activity diagrams). However, the technique is not very well described and focuses only on a brief Use Case model, rather than a full specification; project size and effort is not really addressed at all. The eXtensible Mark-up Language (XML) is “becoming the *lingua franca* for data communication between applications” (Peltier et al. 2000) and an attempt to formalise Use Cases in XML is presented by the Use Case Description Mark-up Language (UCDML) in Golbaz et al. (2008). UCDML represents “a UML compatible template utilised for the documentation of Use Cases” (Golbaz et al. 2008) in a universal language representation of the XML. This supports for the portability and interoperability of Use Case definitions and capitalises on strong tool support for the Extensible Stylesheet Language Transformation (XSLT) technology (Golbaz et al. 2008). If XSLT can be applied to create transformations between notations, the language could have real implication, especially in consideration of deriving Use Cases from other notations.

The OMG suggest that the primary user of the CIM has inadequate knowledge of models or modelling concepts (OMG 2003b). However, collaborative modelling has been rife in the business arena for decades. The fact is, behavioural modelling notations, such as the UML Use Case and Collaboration Diagrams are not mainstream artefacts of MDA, nor are they “suitable for code generation or model execution” (McNeile 2003), let alone those notations more akin with business objectives. The OMG write that in the MDA, “CIM requirements should be traceable to the PIM and PSM constructs that implement them, and vice versa” and “developed models can... be validated against requirements” (OMG 2003b). Whilst idyllic, this is not the case since no complete mechanism is in place to facilitate RE concepts and the traceability between them and those of the MDA. The UML is “designed for the development of software systems” (Frank 2002), and does not provide such concepts and graphical notations as are central to BPM tools and techniques which are

accessible to business users. This highlights the problem that the MDA is falling short of expectation. The MDA vision of “automatic transformation” (OMG 2003b) has yet to be realised in the realm of real business solutions. The promise of “machine readable application and data models” (OMG 2003b) is the driving force behind MDA, and by definition, data-centric models may not be a useful way forward in modelling and delivering software systems. The “proper management and coordination of the interactions among humans and between humans and computerised tools are critical and complex activities” (Conradi et al. 1992). Information systems currently do not allow for the flexibility required by human-driven processes (Basson 2009b). “For a system to be successful, that is to be human-centric and process-oriented, it must provide a rich set of features that support automation and adaptation of human interaction with processes” (Basson 2009b). With sufficient modelling knowledge of such business processes, it is feasible that a heavy weight prototype specification could be produced and be more effective at the CIM level.

The OMG propose that MDA decisions are based on both “business and technical considerations” (OMG 2003b). However, it has so far been seen in this chapter that whilst the MDA might be a step in the right direction in development, focus is given on software paradigms at the expense of those related to business. The CIM ought to be used to realise user requirements and functionality, hence the need to explore concepts of RE and BPM in consideration of the CIM. If business process exploration, requirements definition and the CIM are left vague, progress will be difficult to make.

2.2 The Business Perception

As previously noted, requirements in the MDA are resolved in the CIM. Theoretically, code is generated as a direct outcome of MDA model artefacts and, therefore, much importance is laid upon the CIM. It is here where the requirements are first met by computational models and any mistake in the CIM will have ramifications for the implementation. The important thing is that the ordinary business user must be able to understand, validate and apply the CIM, so that requirements are delivered correctly in the final software implementation.

“Understanding user requirements is an integral part of information systems design and is critical to the success of interactive systems” (Maguire and Bevan 2002). It is clear that if there are many misunderstandings, assumptions and perhaps poor elicitation then the result will be a poor quality, or incorrect specification, and since specification may form the basis of a contractual agreement between companies, the importance of clarity is evident. After all, “software is intended to change or guarantee real-world conditions in accordance with the requirements” (Cox et al. 2005b). It is suggested here that business applications extend into the business domain beyond those proposed by the MDA and may help alleviate the concerns raised in

Section 2.1 regarding the MDA because “current CIM modelling notations are often biased towards the mindset, paradigms and constructs of the software domain” (Phalp and Jeary 2010).

In this chapter, literature pertaining to the business perception of software development is investigated with a view to discovering the extent to which business modelling techniques and applications may be useful in the context of software development and, more importantly, within the MDA. However, “modelling techniques are like sand on the beach. They seem to exist in millions of variants, fashions, and styles” (Recker 2006) and have many different uses in the development of software systems (Celms et al. 2003; Lehman 1989). In an effort to document the number of available process modelling techniques, one researcher is quoted as stopping “at the count of 3000” (Recker 2006). In addition to this, some techniques are expected to be better than others at managing context specific concepts. It is therefore clear that the focus of this chapter must be limited to those that appear to offer the most promise or hold the most interest for investigation in consideration of the application the MDA arena as discussed in Section 2.1.

2.2.1 Business Process Nature

In consideration of techniques used in RE and BPM, it is important to have an understanding of the nature of business processes. “Business process modelling is an important phase during requirements collection” (Badica et al. 2005). Before the rise of information systems, flowcharting methods (c1920) were used to describe procedures for internal use and for quality management. Once the IT world became dominant, focus shifted away from modelling the business process in terms of that process to modelling information requirements in order to build information systems (for example, the Data Flow Diagram (DFD) and the Integration DEFinition (IDEF) - c1970s, etc) (Bushell 2005). Information based workflows (c1980s) gave foundation to BPM (Kemsley 2006) and now, with the realisation of greater benefits from directing attention to and enhancing the business process, focus has again returned the management of business processes and modelling in terms of them (Bushell 2005; Ould 2004c).

With greater need in business for emphasis on processes and change (Bushell 2005; Kavakli 2004; Ward-Dutton and Baxter 2009), three challenges in this transition period are identified by Ould (2005). Firstly, IT systems must change as business processes are updated. “As business gets more interested in its processes, so it gets more interested in the alignment of its computer systems with the processes they are supposed to support” (Ould 2005). In 2005, the first International Workshop on Requirements Engineering for Business Need and IT Alignment (REBNITA) was hosted at the Sorbonne, in Paris, which held alignment as the central theme. In the introductory notes of the proceedings for REBNITA it was suggested that “it is no longer possible to consider IT separate from the business organisation it supports, and hence requirements engineering should address the business needs of an organisation” (Cox et al. 2005a). Such alignment must occur as business requirements change, and indeed processes change and therefore systems must be reactive

to the environment in which they are deployed in, rather than being static and focussed on data (as might be presented in a Class Diagram). Alignment is difficult because of opposing perspectives; business is about process and change, IT is about information. "As organisations mature in the Process Age, they will distinguish themselves from those that rely solely on information based concepts to build a competitive advantage" (Basson 2009a). Therefore, systems need to ideally be founded first and foremost on the processes which require them, not the information involved. A change in process brings about a change of what information is required, and where such information might be needed. Requirements management is concerned with "planning and change management" (Sommerville 2004) and is considered essential.

Secondly, it is suggested that information systems are being sold to business users as Business Process Management Systems (BPMS); a problem which may be inconsequential in the larger scheme, yet deserves some consideration. An example can be drawn from the structure of the MDA, defined by the OMG (2003b), where much focus is given to software, rather than business process development tools. Such tools are not designed to be reactive to the dynamics of the business environment. The search is for new technologies that can manage these concepts.

Thirdly, a process architecture needs to be developed that purposefully and efficiently separates organisational activity processes along "natural cleavage lines of that activity" (Ould 2005), accounting for strategic change. Immediate technological and cultural change should not affect the process architecture. The idea behind BPM is to provide the business analyst with the management philosophy, method and technology to facilitate business model flexibility, product/service innovation and operational efficiency/quality (Ward-Dutton and Baxter 2009).

2.2.2 Workflow Management

Initially, the Workflow Patterns Initiative (1990s) had "the aim of identifying generic recurring constructs in the workflow domain and describing them in the form of patterns" (Russell 2007). With further advancements, a multitude of workflow systems arrived and in 1993 the Workflow Management Coalition formed with the objective of standardising the workflow arena (Russell 2007). This led to the Workflow Reference Model which addressed the need for solutions that were interoperable (Russell 2007). In consideration of modelling business processes and the design of supporting software systems, there is a "shift from data to process orientation and... [a] focus on process-aware information systems" (Wohed et al. 2006). This shift has led to languages aligning to support the behavioural nature of the business process (such as the introduction of Activity Diagrams to the UML) and to the influx of a multitude of new languages (such as BPMN; BPEL4WS) built specifically for the business process task (Wohed et al. 2006).

A “reference analysis framework” (Wohed et al. 2006) known as the Workflow Patterns framework is available from www.workflowpatterns.com. This framework is “fine-grained” and provides over 100 patterns that address three different areas for evaluation which are control-flow; data and resource (Wohed et al. 2006). Several recognisable patterns are defined in six categories, which can be used as the basis for a comparative framework to investigate BPM technologies. Patterns are “universally applicable solutions to the complex process problems that BPM projects encounter daily” (Atwood 2006) and can be used to review process strategy in order to ensure that they represent the best way of doing things via a “process walk-through” (Atwood 2006). The six categories and associated patterns are outlined below.

- Basic Control (Sequence / Exclusive Choice / Simple Merge / Parallel Split and Synchronisation).
- Advanced Branching (Multiple Choice and Synchronising Merge Patterns / Multiple Discriminator and N-out-of-M Join Patterns / Multiple Merge Pattern).
- Structural (Arbitrary Cycles Pattern / Collaboration / Implicit Termination Pattern).
- Multiple Instances (Without Synchronisation / With Design and or Runtime Knowledge Patterns).
- State Based (Deferred Choice / Milestone).
- Cancellation (Cancel Activity / Case).

The Workflow Patterns framework is purported to be “the most comprehensive framework in existence” (Wohed et al. 2006). Workflow patterns in Business Process Modelling can produce advantages in terms of reusability and a pattern is recognised as a composition of one or many defined patterns (Thom et al. 2007). However, several problems have been identified and associated with such BPM patterns by Atwood (2006). They are easily mistaken for object oriented patterns, intimidating for business users due to a perceived complexity surrounding them and glorified by some as an all inclusive solution (Atwood 2006).

Yet Another Workflow Language (YAWL) is a pattern-based language. In comparison with the YAWL model, Wohed et al. (2002, 2005, 2006) examine the UML Activity Diagrams, BPMN, BPEL and BPEL4WS specifications, finding that several patterns are not supported by these notations. For example, the BPMN “provides direct support for the majority of the control-flow patterns and for nearly half of the data patterns, while support for the resource patterns is scant” (Wohed et al. 2006). Furthermore, other solutions are said to represent a *workaround* that sufficiently deviates from the pattern to invalidate the conformance; such workarounds may actually conform to the pattern, depending on interpretation. This argument is extended by examining the advanced patterns of the analysis framework. Wohed et al. (2002) suggest that “the patterns referring to more advanced constructs are often poorly supported in the different languages” (Wohed et al. 2002), which perhaps highlights that the advanced patterns of the framework may not be suitable for analysing the usefulness of a notation or language since it could be arguable that the advanced patterns are not needed by all languages, or that advanced patterns specific to the notation or language in question remain undefined in the definition of patterns.

In Russell (2007), 126 patterns were identified and defined with a formal reference language known as newYAWL being proposed for “business process modelling and enactment” (Russell 2007). NewYAWL is a derivative of YAWL but significantly incorporates 118 of the 126 patterns that were identified. On completion, the language was evaluated against criteria suggested to provide good foundation for process-aware information systems. This criteria addressed the formality; suitability; conceptuality; enactability; and comprehensibility of the language (Russell 2007). No limitations were discussed regarding the solution language.

Two concerns surround BPM. Firstly, it has inherited traits from original workflow automation technologies which presume all processes to be mechanistic in nature. Original workflow techniques are inadequate at supporting human-driven processes, such as “problem solving and design” (Harrison-Broninski and Hayden 2004) due to the involved sequential flow structure. Secondly, BPM is being adopted in the world of Software Engineering because of that trait. “Rather than being an extension of workflow concepts, BPM is now seen as systems-to-systems technology... BPM is becoming an IT Technology solution as opposed to the business process solution it was meant to be” (Pyke 2006). It was proposed that business process constructs which can facilitate “the execution of a business process described in terms of these constructs in a deterministic way” (Russell 2007) be formalised (Gonzales 2009a). This could be questionable since much of human interactivity has a non-deterministic nature (Conradi et al. 1992). For all the complexity of workflow patterns, this doesn't appear to be addressed. Many modelling techniques are used in industry to help define business processes for quality or policy purposes, however, “using traditional workflow notation to capture human-driven processes simply provides business people with a false sense of reassurance” (Harrison-Broninski 2006a). Further to this, every notation or language will have independent patterns and variations on different patterns that do not match up to those defined. For example, the BPMN doesn't stand up to state-based patterns (such as milestone or data based routing patterns) since the notation has no notion of state, which can lead to workarounds, for example involving intermediate events to simulate states. It could be argued that the pattern framework may not be the best means for evaluating languages and notations since there is an underlying assumption that conformance to patterns equals a useful notation or language, and no consideration is made for usage context. It is suggested here that a notation or language conforming to all patterns might actually represent one that is overly complex and unfit for purpose and that one notation or language may suit a particular context better than those which might support more patterns.

2.2.3 Business Process Modelling

The aim of Business Process Modelling “in the phase of analysis is to understand processes in a domain” (Macek and Richta 2009). Business Process Modelling “aids the software developer, by helping to reduce the problems associated with the elicitation of systems requirements” (Phalp 1998). Much has been written about

the benefits of standardisation in the BPM world and the need for commonality in process specification. BPMS providers now have a “near-unanimous” (Silver 2008b) acceptance of the BPMN definition and support for the notation is given by the OMG (OMG 2005, 2008a). Version 1.x is now supported by most mainstream software technologies, with Version 2.0 carrying the support of main IT software producers (IBM, ORACLE, SAP, MICROSOFT), “any BPM tool... that does not support BPMN will be relegated to the “legacy” category” (Silver 2008b). The BPMN is entering a second incarnation with the 2.0 specification and Silver (2009b), a long time proponent of the BPMN, summarises the focus of the Version 2.0 specification, identifying advancements intended to resolve some of the difficulties represented in earlier specifications in Silver (2009b) and direction to features excluded from the Version 2.0 specification in Silver (2009a).

The BPEL was designed as a formal method, complementary to BPMN and provides the specification of business processes in serialised XML. The two techniques are so closely aligned that it is entirely possible to transform from one technique to the other, thereby enabling the bridging of “the gap between business process design and implementation” (Gao 2006). However, the “BPEL is a low level language that is necessary for IT to effectively build and expose business processes as services. The downside is that business doesn’t want the low level discussions and the IT to business communication becomes challenging because of the detail required” (Kavis 2008). The BPMN was designed to be user friendly in comparison with the BPEL and the 2.0 specification effectively replaces BPEL as an execution language (Silver 2009a). The “BPMN is on the way to universal adoption” (Harrison-Broninski 2006c) and with it, a prediction that the usage of the BPEL will diminish. This is because, as well as the BPEL mapping functionality; the BPMN is defined alongside its own Business Process Definition Metamodel (BPDM). This specification enables the XML to be generated directly from the BPMN and stored in the XMI format. Since the XML is complete, it can be used for target transformation into J2EE and .NET platforms, hence rendering the BPEL extinct. The prediction here is, that the BPEL will become surplus to requirement. SOA technologists are likely to recognise that the alignment of BPMN and SOA, taking advantage of the opportunity in adopting the BPMN within their products. Regarding business processes, SOA is currently suffering the same difficulty as the MDA in being IT oriented. To overcome this, the SOA “needs to take on board the general principles and patterns that underpin business activity” (Harrison-Broninski 2006c).

Techniques such as the BPEL, BPMN (and UML) are “biased towards providing IT support, rather than toward describing human behaviour” and “not really suitable for high-level description of business activity” (Harrison-Broninski 2005c). The difficulty becomes evident in that “users (both business and IT) think they understand BPMN” (Silver 2008b) and the truth is, the specification can be understood quite differently by those involved. The specification is different to general flowcharting in three main ways. Firstly, the definition is semantically rich, whereby each process is defined using nodes and connections that have specific meaning, and it is those semantics that are of interest to IT. The notation can be used to convey ideas

of surface flowcharting but semantics drawn from such sketching may result in misunderstandings, especially in terms of the MDA and the CIM. Secondly, the BPMN is event driven, allowing focus to be given to alternative and exceptional behaviours, which is something that is not necessarily considered when attention is given to flowcharting methods. Lastly, sub-processes can be defined and hierarchically structured, allowing the user to give focus to the level of abstraction in the process as required. This is not always reflective of real business processes which are humanistic in nature and not always neatly structured in such a sequential and hierarchical manner. Wholly humanistic processes have been subject to scrutiny in the suitability of BPMN to model such processes, they have also been “problematic” for description in the BPEL (Havey 2007). The RAD is an alternative technique to modelling human-driven processes more successfully. However, it has been suggested that the BPEL can support such tasks via a “larger orchestration” (Havey 2007). In Dwyer (2010), the question “Is the BPMN suitable for use by business people?” is posed to the business process modelling community. The majority of comments seemed to relate to a common theme identified by Dan Madison in that “BPMN is fine for automation and IT users” (cited Dwyer 2010). This is supported by a public debate on BPTrends (see www.bptrends.com) which concluded that “current BPM techniques and tools do not cater for collaborative human work processes” (Harrison-Broninski 2006c). Such human-driven processes are becoming the requirement for software system and web service support, to which concepts of HIM and the use of the RAD notation is recommended since the BPMN is inadequate in managing humanistic processes. Furthermore, the BPMN has “no concept” of requirements and does not support a requirements view of the business process (Perry 2006).

2.2.4 Role Activity Diagram (RAD)

In 1986, Martyn A. Ould and Clive Roberts presented the Process Modelling Language (PML), a formal language for business process execution (based on a previous incarnation known as the Requirements Modelling Language (RML) (Roberts 1988), described in Greenspan et al. (1994)), and a diagrammatic notation for informal process definitions known as the RAD. Central to these languages were the concepts of *role*, *activity*, *assertion* and *entity*, each with their own particular underlying properties (Roberts 1988). Also included was a transformation process allowing for RAD processes to be executed in PML. RADs are useful since they cater for the dynamics of real processes, allowing for them to be flexible and “loose fitting” (Ould 2003). RADs can be developed over time, allowing enough flexibility to adapt to business process needs and remain simple to use for the business user in modelling business processes (Dawkins 1998). According to Ould (2004c), the RAD facilitates modelling of “aspects of the real-world” (Ould 2004c) by providing natural concepts such as *roles* and *interactions*. The software world of the machine is formal and based on logic (Jackson 2000).

In the RAD, processes are divided across roles. The RAD notation is rather more simplistic than alternative software techniques for describing business activity, such as the BPMN, requiring little to no training to use.

The RAD notation is included as part of the RIVA method (formally known as the Systematic Technique for Role and Interaction Modelling (STRIM)). The key enhancement included with RIVA is that it facilitates the construction of what is defined as a *process architecture*, which is formed upon what are known as “essential business entities” (Ould 2003). The *Process Trinity* (as described in Ould (2004a, 2006)) is used to derive the essential business entities and process architecture. The process architecture facilitates process strategy and change management, which is a “vital requirement of BPM of supporting the agile business” (Ould 2003) and, with the added concept of *persistence*, views can be created of past, present and potential processes. RIVA provides an architectural theory to enable the business user to recognise where natural fault lines occur, i.e. to identify a complete process, extract it using an appropriate tool and thereby allow for it to be suitably modelled. This allows the models to retain the “coherence that exists in the real-world” (Ould 2004c). There are essentially three themes to address when considering real-world business processes. They are: *Collaboration*; *Concurrency*; and *Mobility* (Ould 2004b, 2004d, 2004e).

In an effort to create a “new approach to process modelling” (Abeyasinghe and Phalp 1997), Hoare’s Communicating Sequential Processes (CSP), which gives focus to input/output events and concurrent processes (Hoare 1978), and the RAD, were combined via a methodological mapping process involving mapping rules. These approaches were selected as representative of *best practice* examples from the formal and informal approaches available (Abeyasinghe and Phalp 1997). This work is extended in Phalp et al. (1998), where the RAD is used to form an executable specification of a business process by translating the RAD into RolEnact syntax; models can then be run on a computer using a Windows based interface, allowing for the business process to be “debugged before its implementation, and process specification errors captured far earlier” (Phalp et al. 1998). Specifically, the RAD facilitates understanding and accessibility for business users. It is “difficult to validate process scenarios with users” (Abeyasinghe and Phalp 1997) when using a formal notations alone (Johnson 1987; Johnson 1988). The combined approaches allow for this advantage, whilst remaining formal in application. A communication and decision making framework suggested in Beeson et al. (2002) supports this notion where “plans can be reviewed and modified in the light of changing circumstances” (Beeson et al. 2002). A case study involving AXA Sun Life, Bristol HQ was the focal point for a conceptual process model produced to reflect the communication and decision points. RADs were found to be “very useful for capturing the essential dynamics and information in the process” (Beeson et al. 2002) but presented some difficulty in describing role merging and over-constraint on sequential activities.

RIVA is a method that gives focus to processes, whereas other techniques (BPEL and BPMN) are machine focussed. Besides supporting agile processes, a key benefit with the combination of the techniques presented in RIVA in application to the MDA is that the concurrency of business processes could be adequately addressed. This is “how concurrent processes interact, and which processes can have many concurrent instances and how they interact” (Ould 2003). Moreover, the process architecture develops “over time and with use, rather than being set in concrete at the outset” (Ould 2003). The difficulty of applying the RAD to

the MDA is that describing software is not the original intention of the RAD. "Role Activity Theory has acquired a core of adherents over the 20 years since its invention, but never quite made it into the IT mainstream, which over the years has concentrated on building information-based, not process-based, systems" (Bushell 2005). Therefore, it may be important to draw on knowledge of software modelling if any CIM application of the RAD is to be made.

2.2.5 Human Interaction Management Systems (HIMS)

Human Interaction Management (HIM) is a technique for modelling human system behaviour via the use of the RAD notation (see www.human-interaction-management.info) (Bushell 2005). The combination of HIM and the *role* concept is central to this movement, since the "proper division of information is linked intimately to proper division of behaviour" (Fingar 2007) and thus, the behaviour is represented in terms of human collaboration. The role utilises private data to validate conditions in terminating role activity and instantiate other roles or activity. Notations such as the BPMN allow only for messages to interact between two single entities; with human-driven processes, interaction is usually between many (for example, a conference call), which is allowed in RAD. BPMN is a notation that "only vaguely captures the process" (Harrison-Broninski 2006a), RADs can adequately model human-driven interactions and they are understandable to non-technical business users, without the need for exhaustive training. An additional benefit is that the number of documents involved in business process modelling can be reduced; with the introduction the RAD a single-page (Harrison-Broninski 2006a).

This technique is both contrary, and complementary to techniques of Software Engineering such as the BPEL, BPMN and UML, which give focus to processes that are automata. The MDA describes the use of UML in the creation of the PIM. However, "people are not programs, and their behaviour cannot be properly described, controlled or supported using techniques such as BPEL, BPMN or the UML" (Harrison-Broninski 2005c) since such techniques are focussed upon the requirement of software implementation rather than providing support for defining the requirements of human-driven processes and are therefore unsuitable, which is the reason why (perhaps in reference to Smith and Fingar's book "Business Process Management: The Third Wave" (Smith and Fingar 2003)) HIM is described as the fourth wave of BPM, beyond technologies such as the BPMN, where "contracted processes" and "irregular collaborations" are accounted for (Korhonen 2008).

2.2.6 Goal Modelling

In the modern world of business, dynamic demands placed on organisations force them into "reactive patterns of change" (Kavakli 2004). "Relationships in modern thinking go far beyond inputs and outputs" (Owen 2009b). This requires that businesses are able to be agile enough to achieve such organisational change

through the precise tuning and balancing of organisational goals and strategy. The point of implementing information systems in the business arena is to achieve some form of business strategy. It is therefore important that such IT systems are correctly aligned with the high-level business strategy, where “humans are intrinsically goal-driven” (Conradi et al. 1992). Goal modelling is used in the field of Software Engineering to ensure that the system is built to specification, and that the specification is representative of the *right* system. It is suggested that “existing software development methodologies... have traditionally been inspired by programming concepts, not organisational ones, leading to a semantic gap between the software system and its operational environment” (Castro et al. 2002). Owen (2009b) discusses the definition of intangible elements into the system and the design of systems based on stakeholder needs, business goals and environmental changes (Owen 2009b).

In its most generic form, goal modelling is comprised of a tree or network diagram which begins with some high-level goal and branches into a series of sub goals, indicating the causal relationship between each goal (Kavakli 2004). Goals range hierarchically from high-level strategic goals to lower level technical and sub-goals. Many techniques are available in consideration of goal modelling and the choice of techniques is best suited to the individual context. Research has shown that the integration of methods can also be of benefit in the required context (Kavakli 2004). Three key trends in process modelling adversely affect small business ability and desire to engage in process modelling. They are that there is a plethora of notations available; a large degree of complexity associated with notations and process models; and a serious lack of real evidence to support the application of process modelling (Phalp and Shepperd 1994). To address this, Phalp and Shepperd (1994) present the Goal, Use, Investment, Deliverables, Experience/Environment (GUIDE) checklist to “tie the modelling notation used to the goal of the work” (Phalp and Shepperd 1994), using the DFD as a communication mechanism.

In Basson (2009a), five basic capabilities are identified as the “essence of Integrated Business Management” (Basson 2009a), suited to the changing business environment. They are *People*; *Guidance*; *Process*; *Information*; and *Resource*, where goals are held central to the guidance capability. Another drive to align goal modelling within software development is presented in Castro et al. (2002) where four phases are associated with outputs in the Tropos methodology and demonstrated in table 2.2.6.1.

Phase	Output
(1) Early Requirements - Organisational Model	Strategic Dependency Model Strategic Rationale Model
(2) Late Requirements - System To Be	Revised Strategic Dependency Model Revised Strategic Rationale Model
(3) Architectural Design - Global Architecture With Sub Systems	Non-Functional Requirements Diagram Revised Strategic Dependency Model Revised Strategic Rationale Model
(4) Detailed Design - Each Architectural Component Described In Detail (Map To UML)	Agent Class Diagrams Sequence Diagrams Collaboration Diagrams Plan Diagrams
Implementation - Representation And Generation Of Code Base	Beliefs-Desires-Intentions Agent Architecture for implementation in JACK

table 2.2.6.1, *four phases of requirements driven design* (Source: adapted from Castro et al. (2002)).

Rather than basing a development methodology in the concerns of Software Engineering, the Tropos methodology is proposed and presented in a case study format with RE ideals; the hope being to reduce the semantic gap and eliminate factors apparent in causing project failure (Castro et al. 2002). It is claimed that “requirements analysis is arguably the most important stage of information system development” (Castro et al. 2002) and time spent focussing on this stage of development is valuable, since failure to recognise a mistake in a requirement could lead to expensive design alterations later in the development process. A similar technique known as Goal-Oriented Requirements Engineering (GORE) is defined for “eliciting, elaborating, structuring, specifying, analysing, negotiating, documenting and modifying requirements” to help ensure that goals are maintained and satisfied during software development (Robinson 2007). The *i** (distributed intentionality) is another framework that gives attention to elements such as *actors*, *responsibilities*, *objectives*, *tasks* and *resources* rather than Software Engineering concepts like *objects*, *agents* etc (Castro et al. 2002; Dowson 1987a, 1987b). However, it appears that much of the soft-goal requirements are lost in translation when the process moves into UML mapping, since business *needs* and *wants* have no representation in the UML. The Business Strategy Context Process (B-SCP) framework is presented to address the alignment of business and IT in software development by ensuring the “validation and verification” of strategic alignment in terms of the information technology involved and the underlying requirements (Bleistein et al. 2006). This is achieved through the traceability of the framework in that system processes can be traced back to the initial strategy of which the company is setting out to achieve. This framework is an integration of three separate methods (supporting what was previously noted regarding how the integration of methods can sometimes be beneficial given the right context (Kavakli 2004)) which are *Organisational Goal*, *Context* and *Process Modelling* in order to achieve such alignment and traceability. Model integration is achieved through mapping rules with respect to elements of each particular model. *Goals* (Strategy) are directly linked to *Requirements* in the Jackson Context Diagrams (Context). *Domains of interest* (Context) are associated with *Roles* of the RAD (Process) and *Goals* (Strategy) are related to RAD

state transitions (Process). The central concept of connecting strategy to requirements is perhaps the most valuable part of the research with an interesting look at model mapping which is quite akin to the MDA.

Stakeholders are “seldom aware of how their role contributes to the realisation of business-wide objectives” (Kavakli 2004) and goal changes can be revealed in the process of asking *how* and *why* questions in abstracting high-level goals into sub-goals. It is said that “development methodologies have traditionally been inspired and driven by the programming paradigm of the day” (Castro et al. 2002). Goal Modelling techniques adapt if the requirement is made to do so, or in light of new information, which leads to the benefit of integration with the MDA in facilitating the agility of changing needs and the inspiration and drive from a business context. Therefore, a consequence of a goal-driven software development process is that goals must also be agile in respect of environmental changes. This is because software development is “non-monotonic” (Thomas 1989). Furthermore, at some point in the decomposition of goals into sub-goals, a primitive is reached, which itself can be “difficult to discern” (Thomas 1989).

2.2.7 Sketch Recognition

A solution to bridge the gap between requirements and design methodologies is provided by Scott W. Ambler in Ambler (2007), where it is suggested that Agile *Inclusive Techniques* be used to provide non technical stakeholders with sufficient tools that are simple to understand and can adequately transfer into a technical PIM (see Section 2.3.9 for an extended description) (Ambler 2007). With this in mind, some interesting research in the area of UML Sketch Recognition is provided in Hammond (2001) by Dr. Tracy Hammond at the Massachusetts Institute of Technology (MIT). Sketching is well practiced in brainstorming requirements and initial ideas for a solution. Usually, once design begins, such sketches are left behind in the development process (in much the same way as requirements models). Sketch recognition allows a designer to draw UML sketches onto a whiteboard or tablet, the same way as they might appear on paper. In the MDA, such requirements level etchings could now transpire to be artefacts useful to design through such a technique. Simulation models are usually used “when the complexity of the system being modelled is beyond what static models or other techniques can usefully represent” (Raffo et al. 1999). ASSIST (A Shrewd Sketch Interpretation and Simulation Tool) uses heuristics to interpret the sketches and enact them, allowing prototypes to be simulated. Objects and associations can be created, deleted and moved, all with a view to gaining a better understanding of the model. The real beauty of this technology is that it has been designed to interface with Rational Rose. This is to say that, once a drawing has been made in the *Natural Sketch Recognition* environment, it is then translated into XMI elements of, for example, a Class Diagram. From this output, it can be imported into Rational Rose which can then generate the required code associated with the diagram and/or any number of transformations. In theory, this would allow for a technically challenged user to participate in the generation of numerous variations of prototype models, all from a simple sketch. “Sketches are recognised based on what the drawn object looks like, rather than how it is drawn” (Hammond

2001). The concept can ultimately be extended, since it does not have to include the UML as a default standard, it could in fact be extended to any modelling format. See the MIT's ASSIST project at http://rationale.csail.mit.edu/project_assist.shtml for further information.

Professor Randall Davis from the MIT and his team have created a software system known as *SketchPad*, which is tantamount to DSM for mechanical engineering, extending the application of sketch recognition. With the *SketchPad* environment, mechanical behaviours can be modelled and simulated, thereby connecting initial requirements to real design without any constraint in complexity. This enables stakeholders to have close involvement in the brainstorming process. Currently, only a limited number of mechanical components are recognisable and there is some concern as to whether or not the system could be effective at distinguishing between components that are similar in design. One solution might be to establish a best guess based on context, allowing for user interaction to fix any errors, or limit the system to only containing distinctive components (Adler 2001). With a degree of vision, this type of domain specific sketch recognition could be applicable to other graphical techniques, such as Checkland's Rich Pictures (Checkland 1981, 2000; Checkland and Scholes 1990), which is a flexible sketching technique for use in the analysis and design of software systems (Horan 2000), whereby user-defined concepts illustrated in a Rich Picture could be associated with design elements via the *SketchPad* Environment.

2.3 Alternative Approaches

Whilst the MDA remains popular subject matter in literature, it is important to recognise and discuss other popular approaches to the development of software systems and the alignment of those systems with business processes and requirements. Similar to those techniques discussed in Section 2.2, there are numerous approaches available for investigation and therefore this chapter limits discussion to those which hold a particular and popular interest in the domain and those which crossover specifically with the MDA in the literature.

2.3.1 Value Chain

The Lean Enterprise Institute (www.lean.org) approaches the connection between business and IT in a different way. The focus here is on workflow techniques that derive from Toyota research and are based upon ideas presented in James Womack, Daniel Jones and Daniel Roos' book "The Machine That Changed The World: The Story Of Lean Production" – A MIT study of the practices employed at Toyota. Firstly, a high-level value chain is mapped (*Flow Kaizen*) and streamlined, before giving direct attention to the lower level processes (*Process Kaizen*). Organisations "need to begin by streamlining the entire value stream, and only after that, drill down into specific processes to eliminate waste" (Harmon 2006) to achieve real efficiency in

the value chain and associated processes (therein lays the foundation of software development). For this reason, focus is given to Flow Kaizen, rather than Process Kaizen. The argument is that process management is ineffective. Research has shown that process optimisation is firstly focussed on a single process, rather than the system as a whole and in turn, the overall system is compromised (Dowdle and Stevens 2009). Secondly, process optimisation is not viewed as long term (Dowdle and Stevens 2009). In either case, the difficulty appears that, although theoretical basis and tool support are available, the guidance on how and when to implement and manage process based strategies is not (Dowdle and Stevens 2009). The method offers such guidance with the intention to be completely managerial in nature and that “software tools should not be used, since they distract managers from focussing on the work of developing a high-level map” (Harmon 2006). This is an important distinction from other methods that promote the use of software tools within BPM and the MDA. David LaHote is the president of Lean Enterprise Institute’s Lean Education Value Stream, and he outlines the key concepts of the *Lean* conceptual model in Lahote (2008). The concepts fall broadly into two categories, the first being that *Value* is to be defined from the customers’ perspective, per product. The second is focussed upon the importance of having a streamlined value chain. All of this is done “in pursuit of perfection” and with an “understanding of the system that best supports” those concepts (Lahote 2008). The Toyota Production System is a real implementation of concepts introduced by Lean which of course is rooted in the car manufacturing sector. However, the conceptual Lean model can be applicable to any business and forms the foundation for the Lean Six Sigma (LSS) method. Three central areas to understanding are *People*, *Process* and *Purpose* and that “tools need to work together as a system” (Lahote 2008).

2.3.2 Balanced Scorecard

The Balanced Scorecard (BSC) is commonly used to manage business strategy, the BSC is implemented and executed via the utilisation of key performance indicators and the technique is “IT-supported and [forms] a conceptual basis for management information systems” (Niehaves and Stirna 2006). In the enterprise it is of the utmost importance for executives to align IT with their business strategy (Wegmann et al. 2005; Wegmann et al. 2007). However, not much is offered by way of means to ensure the validity of initial strategy and goals. Kokune et al. (2007) argue that “the validity of the strategy model itself is essential [but is] almost entirely ignored in requirements engineering research literature” (Kokune et al. 2007), not to mention software processes.

In Kokune et al. (2005, 2007), Fact Based Collaboration Modelling (FBCM) is proposed as a methodology for “defining and validating business requirements” (Kokune et al. 2007) to verify IT alignment with business strategy by giving focus to the strategic model, ensuring that resulting system functions are based on a valid strategy. FBCM consists of five generic steps which are roughly based on BSC techniques. However, FBCM extends the BSC method to incorporate the availability of field and statistical information (Kokune et al.

2007). The five steps of FBCM consist of several sub-steps to be completed. Each step has many tasks associated with it. For example, Step 1 has five associated sub-tasks; Step 2 has two outcomes, one with seven elements and the other involving the analysis of collected information in a matrix – which is quite complex, relying on the availability of information to evaluate the validity of strategy structure (the causal relationships) using a correlation coefficient. Therefore, whilst raising many interesting insights in how information can be used to ensure the alignment of business with IT, the unavailability of such information may hinder application. Indeed, the availability of appropriate business knowledge to enable the implementation of BSC methods has been seen as a difficulty in other research, such as Niehaves and Stirna (2006). This difficulty is proposed to be overcome via the integration of the BSC with Enterprise Modelling via the Enterprise Knowledge Development approach (Niehaves and Stirna 2006). The suggestion being that Enterprise Modelling “can support BSC implementation projects that comprise activities requiring the discovery and documentation of organisational knowledge that is not easily accessible or not of sufficient quality” (Niehaves and Stirna 2006). For IT “system development, the aim of FBCM is just to clarify goals, objectives and the ways to measure the degree of their achievement” (Kokune et al. 2007) and, therefore, “it is necessary to utilise it in combination with a business process modelling method” (Kokune et al. 2007), which means it could be offered as complementary to any software process (or indeed business process) in terms of the definition and validation of the strategic model.

2.3.3 Enterprise Modelling

As previously noted, the alignment of business strategy with IT is important for the success of the enterprise and it is therefore recognised that such alignment must be considered on an enterprise level (Wegmann et al. 2005; Wegmann et al. 2007). However, the task of integrating IT into the business organisation is complex. The organisation of IT within the company is affected by current business processes and the strategies behind those processes. Current business processes are also affected by the organisation as new ways of completing tasks are discovered through the use of IT. Such complexity causes “language barriers” (Frank 2002) between business users and Software Engineers. Enterprise Modelling is seen as a way of alleviating involved concerns and degrading such barriers.

The Multi-perspective Enterprise Modelling (MEMO) framework is an example of a conceptual framework for Enterprise Modelling. The focus of the MEMO framework is on three distinct modelling perspectives, across five alternative perspectives (see figure 2.3.2.1). By addressing these complementary views, complete and precise software systems might be created, grounded in intuitive business process design and information systems knowledge, both being aligned with organisational strategy.

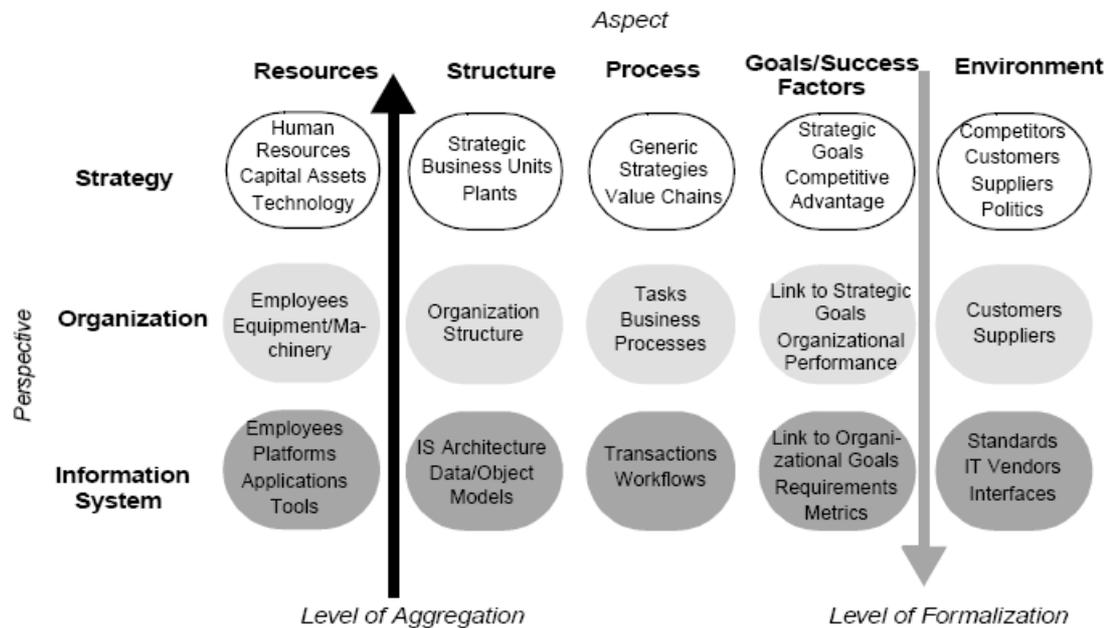


figure 2.3.2.1, perspectives and aspects related to MEMO framework (Source: Frank (2002)).

The two main goals of the MEMO framework are to enable Enterprise Modelling with strategic alignment and to create an organisational wide knowledge base consisting of core models which can be integrated and reused.

The Systemic Enterprise Architecture Method (SEAM) is offered as a means to achieve more of a complete alignment of all environmental concerns in comparison with traditional mechanisms, such as the value chain, the business process model and the UML (Wegmann et al. 2005). Business and IT alignment is defined by SEAM specifically as “system alignment between organisational levels (from business down to IT) and system alignment between functions levels (within the same organisational levels)” (Wegmann et al. 2005). The approach is based on systems thinking (see Checkland (1981, 2000), Checkland and Scholes (1990)) where systems might be viewed as a whole or as a composite (depending on audience) and the emphasis is placed on a “well-built enterprise model and not on the process of building it” (Wegmann et al. 2007). The enterprise model is created and, from that, particular views are generated and focussed upon by the designer. The enterprise model describes the business goals, strategies and needs of the stakeholder in terms of “the markets of an enterprise, the enterprise itself and its IT systems” (Wegmann et al. 2005) in the context of how things are at project start-up and how things are proposed to be at project end. However, the author concedes that “to be truly practical, SEAM needs to have tool support” (Wegmann et al. 2005).

Enterprise BPM is “the modelling of all processes in the enterprise, as part of and in context of the total enterprise business model” (Musschoot 2009). That is, “the process of modelling all relevant aspects of the

business” (Musschoot 2009). By examining the “building blocks” of Enterprise BPM, it is possible to “build a solid base for modelling meaningful and correct business processes in an enterprise context”(Musschoot 2009). Silver (2008a) discusses 3 levels of process modelling. They are *Descriptive*, *Analytical* and *Executable*. A base diagram is created at the *Descriptive* level, to be informative about a process. The more detail that is added, the higher the model will progress until it becomes fully *Executable*. Therefore, “the amount of detail depends on the level you want to achieve” (Musschoot 2009).

If you are modelling business processes to assure the mutual comprehension between business and IT, it is mandatory to involve both stakeholders in the modelling process. If not, you'll put yourself in a very difficult position (Musschoot 2009).

Therefore, the languages used for modelling software systems should provide business users with the capability to model elements in a language that they understand; technical interpretation should be provided in a similar manner since “a modelling language is an instrument, not an end in itself” (Frank 2002). With Enterprise Modelling, models are created specifically for that domain whereas with the MDA, models are typically in the UML, which is “designed for the development of software systems” (Frank 2002), and does “not provide concepts and graphical representations that are appropriate for all aspects of an enterprise model” (Frank 2002). Enterprise systems are inherently difficult to model due them being large and complex. However, the “MDA envisages systems being comprised of many small, manageable models rather than one gigantic monolithic model” (Appukuttan et al. 2003a), which supports the idea for model abstraction in such cases.

2.3.4 Software Process Modelling

Traditionally, software process modelling focused on one complete model of the software process. As languages evolved and became more complex, so too have software process models. Models of the software life-cycle first came about during the 1950s/60s. Primarily, they were to “provide a conceptual scheme for rationally managing the development of software systems” (Scacchi 2002). Examples include the Classic Software Life-cycle (or *waterfall* approach); Stepwise Refinement; Incremental Development and Release (*prototyping*); and Industrial and Military Standards, and Capability Models, for example (ISO12207) (Scacchi 2002). “A software life-cycle model is either a descriptive or prescriptive characterisation of how software is or should be developed... software process models often represent a networked sequence of activities, objects, transformations, and events that embody strategies for accomplishing software evolution” (Scacchi 2002). A difficulty in software development is the “representation and integration” of project information in a manner which projects can be controlled efficiently (Huseth and Vines 1987). Standards relating to process modelling are typically paper based, therefore meta-processes are suggested to structure

processes and enable the automatic verification of implementations and other standards against them (Ledru et al. 2006; Purper 2000; Turgeon and Madhavji 2000a). “Software development methodologies are intended to improve software development by specifying the products to be created, describing the activities to be performed, and guiding the execution of these activities and the use of the products” (Sutton et al. 1991). However, Starke (1994) extends the difficulty of modelling the software process by identifying five key areas which require consideration and relate to the terminologies used, number of available languages and paradigms, types and instances of process models, dynamic changes which occur, and standardisation of approaches (Starke 1994). Zave (1989) proposes that “some application domains for which software is written are well understood, and some are not. This distinction is crucial to understanding – and improving – the software process” (Zave 1989). A close relationship is suggested between the domains software process and information systems modelling, in that both address a similar problem but use different approaches, and one could learn from the other to alleviate such difficulties (Conradi et al. 1994).

It is claimed that “by formalising the methods that are... used to develop software, we will be able to correct deficiencies, and incrementally enhance the way software is constructed” (Huseth and Vines 1987). “The process of software design... is one of the most creative of human activities” (Katayama 1988) and therefore, to formalise such an activity may seem counter intuitive. “Difficulties in (formally) specifying... software processes... are due to the fact we have not... clearly identified what we can (because it is purely mechanical) and what we cannot automate (because it requires creativity, intuition)” (Rombach 1988). Focussing on *characterising, planning, executing, learning and feedback*, a formalised Specification Framework is suggested to account for automated support in the Software Engineering environment (Rombach 1988). A formal approach to specification is extended to account specifically for process modelling and is presented as the Organisational Base Model in Sa and Warboys (1994). It utilises a formal stepwise refinement technique which is extremely complex and plagued with unfamiliar terminology and temporal operators, understanding of which is required to understand and apply the method. The logic also requires further clarification as it appears to have an altogether sequential nature. Therefore, it is difficult to see how alternative and exceptional cases could be modelled. It is considered important that any formalisation should be human understandable (specifically by managers); able to develop with the changing environment; hierarchical in design; able to accommodate process concurrency; and design alternatives (Katayama 1988). Justification for such a formalisation can be given in that “every scientific study begins with description; software methods... need to be described in some language so that they can be better used and communicated; and the software industry needs some means of process description to achieve better quality control over products” (Katayama 1988). However, it is argued that complete formalisation is not required so long as there are enough *manual-overrides* available for human users to account for any alternate or exceptional behaviours within a process (Jackson 2000).

Integrated Project Support Environments (IPSEs) can be used to support the software development process. Focus in Ashok et al. (1988) is given to the architecture of such an environment and it is argued that they should “maintain an explicit model (representation) of the software process that is to be followed in a project” (Ashok et al. 1988). The architecture is hierarchically arranged, “communicating sequential tasks” (Ashok et al. 1988) in the form of activities. Researchers at the Software Engineering Institute examined ways to augment software processes with management support (Kellner 1991). View-based models focus on eliciting information about the software process from multiple sources, thereby constructing a model of the software process from numerous alternate views. By giving focus to a single view, each view can be subject to an individual verification procedure. The application here looks to provide management planning and control mechanisms within software process modelling. The approach uses the STATEMATE tool to “represent, analyses, and simulate software processes” (Kellner 1989, 1991) and is characterised by the inclusion of three alternate modelling viewpoints. They are firstly, *function* (in the form of an Activity Chart displaying what happens in the process); Secondly, *behavioural* (in the form of a State Chart displaying when and how things happen); and lastly, *organisational* (in the form of a Module Chart displaying where things happen and who is involved in such happening) (Kellner 1989, 1991). In Nuseibeh et al. (1993) two views of granularity are suggested to enhance the model of the software process. This method supports the decomposition of the *global* view of the software process into smaller processes in terms of individual developers (Nuseibeh et al. 1993). The “big problem” faced with traditional project life-cycles is that they are both costly and time consuming, which leads for the need for process definitions to be agile (Hogg 2009). One way to improve the software development process is suggested in Turgeon and Madhavji (2000b) where a similar view-based modelling approach to software process modelling is considered. It is suggested that “models of software processes elicited using a view-based approach are generally of higher quality (specifically, more complete) than those elicited using traditional, non-view based, modelling approaches” (Turgeon and Madhavji 2000b).

Users require that “business systems... be... available instantly and operate flawlessly” (UC4 2008). The concept of *Case Management* was first introduced in the 1990s. It is suggested that the traditional structured environment is fixed and considered at design time, whereas an unstructured and ad hoc process flow can be considered at execution time. Therefore, a successful BPMS must provide both *Design-Time* and *Run-Time* Case Management (Hogg 2009). To facilitate such agility, it is suggested that the process be able to be adapted by the user role in execution time and not be bound by design-time decisions (Hogg 2009), giving flexibility to user roles. The scale of distributed and enterprise computing adds to the complications involved in creating such systems and, therefore, the objective is to change processes “to respond dynamically to changing business requirements and competitive pressures” (UC4 2008). Tools must venture “into a new realm where applications and events can be driven dynamically in response to constantly changing business conditions” (UC4 2008).

Contemporary models of software development must account for software, the interrelationships between software products and production processes, as well as for the roles played by tools, people and their workplaces (Scacchi 2002).

Software process technologies were originally intended for computer expert use and as such, they describe the software process in a sequential nature and do not account for *concurrency* (Taylor 1987). Traditional software processes are simple and stepwise, but lack the flexibility to support the natural environment and “capture and formalise a wide variety of data types, types of users, classes of tools, and the complex relationships between the steps of a process that will be used in a large software development project” (Huseth and Vines 1987). “In order to improve software development capability, one should improve software development processes” (Turgeon and Madhavji 2000b) and provide “automated support” (Rombach 1988). “It could be argued that process descriptions should not be procedural at all, and that functional descriptions have much greater potential for promoting concurrency” (Taylor 1987). An alternative consideration when modelling the software process is to take business goals as the foundation for that process, rather than activities or products (Taylor 1987; Thomas 1989). Difficulties are compounded when the process involves a team effort as there is a need to ensure that individual members are all singing of the same specification (Taylor 1987). Therefore, methods which embrace business process research in modelling concurrent activities and business goals in the software process could be preferred over procedural ones. “The use of behavioural description makes it possible to describe the software process at any desired level of abstraction and, therefore, assists in accommodating aspects of the process which are poorly understood” (Williams 1988). Models “must go beyond representation. They must support comprehensive analysis of a process. In addition, models should allow predictions regarding the consequences of potential changes and improvements” (Kellner 1989); which leads to the recommendation of enactable models where simulation can be used to “draw a clear and touchable picture of the future system for the managers and users” (Mahmudi and Tavakkoli 2005), supported by Grützner et al. (2004).

2.3.5 Process Programming

An approach to formalised process modelling is known as *Process Programming* (Osterweil 1987). Software process programming is an approach to specifying the software process, in which the software process is formalised. Conventionally, the interpretation and implementation of such a development ethos has been a manual process. This can lead to a degraded understanding and interpretation; with both incorrect and inadequate implementations resulting. Models of process programming “can be machine interpreted and can, therefore, be used as a process control mechanism” (Lehman 1988).

Processes are complex to program and each programming paradigm is associated with individual difficulties. For example, declarative languages have structural issues as they do not have well defined import/export interfaces; object oriented approaches have dependency issues; and *net-like* approaches are complex and lack management capabilities (Deiters et al. 1989). Indeed, process programming is comparable with programming “on a very high or abstract level” (Deiters et al. 1989). To address these issues and apply process programming to the software process, a multi-paradigm approach is suggested which encompasses declarative, object oriented and conventional approaches (Deiters et al. 1989). A method is defined to enable the development of “complex process programs” that uses a structured framework of mechanisms known as the Model for Software Processes which has “well-defined import/export interfaces” (Deiters et al. 1989).

In Sutton et al. (1991), REBUS was developed as “a prototype process program for the specification of software requirements” (Sutton et al. 1991) to address issues and demonstrate the feasibility in part of software process programming and includes the definition of a data model (in the form of a Directed Acyclic Graph) and a process model (in the form of pseudo code and state charts). The investigation found that “REBUS demonstrates to a significant extent that process programming is feasible” (Sutton et al. 1991). The method facilitated user interaction, automated a great deal of the more *mundane* features, and formally included a requirements model - although not in a formal language (Sutton et al. 1991). However, the given focus on a single phase (i.e. requirements-specification) made it difficult to avoid promoting the software process behind REBUS, rather than the concept of software process programming. Moreover, a requirements-specification is inherently not something expected to be subjected to programming and automation. The discovery here is about how programming can be used to situate the software process within a development environment; the content of that process is not necessarily important. The investigation may have been better applied at a phase further downstream from requirements since, as the authors quite rightly elude to requirements inherently containing a great deal of human interactivity, some of which may not be candidate for programming in a software environment. “Formalisation and mechanisation do not enhance human intelligence. Formalism can provide support for human understanding; mechanisation will reduce or replace such human repetitive activity for which the need and make-up is predictable. And so on” (Lehman 1988). Therefore, a concern is raised that by increasing the formality of the process specification; an increase in complexity in both definition and understanding of the process might be experienced. This may prove to be counterproductive in terms of relaying information between stakeholders.

2.3.6 Functional Modelling

As previously noted in Section 2.2.1, the rise of information systems inspired modelling techniques in terms of functionality. According to Owen (2007), product development is faced with two problem types: those of depth (“failing to spend time and resources establishing what to make or implement... before committing to

planning how to make it" (Owen 2007)) and those of breadth ("failing to consider the full range of users, and its remedy: establishing who the users are and the aspects of system functionality they are concerned with" (Owen 2007)) and it is the latter to which functional modelling is concerned with.

Structured analysis techniques can be used to model the functionality of a system and are considered better at communicating ideas, being easy to understand and use, maintaining clear system boundaries and accounting for abstraction & partitions and tool automation via Computer Aided Software Engineering (CASE) tools (Easterbrook 2003). Structured Analysis and Design Technique (SADT) is a top-down "diagrammatic notation for constructing a sketch for an application" (Mylopoulos 2004a). In the notation, boxes are used to represent data or activities and arrows represent relationships between boxes in the form of inputs, outputs and controls. With up to six boxes in each diagram, each box can be broken down into further diagrams, "leading to hierarchical models of activities and data" (Mylopoulos 2004a) in terms of functionality. However, SADT neglects project projection and timing issues and confuses the modelling of the problem with the modelling of the solution (Easterbrook 2003).

The IDEF (ICAM DEFinition) family of languages began development in the 1970s U.S. Air Force Integrated Computer Aided Manufacturing (ICAM) program. IDEF has also become known as "Integration DEFinition" due to its later focus on the integration of modelling methods with other (IDEF and non-IDEF) methods and tools (Menzel and Mayer 1998; Russell 2007). IDEF0 is a commercial SADT based CASE tool for describing processes in the form of functions. *Inputs* are transformed via *Controls* into *Outputs*, subject to resource *Mechanisms*. These concepts are known collectively in IDEF as ICOMs (Menzel and Mayer 1998). The IDEF construct forces the consideration of each function in terms of each ICOM. This is positive in that models are likely to have a greater accuracy in respect of ICOMs, although the method does produce diagrams in a sequential fashion. Functions can be broken down to a detailed diagram and abstracted upon to the context level 0 diagram. IDEF3 is a "general purpose modelling method" (Menzel and Mayer 1998) where the focus is on process, rather than function. An IDEF3 process is also known as a Unit of Behaviour and is characterised "in terms of the objects it may contain, the interval of time over which it occurs, and the temporal relations it may bear to other processes" (Menzel and Mayer 1998). However, these types of solutions are "essentially focussed on business processes [and not] the ultimate realisation of the systems that they describe" (Russell 2007).

In consideration of modelling behaviour, a procedural modelling technique known as the DFD is proposed to model the functionality of complete systems (Stevens et al. 1974). Information flow is modelled, typically via notations derived from the influential works of Demarco (1979), Gane and Sarson (1977), Yourdon (1989), where square boxes represent internal and external entities, arrows show the flow of information, rounded boxes represent processing to carried out on information with open ended rectangles representing information stores (Mylopoulos 2004a). There are three levels of DFD; the objective of each level is to remove abstraction

from the previous level by adding further detail to it. A low level (Level 0) context diagram is first created showing the basic data flows between objects of the system and represents the *System Description* where processes and data flows are first identified (Cachia 2005). The practice of de-abstraction continues through DFD levels 2 and 3, until all important aspects of the system have been identified and visualised in the notation. DFDs are considered semi-formal due to the limitations that, whilst having formal syntax in the form of notation, DFDs lack formal semantics (and therefore executable DFDs are not possible) and control semantics (such as choice, concurrency and synchronisation) remain undefined in the notation (Gupta 2007c). These may be addressed by complementing DFDs with other notations, expanding the DFD notational set or by initiating a full DFD revision from semi-formal to formal, to account for such limitations (Gupta 2007c). Confirming all requirements are included within the *System Description* is also difficult; those that are not may become overlooked and never materialise as part of the system design. Therefore, a detailed definition is paramount in such an approach. Moreover, the DFD appears to represent only sequential processes; it is not clear how processes which have greater dynamics, such as those that are humanistic might be accounted for or how the DFD might be adaptable to new software development approaches, specifically model driven approaches such as the MDA.

It is argued that “good system coverage will ensure that all system users have their interests considered... Establishing the functions to be performed establishes the criteria to be met” (Owen 2007), and therefore the requirements. The Entity Relationship Diagram (ERD), which can be used to complement the DFD to describe “conceptual data models” (Gupta 2007a), supported by Saeki et al. (1991), is a simple and *easy to use* technique for modelling information (Easterbrook 2003). Formally proposed in Chen (1976), it is considered to be “easy to understand not only for systems analysts and database designers but also for managers and users” (Chen 1983). It provides a static view of the system (being the origin of the UML Class Diagram (Gupta 2007a)) and can translate “readily to relational schema for database design” (Easterbrook 2003). The main notational elements of an ERD are *entities*, which are classes of “autonomous” objects; and *relationships*, which exist between entities (Easterbrook 2003). Relationships can be of the type AND/XOR and an entity can also be related to itself. Cardinality is used to denote the minimum and maximum related objects. Like the DFD, the ERD lacks precise semantics and is therefore semi-formal technique (Gupta 2007a). There is also evidence to suggest that the ERD may be directly translated from natural English requirements by close examination of the sentence structure of provided natural English (Chen 1983). Easterbrook (2003) argues that object oriented techniques are considered to be more flexible because object orientation “emphasises the importance of well-defined interfaces between objects compared to ambiguities of dataflow relationships” (Easterbrook 2003). This is because the functions tend to change, but objects stay the same. Conversely, it is highlighted that “nearly anything can be an object” (Easterbrook 2003) and thought that this could be a problem leading to ambiguities that transcend those related to dataflow techniques. Some advantages of using object orientated techniques in RE are that they fit well with object oriented design; emphasis on functions is removed and they are more coherent than the techniques of

structured analysis (Easterbrook 2003). This does not come without disadvantage. By focussing on static models, emphasis is not given to dynamic modelling; Such elements of objects and associations may not be appropriate for modelling of those described in the real-world and object oriented techniques can tempt the user into focussing on design rather than analysis (Easterbrook 2003).

2.3.7 Business Rules

Business rules can be used to define the operation of a business process or the requirements for a software system with the objective of rule based languages being to match data to those rules. In Musschoot (2010), it is argued that functional models and business rules are common in IT, and therefore it should not be difficult to adopt the business process in IT via the definition of such models and rules. The greatest challenge posed by MDA appears to be the conflict between what is required in nature by software engineers, that is a high-level abstraction, in comparison with the business user's requirement of a low lever abstraction, that is "semantically rich enough to specify all the necessary business rules (including pre and post conditions)" (Berrisford 2004).

MARVEL is a model driven knowledge-based programming environment for the software process (Kaiser 1988). In this environment, business knowledge is retained in the form of *strategies* which are used to support technical aspects of the software process (Kaiser 1988). Strategies are rule based with each rule being associated with a pre-condition; an activity; and post-conditions (which address both successful and unsuccessful, or exceptional, process completion) (Kaiser 1988). *Opportunistic processing* is used to control the automation with rules via forward and backward chaining (Kaiser 1988). This highlights the overall sequential nature of the environment and presumes that all conditions can be realised prior to design; in effect excluding exceptional circumstances at run time. For such an environment to be useful in terms of agility it ought to account for both design and run time case management (Hogg 2009).

A trend taking hold in the business world is that companies are realising the benefit and reality of implementing specialist business knowledge in the form of business rules, especially in the composition of web services for use with the SOA, with the OCL being the language used by web service developers for writing business rules (Frankel 2006). In the MDA, business rules may be required to be applied across a multitude of applications, which in turn requires that the meaning of information and the method of exchange must be clearly defined, including behavioural constraints and restrictions. Beyond this, imposed rules must be continually and consistently validated by the domain experts involved. This highlights the need for clarity in defining business rules. In consideration of CIM transformations, Casallas et al. (2005) suggest a three-fold transformation. Business rules are first specified in natural language, and then manually transcribed into a UML Activity Diagram and finally, the Activity Diagram is annotated with elements pertaining to a

specialised UML Profile (which forms an integration infrastructure, defining semantics, responsibilities and restrictions of the associated elements). From this, sufficient information is apparently provided to execute transformations from CIM-to-PIM. However, by including such references to the defined UML Profile, one might rather consider this model as a rudimentary PIM rather than a CIM (due to the use of computational elements). An automated transformation method is suggested in Subramaniam et al. (2004), using a natural language parser developed by the University of Pennsylvania known as the Natural Language Toolkit. The natural language parser is at the heart of the architecture, which aims to generate Use Case diagrams and specifications. The key element is the business rules that the parser uses to generate the Use Case diagrams and specifications. "Business rules specify the guidelines to identify actors and use cases" (Subramaniam et al. 2004). The "tool is helpful to novice software designers for more efficient software design and thereby increasing software design productivity" (Subramaniam et al. 2004), and such could be a useful application in the MDA to help bridge between business user and software producer. With the Reference Model of Open Distributed Processing (RM-ODP), the CIM is representative of the *Enterprise* viewpoint; this includes defining business rules, facts and terms (Wood 2005). The *Information* and *Enterprise* viewpoints of the RM-ODP can be integrated with the MDA using the UML (OMG 2003b), supported by Wood (2002). "Business rules, facts, and terms in [the] CIM have corresponding elements in [the] PIM and PSM, obtained through transformations... Mapping the correspondences provides traceability of business rules between origin and implementation" (Hendryx et al. 2002). The PIM represents the *Information* and *Computation* viewpoints of the RM-ODP. It includes a static view (class diagram) and dynamic view (state chart), that "may correspond to other technology objects in PSM" (Hendryx et al. 2002). Business Rules in ODP are commonly defined in natural English policies; formal specification languages, the Language of Temporal Ordering Specification (LOTOS) and Z, can accommodate ODP viewpoints to varying degrees having "particular advantages and disadvantages in formalising the architecture of ODP" (Sinnott and Turner 1994). Business rules can also be used as the basis of model verification. An automatic approach to the analysis of UML Class Diagrams is presented in Massoni et al. (2004) where a formal object orientated modelling language known as Alloy is used. Business rules relating to UML Class Diagrams are expressed in OCL and mapped to Alloy rules, focusing on the validation of those OCL expressions (Massoni et al. 2004). However, the technique presented is quite complex. Furthermore, there is no indication that the OCL and Alloy are completely compatible. For business systems, this avenue of validation may not warrant the expenditure. However, for critical systems, where error reduction is paramount, or in a fully automated MDA environment, where the model is representative of code, this type of validation may be useful.

The difficulty with formal description techniques such as LOTOS or Z, and even the OCL, is that they have a complex nature.

An informal language may cause difficulty by sheer imprecision of both syntax and semantics. A formal language may cause difficulty in spite of well-defined syntax and semantics: some things cannot be said directly in the language and must be obscurely encoded; and a formal semantics may be too abstract to capture meaning effectively (Jackson and Zave 1993).

Should such languages be used to formalise requirements in specification, it may be complicated to involve business stakeholders, which is an important consideration (Kanyaru and Phalp 2005). Business process models “often include a ‘richness of description’, which is lost in moving to a standard specification” (Kanyaru and Phalp 2005); which can be due to such “formal syntax and semantics” (Kanyaru and Phalp 2005). Moreover, such formalisations “are more appropriate for specifying what a software component needs to do during design, rather than model the world” (Mylopoulos 2004b).

2.3.8 Software Product Lines

The Software Product Line (SPL) facilitated the rise in model usage, understanding and implementation. Earlier techniques in software development endeavoured to “organise software processes in terms of activities” (Nakagawa and Futatsugi 1989) but the real output is considered to be a product in terms of code, test cases or supporting documentation, leading to the idea of organising the software process in terms of such output product, rather than individual activity. SPLs look to industrialise the software process by focussing on the creation of unique encapsulated software artefacts that, when drawn together, define the complete software factory. An altogether more structured approach is given to realising consumer requirements. A software factory schema is *like a recipe of models, patterns, templates, frameworks, components, processes, test cases and tools* (Cook 2004b). A SPL is composed of a set of *features*. A feature is an aspect or function which is derived from one or numerous customer requirements. Feature dependencies match the dependencies of requirements, all of which enables customer requirements to be mapped directly into the product line architecture. Mapping rules are implemented to transform customer requirements into features, and features into assets (Zhu et al. 2006). A drive to feature-oriented domain analysis is presented in Thiemann (2009) where individual feature types are defined in the UML via the definition of UML profiles and related to the MOF metamodel (Thiemann 2009). By defining systems in this manner, development knowledge and tools are brought together in one place, reducing “cost and risk by distributing the software life cycle” (Cook 2004b).

A similar consideration can be made when thinking about the MDA, that is, the influence of requirements on the MDA, and how requirements and dependencies can be transformed into MDA artefacts. An example of how the SPL could be integrated with the MDA is the Requirement Specification Model for Product Lines (RSPL). The RSPL is defined by Kabanda and Adigun (2006) to facilitate the automatic generation of

requirements and allow for the separation of “interface logic from the business logic” (Kabanda and Adigun 2006). In this model, the CIM is represented as a *Domain Knowledge* repository. Tailored requirements by a systems specialist are representation of the PIM in the *User Perspective*, which are in turn transformed into the output *Requirements PSM* using a standardised template, with the output artefact being the Requirements Specification Document. SPL and the MDA combined form Model Driven Product Lines and “large scale reuse by means of structured and configurable representations of platform independent software assets” (Oliveira et al. 2004) can be achieved. A mapping process is used to combine artefacts in order to create platform independent features. Stakeholders identify design constraints which, along with the annotated class model, can be used to generate and execute instantiation script within the execution environment (Oliveira et al. 2004). This is a semi-automatic approach as the system developer will still be required to make some form of input in terms of class names, attributes, types and methods. The real benefit to this is that it ensures the interactive collaboration between non-technical stakeholders and technical developers in generating requirements.

2.3.9 Agile Methods

Currently in software development, there are those that believe systems should be delivered to “directly satisfy business requirements... [and be] less heavy-weight... with the vision of moving towards more agile modelling tools where users can quickly respond to change” (Koehler et al. 2007). With the advent of MDA, followers of Agile methods have embraced executable models as representation of code and not artefacts to be discarded during the development process. It is believed that the bridge between technologists and consumers could be made via a “highly abstract modelling language” (Mellor 2004). The MDA is conceptually very simple to understand, but very complex to implement. To make MDA work, there is a requirement to go beyond MDA modelling tools, such as Bridgepoint, TogetherCC and OptimalJ (Ambler 2007). In his article “A Roadmap for Agile MDA”, Scott W. Ambler presents MDA from the Agile viewpoint, terming this Agile Model Driven Development (AMDD). It is highlighted that not enough emphasis is placed on CIM development under the MDA (Ambler 2007); instead it is suggested to be better to be replaced by Agile’s *Inclusive Techniques*. Whether following strict guidelines of the MDA on development or an Agile process, it is proposed that these inclusive modelling methods (see <http://www.agilemodeling.com/essays/inclusiveModels.htm>) are sufficient for bridging from technological developers to stakeholders; there is however no automatic transformation from inclusive models to the CIM or the PIM. It is suggested that the UML is not a sufficient enough language for the MDA and that “the MDD community should focus on what’s practical and not on ivory tower theories” (Uhl and Ambler 2003). Albeit independent of platform and as previously noted, modelling tools go beyond the UML and therefore, developers become locked into such tools since outputs are unlikely to be interoperable with other tools. There is no PIM level universal Action Semantic Language that would allow for toolset integration. XMI integration is ineffective in practice and therefore an extension to the UML is required to take it beyond being

an “object and component technology” (Ambler 2007). AMDD methods have a good chance of success in the domain of MDA since they provide greater flexibility and a more lightweight approach to the MDA. *Inclusive Techniques* are very much akin to those of RE and could be employed to investigate the PD, allowing for stakeholder involvement in the development process. Simple design tools such as Together ControlCenter or Poseidon for object and Erwin for data modelling could provide developers with sufficient direction and support (Uhl and Ambler 2003). However, tools to directly integrate user requirements with AMDD are unavailable.

2.3.10 Service Oriented Architecture (SOA)

The birth of the SOA and the MDA has partially been a result of the narrowing of abstraction layers between which software systems have needed to interact over time (Frankel 2006). Currently, business users are accustomed to using generic business logic that is embedded into common off the shelf business systems. The SOA provides a *plug-and-play* platform, allowing for the production and reusability of standalone components (Skalle 2009) based on specialist business knowledge.

In Basson (2009b), the SOA is exemplified as a process-oriented methodology whereby services are governed by business processes, and therefore describes the integration of the SOA with BPM. “Business processes exist at two levels - the predictable (the systems) and the un-predictable (the people)” (Basson 2009b). This unpredictable element is increasingly difficult to accommodate in the Software Engineering environment. This is because “current technologies do not allow for the recognition and recording of these un-predictable activities” (Basson 2009b). This is to say that in a given situation whereby a human might well intervene and resolve a difficult situation regarding a business process, current technology is unable to make such an intervention and therefore some integration is the requirement. The argument is that “processes do not manage people - people manage processes” (Basson 2009b) and therefore attention is given to process-oriented tools and techniques. It is interesting to note that where the SOA is considered process-oriented, other development approaches (such as the MDA) are not. Although clearly most (if not all) development approaches could account for the business process, they perhaps do not offer enough attention to them and this may well highlight why approaches like the MDA fail to interface adequately with the business domain. Of the process-oriented approaches, the author writes:

These methodologies need further research into their possible application, but I am convinced that these (and possibly others) will shape our thinking about designing and engineering future systems (Basson 2009b).

Hans Skalle of Global Business Integration, IBM Software Group, discusses the combination of LSS with SOA and BPM to deliver “real business results” (Skalle 2009). Based on the presentation of business cases to

support investment in IBM, Global Business Integration is concerned with adopting a SOA, BPM and LSS development methodology to align business with IT, whilst remaining agile to change (Skalle 2009). With the SOA platform, BPM is made easier and business processes become reusable since they are service-based; enabling for process “improvement and design” simplicity (Skalle 2009). LSS can be employed to support SOA and BPM to ensure that sufficient key performance indicators are established and maintained effectively. SOA, BPM and LSS are brought together by implementing an iterative improvement lifecycle (Skalle 2009). Further investigation into this combination of methods is required to analyse the merit for the inclusion of each particular technology. Moreover, no real data is provided to prove the worthiness of one method over another. It is suspected that the alternate combinations of methods may be beneficial, dependent on the particular context. For example, Kim (2008) suggests benefit in the combination of the SOA with the MDA for modelling distributed systems (Kim 2008).

In an SOA, services may also be combined. Composite applications are the innovative binding together of various available web services. In the MDA, tools need not generate the required code statically, since all that is required is to generate the associated *invocations* for the required services which enables dynamic code generation (Frankel 2006). This is essentially how BPM tools work, giving the business user the power to bridge to the process architect in the Software Engineering domain. These concepts of course present additional challenges for the industry. Since descriptions in the Web Service Definition Language (WSDL) are written in natural English, there is an issue of semantics in that different people are able to interpret services with different meanings attached to them. Also, when invoking a predefined service, information about the quality of the service is not necessarily specified, thereby leaving quality control as another challenge for this domain. The proposed solution to these issues is that the composite application producer has to have access to a sufficient amount of *machine readable metadata* in respect of the services. An associated benefit to being forced to author services in this manner is that an amount of design issues and constraint confliction is resolved before the service is ever made available.

2.4 Summary

Scacchi (2002) proposes that “the death of the traditional system life cycle model may be at hand” (Scacchi 2002). This is due to the arrival of process modelling solutions that are better aligned to the current operating environment of software development. The MDA has evolved as a “consequence” of the UML and is purported to be the last abstraction to human-computer interaction (Génova et al. 2005). In comparison with the alternative approaches discussed in Section 2.3, the MDA appears to offer the most promise as a framework to complete this human-computer interface whilst remaining broad enough to integrate with other approaches and continuing to be a popular focus in academia. The potential the MDA holds for the business

analyst means that “understanding MDA is going to be crucial for business analysts of the future” (Slack 2008). Therefore, this research gives focus to the MDA as such an interface.

From reviewing available literature, it has been found that the CIM is not central to most MDA implementations. MDA committed companies and products neglect the creation of the CIM and related transformations (Ambler 2007; Kabanda and Adigun 2006; Karow and Gehlert 2006; Phalp et al. 2007), thereby highlighting the insignificance associated with it. “Successful technologies are those that are in harmony with users’ needs” (Shneiderman 2002) and, in consideration, RE and the connection to the MDA appears to be vital for the future of both the MDA and BPM. Process orientation is not the same as building object oriented software systems. “Object orientation is about building and maintaining IT systems; BPM is about building and maintaining business processes, and putting those capabilities in the hands of ordinary business people who get work done through their work processes – not their “objects”” (Bushell 2005). The ideals of the RAD could be beneficial in application to the CIM since a suitable platform for the development of software systems that are tightly linked with the business process is provided. The RAD can be used to describe iterations, concurring role activity and evolving processes, all of which are difficult to describe using software modelling techniques. Although research is limited, it is suggested here that considerable enhancement to the MDA could be made addressing concerns raised in Section 2.1 with the inclusion of RAD descriptions, allowing for “the opportunity to integrate humanistic and mechanistic processes” (Harrison-Broninski 2005c). Solutions may involve multiple language selection and therefore, some interoperable solution between modelling language and transformation technique is required for applicability to the MDA since “no single language can be adapted to all application domains”(Jouault and Kurtev 2006), supported by Rombach (1988). It is suggested that a solution will be achievable once a specification can be provided that is machine readable, as well as human understandable (Lautenbacher et al. 2007). Since any such method in determining a resultant system should carry the outright support of involved parties, the automation of transformations into design is likely to also be important for consideration.

Ultimately, the MDA lacks a formalised requirements model (Karow and Gehlert 2006) and there is a real danger in confusing business and software artefacts by “building a model of the real-world and then using it as a specification of the software system, producing a system that needlessly matches the structure of the real-world” (Génova et al. 2005), supported by Easterbrook (2003), Nuseibeh and Easterbrook (2000). This is evident in research that goes into extracting design models from CIM definitions, i.e. CIM-to-PIM transformations that do not account for specification and the system boundary. Focus therefore should be given to the MDA framework and how it might be extended to account for these concepts. This can be achieved by investigating the CIM in terms of how appropriate the phase is at delivering user requirements and available solutions in achieving a better integration of requirements within the MDA. It is thought that specification ought to be central to the MDA, and the information delivered in the CIM should be complete in terms of both business and software needs.

Chapter 3

Research Overview

In this chapter an overview of the research process is given and discussion relating to research paradigms is provided before a methodology is selected to define the direction of this research. Specific objectives identified to approach this research are highlighted in Section 3.4. Tasks relating to proposed objectives are then identified in turn and discussed in further detail with respect to the chosen methodology in view of answering the research question identified in Chapter 1.0, relating to aims 1 to 4.

Lubbe (2003) suggests that “an important step any researcher should take is establishing a framework in which to conduct the research” (Lubbe 2003) and, therefore, to achieve the research objectives it is important to first gain an understanding of the foundations of research. Figure 3.1.1 gives an overview of the research process used to realise the knowledge that the objectives seek to attain; the research process is detailed in the subsequent sections of this chapter.

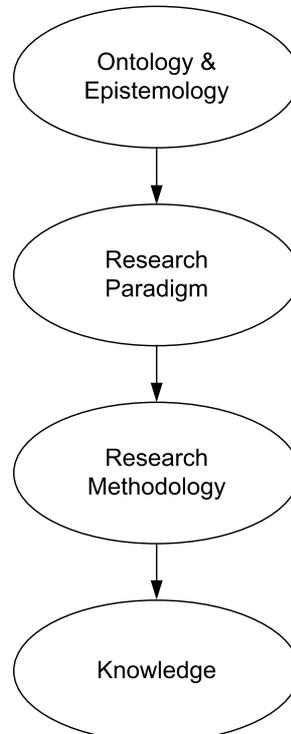


figure 3.1.1, overview of the research process (Source: adapted from Shelmerdine (2010)).

3.1 Ontology & Epistemology

Ontology is the metaphysical study of the nature of the reality in which things exist; that is, the study of being. It is important to understand the very foundation of a topic in order that a “psychological schizophrenia” be avoided (Hampton 2004). “Thinking is the epistemological path to conceptual comprehension. Knowing is the metaphysical path to apprehension – to the acceptance of a concept as true or valid” (Cobern 1993). The suggestion therefore is to study the foundation of both what exists (the metaphysical), and that which could exist (the epistemological). From reviewing literature in Chapter 2.0, it is clear that the MDA has been established from the domain of Software Engineering, within which philosophy might be contrary to what is required of the MDA in application to the business domain. Therefore, the study is focussed on accounting for this application. To achieve ontology, an understanding of the relationship between the philosophical nature of knowledge and the reality to which it is exposed is required (Shelmerdine 2010). Epistemology is the philosophical study of the nature of knowledge and the relationship between that knowledge and reality. Therefore, knowledge of the MDA and associated implementations form the foundation of this epistemological journey. The central focus is the application of this knowledge to the reality of the academic and business domain and therefore, a suitable research paradigm is required to investigate this relationship.

3.2 Research Paradigm

There is nothing so practical as a good theory (Gill and Johnson 1997).

Kolb, Rubin et al. (1979) provide a useful research framework in examining the relationship between theory and practice in the form of an experimental learning cycle (see figure 3.2.1).

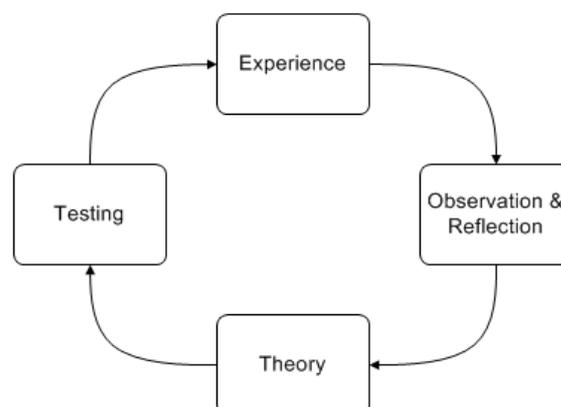


figure 3.2.1, experimental learning cycle (Source: adapted from Kolb et al. (1979)).

Interventionism is concerned with gaining “learning and knowledge through making deliberate interventions in order to achieve some desirable change in the organisational setting” (Braa and Vidgen 1999). This forms the basis of action research where “researchers both observe and participate in the phenomena under study” (Baskerville 1997). Action research started to gain prominence in information systems in the 1990s (Baskerville 1999) and it is the central motivation in the case that the outcome of this research will enforce a fundamental change within the MDA community via an approach to which the practice of the MDA will be improved, adding intellectual, academic and practical value. In simplistic terms, action research is about “learning by doing” (O'brien 2001). However, such fundamental change is unlikely to occur in the short term. There is scarce availability of accurate facts; quantitative data is limited and a sufficient test arena to produce worthy output from deduction is unavailable. However, knowledge is available in the form of existing and potential MDA experiences and, therefore, an inductive methodology can be followed in order that theory might result from the study of such experiences. It was thought that considerable opportunity existed in that the MDA definition and technical representation in the software development domain could be compared and contrasted upon to discover the theoretical reasoning which is cause for concern in the application of MDA to business users in defining requirements and specification. Further to this, soft cases could be used to verify the application of this research and therefore, an interpretivist approach to research is to be followed. To achieve this, a suitable methodology is required and the next section of this chapter is directed at identifying and discussing the relative merits of several interpretivist methodologies to identify those offering the most potential in application to the subject matter.

3.3 Research Methodology

3.3.1 Phenomenology

Phenomenology is a philosophical method used in qualitative research to examine the experience of an individual in terms of phenomena; the presumption being that the subject has no preconception of the subject matter (Husserl 2001). “The purpose of the phenomenological approach is to illuminate the specific, to identify phenomena through how they are perceived by the actors in a situation” (Lester 1999). This method is usually formed of directed interviews with small groups of individuals with a view to gaining an overall appreciation for the individual experience and the dynamics within the group. However, Phenomenology does not permit “piori” coding in analysing central themes, requiring time to be expended on individual cases (King 2006) which could make this methodology difficult to employ in consideration of available resources.

3.3.2 Ethnography

Ethnography is an anthropological method that can be used to study groups of people and the interactivity between them, and to some degree overlaps with phenomenology (Lester 1999). There are four characteristics presented in the *Manifesto for Ethnography* that distinguish ethnography from other qualitative methodologies. They are: "The recognition of the role of theory as a precursor, medium, and outcome of ethnographic study and writing... The centrality of "culture"... A critical focus in research and writing...[and] An interest in cultural policy and cultural politics" (Willis and Trondman 2002). Typically, observations involve groups of individuals over an extended period of time with focus on cultural aspects and since there is no opportunity to observe a group of individuals involved within the MDA environment, this method is rejected as impractical in consideration of the scope of the research.

3.3.3 Grounded Theory

Grounded Theory was first described in 1967 in Glaser and Strauss (1967) as a sociological method whereby the behaviour of groups of people is systematically observed and the collected data is codified in order that theory is derived from such observation to explain phenomena. An example of how grounded theory can be utilised in information systems research is given in Shannak (2009). Focus groups could be used whereby a concept tool is produced to accommodate any enhancement to the MDA framework and be used to further verify results. However, as previously noted, this is impractical since there is no availability of a sufficient test environment from which individuals could be observed to gain results that would be useful in providing insight with relation to the research question.

3.3.4 Template Analysis

Template Analysis is a method described in King (2006) and is concerned with the codification and thematic analysis of data that can be used where there is the assumption that "there are always multiple interpretations to be made of any phenomenon" (King 2006). Codes (numbered data labels) represent themes that are derived from the analysis of text. Often organised hierarchically to describe relationships between themes, the emphasis of Template Analysis is on "flexible and pragmatic" coding (King 2006). *Template Analysis* differs from other methodologies such as Grounded Theory - which is typically realist and prescriptive, and Phenomenology - which requires greater attention be placed on the individual (King 2006). Since this research is directed at resolving a problem between individuals, it could be appropriate to combine this technique with others, especially in consideration of resolving student feedback regarding the application of extending the MDA.

3.3.5 Surveying

Qualitative survey methods became popular in the 1980s (Marsland et al. 1998) and can be employed to broaden the scale of the target population with relation to research. It could be suggested that much failure is derived from models being presented to high-level managerial stakeholders, rather than lower tier technical representatives or system users. The project is based on an understanding of the stakeholder definition and how software developers can adapt to the paradigms of business. Qualitative surveys could be directed at a mixed sample of business users with the target sample population including members of the business community from both technical and non-technical backgrounds. As per recommendations (Deveaux et al. 2005), a randomised sample would be selected; this selection would then be stratified to increase the representation of the population (Saunders et al. 2003). A methodological framework for combining quantitative and qualitative survey methods is presented in Marsland et al. (1998), however, the extent to which results of qualitative and/or quantitative survey techniques could be useful in consideration of the available knowledge were deemed negligible because of the limitation of scope with regards to a sufficient test arena.

3.3.6 Interviewing

“Throughout the social sciences including language-based studies, interviews are widely used as data collection instruments” (Ikeda 2007). Processes could be modelled in different formats and presented to stakeholders or general users for analysis and data capture. Upon presentation, interviews could take place in a directed format with opportunity for the interviewee to make clear their opinions, along with their responses to pre-defined questions. Techniques relating to how to overcome issues of data quality, specifically in consideration of interviewer and interviewee bias are demonstrated in Saunders et al. (2003). However, as with previous methods described which employ the use of the interview as a research method, such as phenomenology and ethnography, a suitable environment to utilise the interview research method on a useful scale is unavailable.

3.3.7 Case Study

The case study method can be used to address the research question directly via one or more case studies relating to the subject matter and is used in information systems research for “exploratory investigations, both prospectively and retrospectively, that attempt to understand and explain phenomenon or construct theory” (Perry et al. 2004). Cases can be described as hard, soft or action cases (Braa and Vidgen 1999). The case study approach can be used to gather “data with which to develop grounded theory” (Lubbe 2003) and can allow for multiple avenues of data collection with the central notion being to identify themes particular to the case in hand, or across multiple cases. However, it is argued that gathering multiple evidence is not

necessarily able to prove (or disprove) a theorem (Yin 1994). In consideration of the construction of case studies, focus can be given to areas of the research context; exploratory questions; validation; ethical issues; data gathering and analysis; publishing; and reviewing (case replication) (Perry et al. 2004). "As a research strategy the case study research method is a technique for answering who, why and how questions" (Lubbe 2003). This is emphasised by Yin who writes that "case studies are the preferred method when (a) "how" or "why" questions are being posed, (b) the investigator has little control over events, and (c) the focus is on contemporary phenomenon within a real-life context" (Yin 2008), which is particularly suited to the type of qualitative knowledge available with regards to this research. The objective here is to understand the relationship between the business and software use of the MDA; to understand *who* the users of the MDA are, *why* it is being used and *how*. Since the problem is set in a real-life context, i.e. that of the MDA and involved organisations, little control over events is available. A good example of case study research in information systems is given in Card et al. (1986) where six characteristics of *good design* are studied (Card et al. 1986) against programming modules.

3.4 Knowledge

The problem highlighted in Chapter 2.0 from the analysis of the current state of the art seems to be about gaining an understanding of, and bridging the gap between, various stakeholders and software developers in the context of the MDA. Once a software system is presented to a user (as a prototype), a significant understanding of the system and an idea of the available potential is gained (Kavis 2008), enabling development. Therefore, by illustrating the underlying strategies and goals in a manner to which they can be understood (such as a model), clear requirements might succinctly follow.

Several objectives are identified in this section directed at achieving the aims outlined in Section 1.2, within the context of the research question. The described problem is multi-faceted and therefore, the selection of research methodology reflects these objectives to best accommodate each. It is thought that, whilst any single research methodology might give invaluable insight to the academic and commercial application of the area under consideration, the results obtained would be difficult to verify without a degree of conjecture. Therefore, from reviewing available research methods in the previous section, a mixed inductive approach, involving specifically a combination of theoretical, case and thematic analysis, is considered to be of benefit to this investigation and is outlined in the subsequent section of this chapter. The approach considers propositions to be directly verifiable by comparing and contrasting results obtained via multiple methodologies, allowing for findings to be triangulated in a complementary manner (Brannen 2005).

3.4.1 MDA Evaluation

The literature review, included in Chapter 2.0, was an ongoing process throughout the duration of this project, with a view to keeping up to date with current research and ensuring that developments within the context of this research are adequately addressed. "Requirements understanding problems inevitably lead to poor customer-supplier relationships, unnecessary re-works, and overruns in costs and/or time" (Elliott and Raynor-Smith 2000). Chapter 2.0 reveals that the MDA appears to neglect a RE perspective (Ambler 2007; Kabanda and Adigun 2006; Karow and Gehlert 2006; Phalp et al. 2007), and is therefore failing to sufficiently bridge between business stakeholders and the software developers. This forms the basis of this research and the point from which the first two objectives are defined for investigation in achieving aim 1.

Aim 1: To examine the definition of the CIM within the MDA and consider the appropriation of it as an interface with the business user for defining requirements in MDA notations.

In Chapter 4.0, a complete and thorough analysis of the MDA will be conducted in the context of user requirements for assessing the adequacy of the CIM in catering for requirements and specification. The first objective is related to the proposition that RE and BPM are somewhat disconnected from the MDA and will need to be verified.

Objective 1: Examine the connection between the MDA and business.

Section 4.1.1 will provide a theoretical analysis of the MDA by reflecting on the literature relating to the connection between the MDA and the business user, with any areas for concern being provided as evidence to support the argument, leading to the second objective.

Objective 2: Determine the sufficiency of the CIM at delivering requirements to the MDA.

Section 4.1.2 will build upon these findings to discover exactly what is required of the PIM. This will be addressed by applying a sample case study relating to a web-based cinema ticketing system adapted from Wa and Leong (2004). Forward and reverse engineering techniques will be developed specifically to be applied to the case study to discover whether what is required by the PIM is at conflict with what is described in requirements documentation.

The purpose of these objectives is to identify any associated difficulties with the accessibility of the MDA in defining business requirements for further investigation and evaluation in achieving aim 2.

3.4.2 Investigate Requirements Solution

Previous objectives are focussed on evaluating the suitability of the MDA as an interface to the business user, leading to the next proposition that requirements solutions may be better suited at facilitating stakeholder involvement and be adaptable to the architecture, and hence aim 2.

Aim 2: To discover how other modelling techniques which are accessible to the business user, might be integrated with the MDA in terms of method and notation, with the focus on transformation and traceability.

In Chapter 5.0, notations and techniques are to be applied to requirements documents and integrated with the MDA framework. In particular, the transformations between business and software models (that is CIM-to-PIM and CIM-to-CIM) are to be addressed. It is first suggested that requirements solutions are not adequately supported by the PIM and transformations are unavailable and therefore, the next objective to be achieved is defined as follows.

Objective 3: Investigate how requirements can be supported by the PIM.

In Section 5.1.1, CIM-to-PIM transformations are to be investigated. This will be achieved via the selection and theoretical analysis of what is offered by a typical business analysis technique and one that is supported by the MDA. As a language of business to model behaviour, it is thought that the RAD could be used to enhance the MDA in comparison with the UML Activity Diagram, a behavioural modelling notation available in the MDA. Soft cases relating to a simple order processing system and traditional musical jukebox system will be created to be used in an attempt to understand the phenomenon experienced in examining the notations and develop theory to explain them. Next, attention is to be directed at examining CIM-to-CIM transformations with the proposal that requirements solutions could be supported by the CIM and provide a better interface to the business user in comparison to techniques already supported by the MDA and thus, the next objective.

Objective 4: Examine how useful the CIM is at describing requirements.

Section 5.1.2 will approach CIM-to-CIM transformations via the case study methodology for a simplified travel reservation system case study adapted from Silver (2008d). In the MDA, the CIM can be represented via the BPMN. A common approach to specification in RE (and part of the UML) is the Use Case diagram. A CIM is to be represented using those notations to discover the perceived differences between what is required by a CIM and what is useful to the business user. A review is to be conducted of each on the suitability and applicability, and a discussion provided on how the notations and techniques may or may not complement one another.

This approach is proposed to provide a greater understanding of the requirements techniques most effective and adaptable to the MDA via transformation methods and lay foundation for the derivation of extensions for aligning the business strategy into the analysis and design phases of the MDA.

3.4.3 Investigate Potential Extension Mechanisms

Upon completion of objectives 3 and 4 and in consideration of aim 3, the investigation and construction of a mechanism to extend the MDA framework defined by the OMG (2003b) using requirements knowledge could begin.

Aim 3: To extend the framework of the MDA to account for specification within the CIM.

It is suggested that business modelling tools and RE concepts, that are available to organisations in industry and academia, can be integrated within the MDA to make a clear connection between the CIM and PIM. Therefore, objective 5 is directed specifically to achieve aim 3.

Objective 5: Derive mechanisms to adequately capture and realise requirements within the MDA.

Outcomes from previous objectives will be used to define theoretical mechanisms to extend the MDA to better facilitate the application of business logic. The notion is that the RAD, combined with specification theory, could be candidate to integrate with the MDA. Reasoning behind the extension mechanisms will be discussed in Chapter 6.0, with the resulting extension being illustrated in Chapter 7.0 via an order processing worked example, which will be created for this purpose.

3.4.4 Verify Extension Mechanisms

The last objective identified to address the research question is concerned with authenticating previous outcomes, specifically those relating to the definition of an extended MDA framework and method in terms of aim 4.

Aim 4: To determine the academic and commercial value of extended mechanisms.

The proposed mechanisms are suggested to enhance the MDA, making it accessible to the business user, whilst providing adequate support for the software user in terms of both academic and commercial validity. Objective 6 is concerned with achieving that aim.

Objective 6: Verify the proposed mechanisms to extend the MDA.

The research is to be presented in an academic setting with written feedback being collected to verify the application in the domain of academia (Cobern 1993). Three methods used in moving from analysis to specification and design (including the proposed mechanisms) will be presented to Honours level students of the Business Processes and Requirements (BPR) unit on the Software Systems framework at Bournemouth University, with written feedback being given in consideration of the proposed methods in order to complete a study of observations made. Template Analysis is to be used in Chapter 8.0 to analyse individual student feedback in order to understand phenomena associated with techniques in moving from analysis to specification and design from the student perspective and to discover whether or not proposed extensions are accessible and viable to them. Academic verification does not address the commercial suitability of the proposed extensions and, therefore, the verification is expected to extend into the factual domain (Cobern 1993). Since a sufficient test environment to build and execute a complete MDA implementation from the extended mechanisms is unavailable, an alternative way to verify the application of proposals is required. A collection of business process documents relating to a commercial case study, based upon The Club at Meyrick Park, Bournemouth, will be created and applied to the extended mechanisms, and available MDA tools and techniques, in Chapter 9.0. After manual application to the proposed extensions, QVT, the OMG standard for defining transformations between MOF metamodels, will be drawn upon to define transformation relations based on previous findings. Tools of the Visualise All Model Driven Programming (VIDE) initiative (VIDE 2009) will then be used to demonstrate the commercial application of transformations resulting from the extended mechanisms.

This approach is expected to allow for propositions relating to the extended mechanisms to be directly verified, with an output of academic and commercial data comparable in conclusion to the evidence derived from previous aims and objectives.

3.4.5 Software Support for Research

Analysis of attributes and values account for the objectives established in this chapter and conclusions are made, drawn from the investigation with relation to the scope of the project. The overall project performance is evaluated in Chapter 10.0 and any difficulties or limitations identified, along with an explanation of how they were managed with direction to further research and follow up studies being given. The objective here is not to fully automate the construction of requirements and system models for specification and, therefore, it would be quite appropriate for models to be hand drawn. However, this research utilises graphical modelling software to enhance the visual experience. Data sets and graphical representation are to be provided via the functionality of Microsoft Visio, Excel, and the VIDE PIM Prototyping Tool (PPT). Data set organisation, content exploration and the management of ideas and interpretations relating to the thematic analysis of

student feedback will be demonstrated via the functionality of NVivo from QSR International. Microsoft Word is selected as a tool to assist the production, with EndNote being used to manage and publish referencing in the Bournemouth University Harvard System style.

3.4.6 Ethical, Health & Safety and Risk

Since this research is theoretical in nature, there are no major ethical, health and safety or risk considerations, and therefore they are discounted as such. Appropriate assessments and analyses are to be made before student interaction and commercial collaboration opportunities are exploited; it is expected that no concerns be raised regarding these activities.

Chapter 4

Adequacy of the CIM

4.1 Examination of the CIM definition within the MDA and the appropriation of it as an interface with the business user for defining requirements in MDA notations

The first aim identified in Section 1.2 is directed at examining the sufficiency of the MDA for interfacing with the business user in defining requirements in the software process. As outlined in Section 3.4.1, this has been achieved by completing a theoretical analysis of literature in consideration of notions regarding the applicability of the MDA in Section 4.1.1, followed by a determination of requirements delivery within the MDA, giving focus to the CIM in Section 4.1.2. Findings are then drawn together and placed in the context of the MDA in Section 4.2.

4.1.1 The Connection between the MDA and Business

From examining the available literature in Chapter 2.0, academic opinion regarding the MDA was found to range from those that support the ideals of the MDA to those that do not. Therefore, initial investigations looked to review related literature from several angles with a view to uncovering the realisable value of the MDA, and hereby termed the hurdles to the MDA.

A striking observation made in Chapter 2.0 was that, within the MDA, there are alternating view points of *Translationists* and *Elaborationists*, a heavy reliance on the BPMN and the UML, a lack of guidance on how the CIM might be employed, vendor lock-in and a neglect of concepts central to RE. In contrast, there are those in opposition highlighting major benefit in the application of the MDA, being brought about by the separation of business and software platform concern (Blanc 2009; Brown 2004a; Kabanda and Adigun 2006; Slack 2008). The unified approach to software systems development also assists with communication between the business and software domain, and suggests facilitating an agile environment in which changes can be reflected and demonstrated to the customer through the use of transformation and traceability. Cost savings and an early release date are proposed to be achieved through the MDA framework (Meservy and Fenstermacher 2005) with a better standard of quality being given in that an automated transformation tool is less error prone in comparison with a human exertion (Kleppe et al. 2003). Above all, the notion of

traceability provided by the MDA ought to ensure that information contained within business and software models are no longer resigned to archived repositories.

Under the MDA, a model is seen as an abstraction on code, and code as an abstraction on the machine. With each abstraction, albeit within code or models, “the level of abstraction at which the developer operates” increases (Mellor and Balcer 2002). Assembly languages led to the evolution of 3rd Generation languages and object orientation, the next step along this evolutionary process is proposed to be the MDA. The MDA views models as a programming language which is abstracted above 3GLs and object oriented techniques (Brown 2004a). This is demonstrated in figure 4.1.1.1, which was developed as part of this investigation and extends previous findings described figure 2.1.1.1.

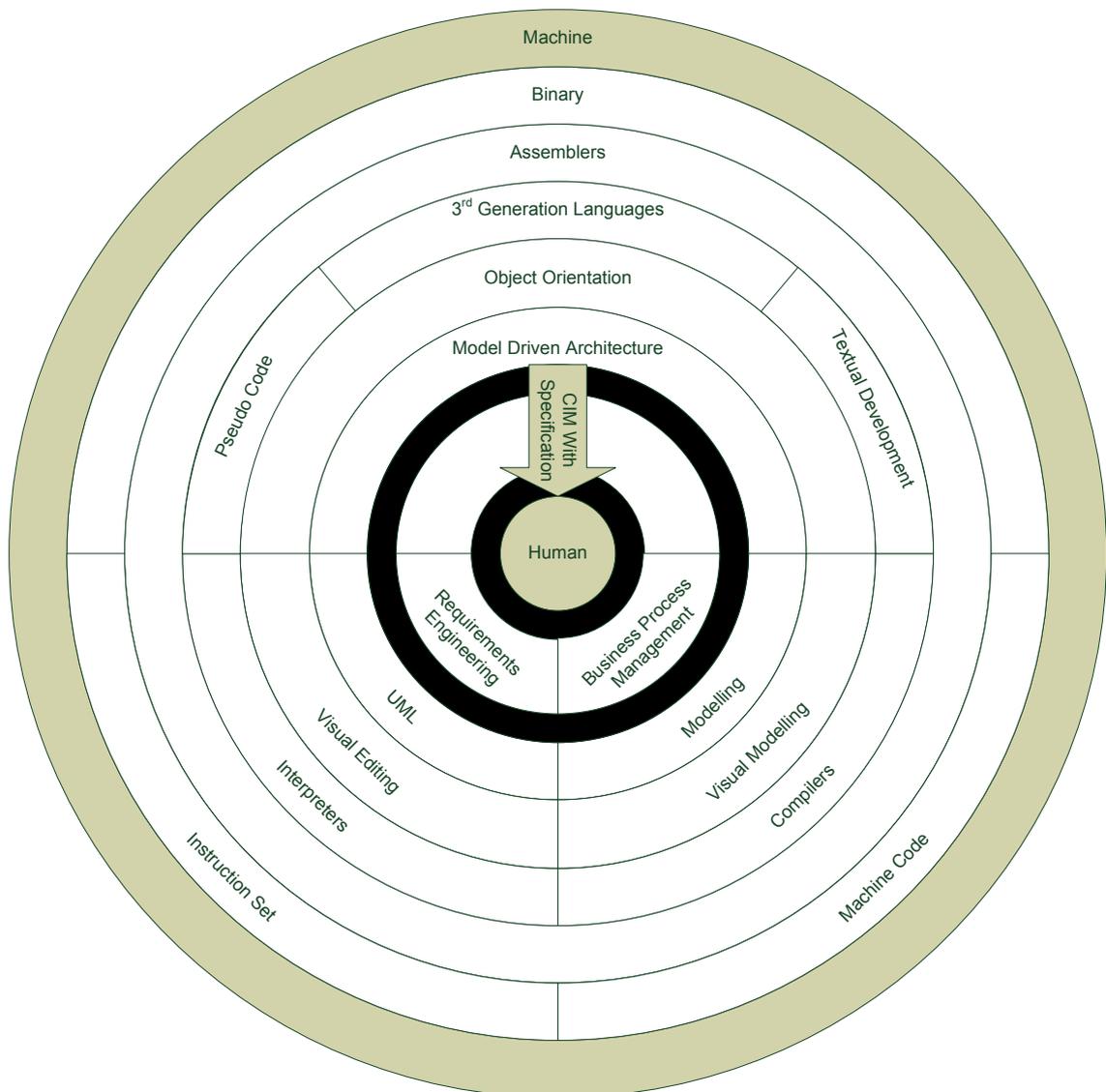


figure 4.1.1.1, technological development of Software Engineering and human interactivity (Source: developed from Brown (2004a)).

Figure 4.1.1.1 is by no means a complete representation of the technological development of Software Engineering and human interactivity. The objective here is to demonstrate a clear picture of where related technologies are situated within the greater scheme.

Historically, the modelling element of software development has been somewhat separate from the production of solution systems. What the MDA does is to ensure that modelling *is* software development by taking the level of abstraction a step higher than coding, thereby reducing any gap that existed between modelling and development technologies. "IT improves productivity through streamlining of process and enhances efficiency and effectiveness of individual workers as well as groups through connectivity that it offers" (Al-Neimat 2005). However, research by the Standish Group (2001) has demonstrated the success of IT projects as a continual difficulty, citing issues relating to "the people [rather] than the technology itself... [such as] estimation mistakes, unclear project goals and objectives, and project objectives changing" (Al-Neimat 2005) as key to such failings. Indeed, "success rates on BPM projects far exceed other more traditional software development projects" (Lombardi 2008).

The technicalities of RE and BPM are found to be instrumental in business-software interaction (Musschoot 2010), but inadequately implemented in industry, leaving a gap in knowledge which is cause for concern in consideration of such failed or failing IT projects (shown in figure 4.1.1.1 as filled black circles). "Process models capture the organisation of activities performed by both users and machines, and the required cooperation between them" (Roberts 1988). Business process approaches (rather than any IT description) can offer a shared understanding for both process and information in an IT support environment (Musschoot 2010). Where the MDA may close the gap between modelling and development, little is perceived to have been achieved in bridging from the modelling element into the domain of business, let alone from there to the end user. RE "is primarily a communication, not technical, activity" (Wiegiers 2000) and current business processes are affected as new ways of completing tasks are discovered through the use of IT and strategic alignment. "Most organisations now understand the need to improve and manage their processes. However, few do so in a disciplined, ongoing basis" (Dowdle and Stevens 2009). The UML "falls into the cracks between technical people (developers, architects) and non-technical people (business analysts, project managers, etc). [The] UML is too technical for non-technical people, and not technical enough for technical people" (Ford 2009). These complexities cause a communication barrier between business users and Software Engineers (Frank 2002), leading to a "semantic gap... between customers and system developers" (Elliott and Raynor-Smith 2000). This gap in knowledge can lead to a decrease in customer satisfaction as implementations bear no resemblance to the requirements they were designed to satisfy and nothing proposed by the CIM accounts for that. Organisations should allow for business and IT co-dependency, "the willingness and ability of these two groups to collaborate is, in the end, critical to success at every stage of implementation" (Lombardi 2008). Haan (2009) supports this argument by describing reasons why MDD

could be dangerous as an approach to software development, including design inflexibility and a lack of industrial experience in practice (Haan 2009). Therefore, before moving forward in systems software development, organisations should be careful about jumping on the MDA bandwagon and sceptical regarding claims made in academia.

A clear definition of the CIM constitution is not described and the MDA lacks a formalised behavioural or functional model. Indeed, "requirements models do not have a proper counterpart in the MDA terminology" (Karow and Gehlert 2006). In order that the accessibility of the MDA be sufficiently addressed, RE and BPM technologies ought to be considered. Ideally, a requirements model would interface between elements of the real-world and the software modelling domain via the CIM. Semantics of requirements is an important issue for the Requirements Engineer since much of what is defined as a requirement is in natural English. Figure 4.1.1.1 extends these revelations by suggesting that, with correct implementation, it may be possible to enhance the MDA with specification theory with the goal of providing sufficient connection between the end user and software developer, thereby bridging "the uncomfortable gap between specification and implementation" (Jackson 1982). Despite this, Requirements Engineers have an added problem in that there are multiple customers, i.e. stakeholders (Coughlan and Macredie 2002; Kavakli 2004; Nuseibeh and Easterbrook 2000; Peixoto et al. 2008; Phalp and Jeary 2010; Phalp et al. 2007; Sommerville 2004), that may have a valid, and sometimes invalid, contribution to make, all of which is required to be managed and refined so that one succinct requirements document might follow.

4.1.2 The Sufficiency of the CIM at Delivering Requirements to the MDA

Moving from requirements to a CIM which can adequately account for those requirements presents many challenges to the software developer. Business rules may be required to be applied across a multitude of applications, which in turn requires that the meaning of information and the method of exchange must be clearly defined, including behavioural constraints and restrictions. Also, it is important to understand that alternate business processes can be defined to achieve the same specific outcome and therefore, more than one requirement can represent what is ultimately the same thing (Macek and Richta 2009). Above this, the imposed rules must be continually and consistently validated by the domain experts involved to ensure changes are implemented. This highlights the need for clarity in defining business rules or requirements. In Section 4.1.1 it was suggested that specification might be a useful enhancement to the MDA and, in consideration of this, two alternate approaches with relation to the CIM are discussed here as part of this research. The first is that the CIM could represent an abstraction of the same system to which design models abstract from. Therefore, requirements should be included as such, rather than representing those relating to the PD or the business process, which abstract on a completely different reality (the real-world). If specification is not held central to the MDA, the formalisation of the CIM should extract elements that are not related to the system to be built, so that the CIM represents a synthesis of elements related to the same

solution system. The *raison d'être* being that the MDA is a software development framework, and therefore should only contain software models; business models and concepts of RE ought to be accounted for further upstream from the CIM. The second views requirements and specification as integral to the CIM, rather than external to the MDA and focuses on such PD elements as part of the CIM.

To understand the essence of the CIM and gain an appreciation of these differing views, documentation relating to the requirements and associated MDA implementation for a sample case study adapted from Wa and Leong (2004) were examined. "Requirements... can be divided into two categories, functional and non-functional. Functional requirements describe what a system should do in response to specific stimulus... Non-functional requirements include performance and system constraints that affect development and design" (STSC 2003). Specifically, this section examines the detail of all such requirements that are (or should be) included when these artefacts are forward and reverse engineered (transformed) to and from one another in the context of the MDA. For the purpose of this study, reverse engineering is described as the de-construction of a model into originating documentation; where forward engineering is the re-construction of a model from originating documentation. This is proposed to be useful in enabling an understanding the constitution of a typical CIM; that is, what is needed in terms of a requirements definition; and what is required in terms of the subsequent PIM phase. The example case study relates to a simple web-based cinema ticketing system and the requirements for the proposed system are given in table 4.1.2.1.

Number	Original Requirement
1	The web page (e.g. the time table page, the main page) will be generated automatically according to the data in the database.
2	A way in which the customer can create its own account (member registration) is to be provided.
3	A way in which the users (both customer and staff) can log on to the system to perform different operations is to be provided.
4	A way in which the customer can modify its own data is to be provided.
5	A way in which the customer can commit an order by just clicking the seat (which is shown on the screen) and insert some card data (some simple operation) is to be provided.
6	A way in which the customer can cancel the order and get the refund is to be provided.
7	A way in which the customer can check the ticket record according to the transaction number is to be provided.
8	A way in which the staff can use the system to add data (e.g. film descriptions) to the database is to be provided.
9	The system can verify the data before the transaction is processed.
10	The system can generate the time table automatically (by just inputting the length of the film) or the time table is set by the staff (2 operating modes for the staff to insert data).
11	The system can generate some statistical information according booking and ticket selling records.
12	Users can check film data by clicking on a certain film on main page (e.g. the cinema which will show these films).
13	Users can check cinema data by clicking on a certain cinema on the main page (e.g. to locate which film is now showing).
14	The web-based system needs about 30 interfaces (web pages) to handle all the functions.
15	Since two or more customers may request for the same seat at the same time, the system needs to remove the chance for two customers getting the same seat.
16	A simulated bank account is to be used within the prototyping process.
17	Lots of the customers will buy tickets in ticket box and those which use the web-based system will still need to take the ticket into the ticket box. So, the online ticket booking service and refund service will be stopped 1 hour before the show time. To do so, we can reduce the chance of 2 people book the same seat and also reduce the time for buying a ticket.
18	The new system needs to be compatible with the existing ticket selling system (original) in the ticket box, because the web-base system and the original system will run on the same time and use the same database.

Note: Requirements that are non-functional are given in bold.

table 4.1.2.1, original requirements relating to the sample case study (Source: adapted from Wa and Leong (2004)).

A discussion follows regarding implications involved in the process of forward and reverse engineering documentation related to this case within the context of the MDA and applicable notations (UML Class and Activity Diagrams).

4.1.2.1 Class Diagram

At PIM level, the Class Diagram relating to the case study (given in figure 4.1.2.1.1) was examined from the perspective of the CIM, that is, to reverse engineer back from the Class Diagram in the hope of discovering what the original requirements within the CIM *should* contain. This is to determine the level of sophistication required of the CIM to produce a Class Diagram useful in the design phase of the MDA.

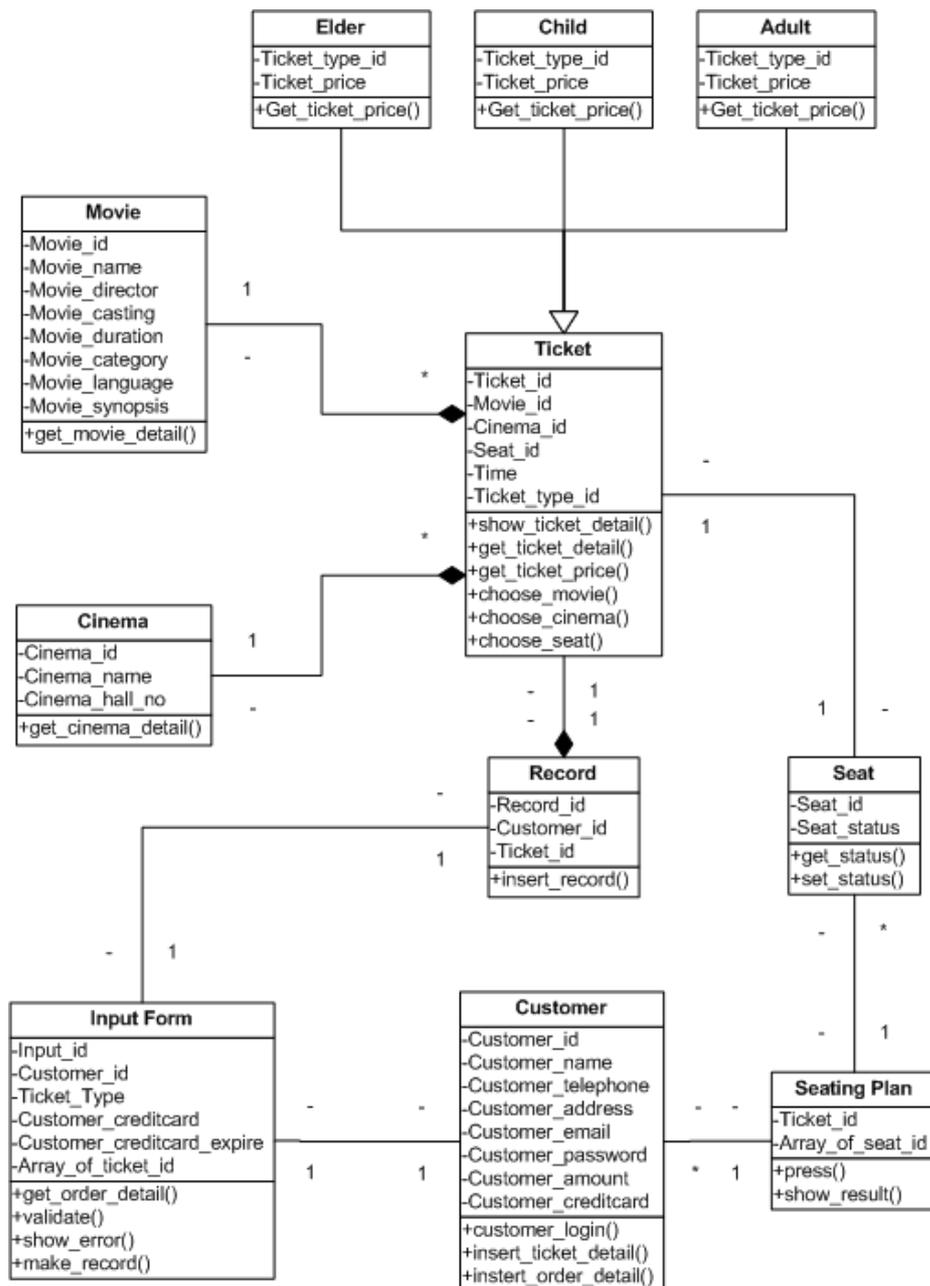


figure 4.1.2.1.1, Class Diagram relating to the sample case study (Source: Wa and Leong (2004)).

The basis of reverse engineering the Class Diagram was that every element should have an associated CIM level requirements translation (Dubielewicz et al. 2006). The CIM, in this case, therefore, represents a copy of the PIM (Génova et al. 2005). This is to say that the output CIM directly reflects the Class Diagram, and therefore retains the structural design of that model and highlights the type of detail required within the CIM to enable a transform from one to the other to be possible. A selection of requirements derived from the Class Diagram of figure 4.1.2.1.1, which would be required to be accommodated within the CIM in this case, are given in table 4.1.2.1.1 below.

Number	Requirement
1	ELDER is a type of ticket
2	ELDER must have a TICKET TYPE ID
3	ELDER must have a TICKET PRICE
4	ELDER must be able to get the ticket price
5	CHILD is a type of ticket
6	CHILD must have a TICKET TYPE ID
7	CHILD must have a TICKET PRICE
8	CHILD must be able to get the ticket price
9	ADULT is a type of ticket
10	ADULT must have a TICKET TYPE ID
11	ADULT must have a TICKET PRICE
12	ADULT must be able to get the ticket price
13	TICKET refers to 1 SEAT
14	TICKET is composed of a RECORD
15	TICKET belongs to 1 CINEMA
16	TICKET refers to 1 MOVIE
17	TICKET must have a TICKET ID
18	TICKET must include the MOVIE ID
19	TICKET must include the CINEMA ID
20	TICKET must include the SEAT ID
21	TICKET must include the TIME
22	TICKET must include the TICKET TYPE ID
23	TICKET must be able to show ticket detail
24	TICKET must be able to get ticket detail
25	TICKET must be able to get ticket price
26	TICKET must be able to choose movie
27	TICKET must be able to choose cinema
28	TICKET must be able to choose seat
29	MOVIE is composed of many tickets
30	MOVIE must have a MOVIE ID
31	MOVIE must have a NAME
32	MOVIE must have a DIRECTOR
33	MOVIE must have CASTING
34	MOVIE must have a DURATION
35	MOVIE must have a CATEGORY
36	MOVIE must have a LANGUAGE
37	MOVIE must have a SYNOPSIS
38	MOVIE must be able to get the movie detail
39	CINEMA is composed of many tickets

40	CINEMA must have a CINEMA ID
41	CINEMA must have a NAME
42	CINEMA must have a HALL NUMBER
43	CINEMA must be able to get cinema detail
44	RECORD refers to 1 TICKET
45	RECORD refers to 1 INPUT FORM
46	RECORD must have a RECORD ID
47	RECORD must have CUSTOMER ID
48	RECORD must have TICKET ID
49	RECORD must be able to insert a RECORD
50	SEAT refers to 1 TICKET
51	SEAT refers to 1 SEATING PLAN
52	SEAT must have a SEAT ID
53	SEAT must have SEAT STATUS
54	SEAT must be able to set status
55	SEAT must be able to get status
56	SEATING PLAN refers to many SEATS
57	SEATING PLAN refers to many CUSTOMERS
58	SEATING PLAN must have TICKET ID
59	SEATING PLAN must have an ARRAY OF SEAT ID
60	SEATING PLAN must be able to press
61	SEATING PLAN must be able to show result
62	CUSTOMER refers to a SEATING PLAN
63	CUSTOMER refers to an INPUT FORM
64	CUSTOMER must have a CUSTOMER ID
65	CUSTOMER must have a NAME
66	CUSTOMER must have a TELEPHONE
67	CUSTOMER must have an ADDRESS
68	CUSTOMER must have an EMAIL
69	CUSTOMER must have a PASSWORD
70	CUSTOMER must have an AMOUNT
71	CUSTOMER must have a CREDITCARD
72	CUSTOMER must be able to log in
73	CUSTOMER must be able to insert ticket details
74	CUSTOMER must be able to insert order details
75	INPUT FORM refers to 1 CUSTOMER
76	INPUT FORM refers to 1 RECORD
77	INPUT FORM must have an INPUT ID
78	INPUT FORM must have a CUSTOMER ID
79	INPUT FORM must have a TICKET TYPE
80	INPUT FORM must have a CUSTOMER CREDIT CARD
81	INPUT FORM must include CUSTOMER CREDIT CARD EXPIRE
82	INPUT FORM must have an ARRAY OF TICKET ID
83	INPUT FORM must be able to get order detail
84	INPUT FORM must be able to validate
85	INPUT FORM must be able to show error
86	INPUT FORM must be able to make record

table 4.1.2.1.1, requirements derived from reverse engineering the Class Diagram of the sample case study.

Focussing on the number of requirements highlighted, the results obtained from examining them in relation to the original requirements are given in table 4.1.2.1.1.

Original requirements identified by the requirements documentation (of which were non-functional)	Requirements extracted from the Class Diagram (of which were non-functional)	Original requirements realised by the Class Diagram (of which were non-functional)	Original requirements not realised by the Class Diagram (of which were non-functional)
18 (3)	86 (0)	4 (0)	14 (3)

table 4.1.2.1.1, analysis of the number of requirements identified from the Class Diagram.

By extracting requirements from the Class Diagram, it was found that many more requirements could be identified than those detailed in the requirements documentation provided (a total of 86), demonstrating that there is significant distance between what is required of design in the PIM and what is described by requirements and the CIM. Surprisingly, fewer requirements identified in the requirements documentation were fully realised in the final Class Diagram (such as those that are non-functional), which perhaps highlights the lack of influence requirements models can have on design models.

The next stage in this investigation was to take the documents relating to requirements and forward engineer them into a Class Diagram, to be comparable with the original Class Diagram of the same system. In completing this task, it was noted that the natural English requirements documentation did not contain enough detailed information to facilitate complete forward engineering and therefore, it is not included here. The requirements documentation was insufficient and incomplete from an object oriented perspective and attributes were neglected. Since business users are unfamiliar with *objects*, they are unlikely to specify detail in terms of classes, attributes, operations and the like. Again, although being specified in the requirements documentation, a mechanism was not available to incorporate non-functional requirements within the context of the Class Diagram.

4.1.2.2 Activity Diagram

In Section 2.1.4, it was seen that behavioural models are not central artefacts of the MDA. The behavioural model in the MDA could be represented at the PIM level via the Activity Diagram and a similar investigation was conducted by looking at the requirements documentation and Activity Diagram (given in figure 4.1.2.2.1) relating to the case study in the same manner.

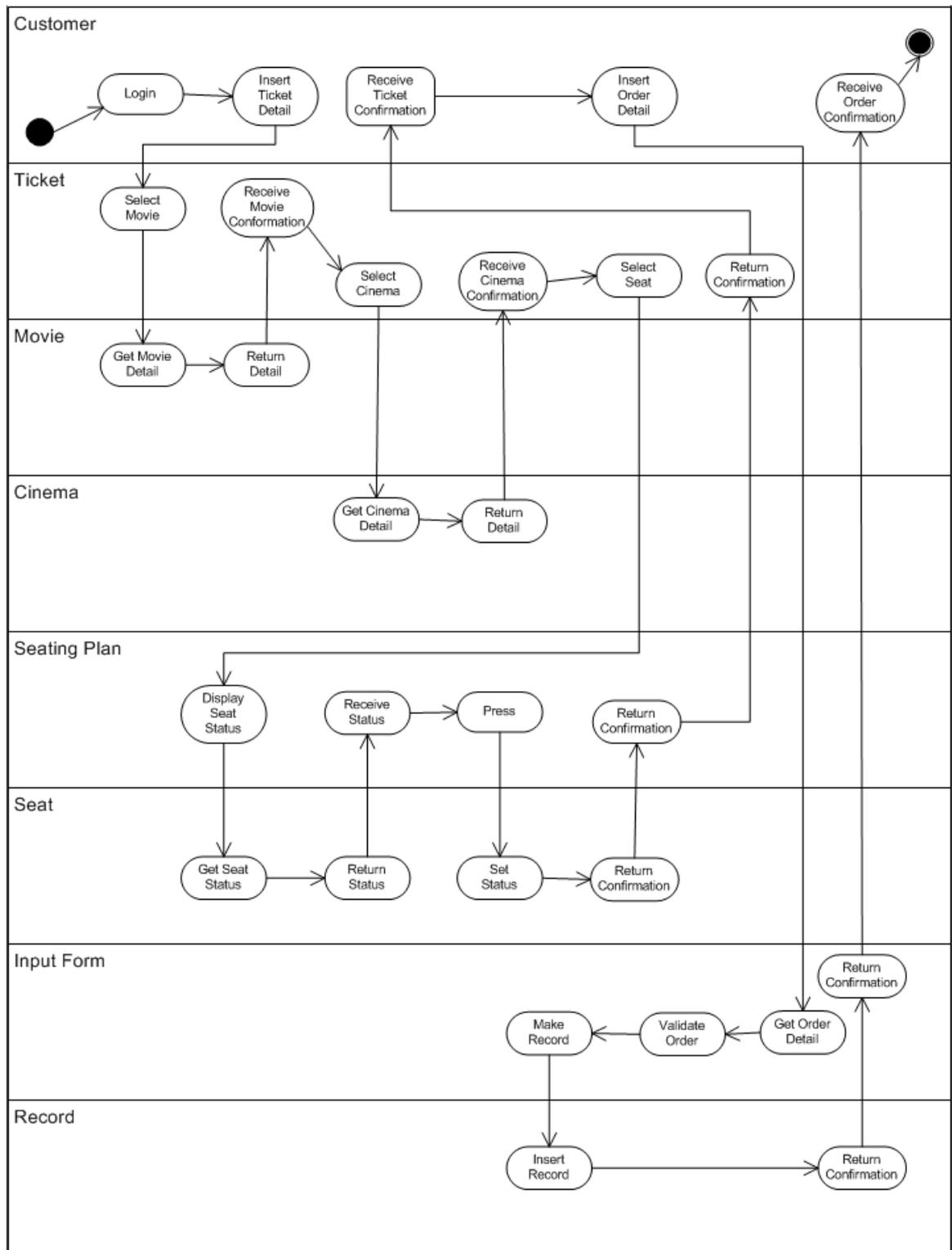


figure 4.1.2.2.1, Activity Diagram relating to the sample case study (Source: adapted from Wa and Leong (2004)).

The basis of reverse engineering for the Activity Diagram was that every behavioural string should have a CIM level representation that is understandable to the business user, and translatable to a PIM representation. The requirements extracted from the Activity Diagram of figure 4.1.2.2.1 and required to be accommodated within the CIM are given in table 4.1.2.2.1 below.

Number	Requirement
1	Customer must be able to login
2	Ticket Detail must be able to be Inserted
3	Movie must be able to be Selected
4	Movie Detail must be able to be Retrieved
5	Movie Detail must be able to be Returned
6	Movie Detail must be able to be Received
7	Cinema must be able to be Selected
8	Cinema Detail must be able to be Retrieved
9	Cinema Detail must be able to be Returned
10	Cinema Detail must be able to be Received
11	Seat must be able to be Selected
12	Seat Status must be able to be Displayed
13	Seat Status must be able to be Retrieved
14	Seat Status must be able to be Returned
15	Seat Status must be able to be Received
16	Press must be able to be Completed
17	Seat Status must be able to be Set
18	Set Seat Status must be able to be Confirmed
19	Set Seat Status must be able to be Received
20	Select Seat Confirmation must be able to be Received
21	Ticket Confirmation must be able to be Returned
22	Ticket Confirmation must be able to be Received
23	Order Detail must be able to be Inserted
24	Order Detail must be able to be Retrieved
25	Order Detail must be able to be Validated
26	Record must be able to be Created
27	Record must be able to be Inserted
28	Insert Record Confirmation must be able to be Returned
29	Insert Record Confirmation must be able to be Received
30	Order Confirmation must be able to be Returned
31	Order Confirmation must be able to be Received

table 4.1.2.2.1, requirements derived from reverse engineering the Activity Diagram of the sample case study.

It immediately became evident that, in similar as with what was found regarding the Class Diagram, the Activity Diagram is somewhat inadequate at specifying non-functional requirements and significant alterations would be required to accommodate the majority of original requirements, perhaps reflecting the inadequacy of the approach. Table 4.1.2.2.2 outlines the results obtained from the forward and reverse engineering of the Activity Diagram.

Original requirements identified by the requirements documentation (of which were non-functional)	Requirements extracted from the Activity Diagram (of which were non-functional)	Original requirements realised by the Activity Diagram (of which were non-functional)	Original requirements not realised by the Activity Diagram (of which were non-functional)
18 (3)	31 (0)	3 (0)	15 (3)

table 4.1.2.2.2, analysis of the number of requirements identified from the Activity Diagram.

Overall, the number of requirements generated from the Activity Diagram amounted to roughly the number of nodes contained within the diagram, which was not unexpected. The real interest of reverse engineering the Activity Diagram was that it could really enable the technician to address and refine the process itself, with respect to the system-to-be, and not just refine the business process in the business context, or replicate that business process as software design.

To forward engineer from the requirements, it was found to first be important to isolate those functions or behaviours that were not to be realised in the new system, thereby drawing attention to the specification and system boundary. This was not easily done since some activity might be replaced by the system, whilst others might be supported by the system or not required at all. It became evident that there would be some difficulty in distinguishing elements that were important and required by the system, from those that were not.

Furthermore, the imposition of object orientation compounds this distinction. Other, vaguer requirements immediately became a cause for concern with verification being required. The requirements were found to be insufficient in many respects and complete forward engineering was not possible and therefore, no forward engineered diagram is included. There appears to be excellent potential for deriving a PIM from concisely defined requirements via a behavioural model. However, the future of the Activity Diagram is somewhat clouded. It has been noted that "the Activity Diagram and Business Process Diagram are very similar and are views for the same metamodel, it is possible that they will converge in the future" (White 2004). This is even more probable now since the OMG have adopted the BPMN specification (OMG 2005, 2008a).

4.2 Extending the Model Driven Architecture with pre-CIM

Section 4.1.2 demonstrated that by reverse engineering artefacts of the MDA, elements that perhaps ought to be discovered from the analysis of the CIM, such as objects and attributes, somewhat conflict with what is defined in requirements documentation. Requirements documentation is typically not intended to address object oriented perspectives, which highlights that CIM definitions do not necessarily account for the unfamiliarity of software development paradigms in the business domain. A mechanism for transferring knowledge of non-functional requirements was also unavailable in consideration of the Class and Activity Diagrams and the importance of the system boundary was uncovered in distinguishing those functions required by the system from those that were not; all of which appears to not be explicitly considered by the

CIM. Therefore, for a union between business and the ideals of software development to be realistic in the context of the current architecture, the CIM would need to be an abstraction of the PIM and formalised via the description of requirements for the system-to-be, which is inadequate in terms of the business need because they are unaware of notions such as objects and attributes. It is argued that the definition of such a CIM could not be completed by the business user alone due to the complexities involved in defining a model that is so tightly aligned with software and thus, pre-CIM activities would be required to facilitate business user interaction (Jeary et al. 2008; Phalp and Jeary 2010).

In figure 4.2.1, the pre-CIM concept is introduced as part of this research and fused with the framework of the MDA and Jackson’s systems of prime concern (Jackson 1995) and associated development activities, thereby redefining the CIM and extending the MDA framework (Fouad et al. 2011). Pre-CIM refers to an area of activity that takes place before the CIM is produced and is proposed to account for requirements gathering activities such as elicitation, validation, traceability and change management. This extension suggests that the MDA is only addressing half of the story, and that PD analysis and specification ought to be integral to the CIM if it is to fully accommodate the need of the business and user community.

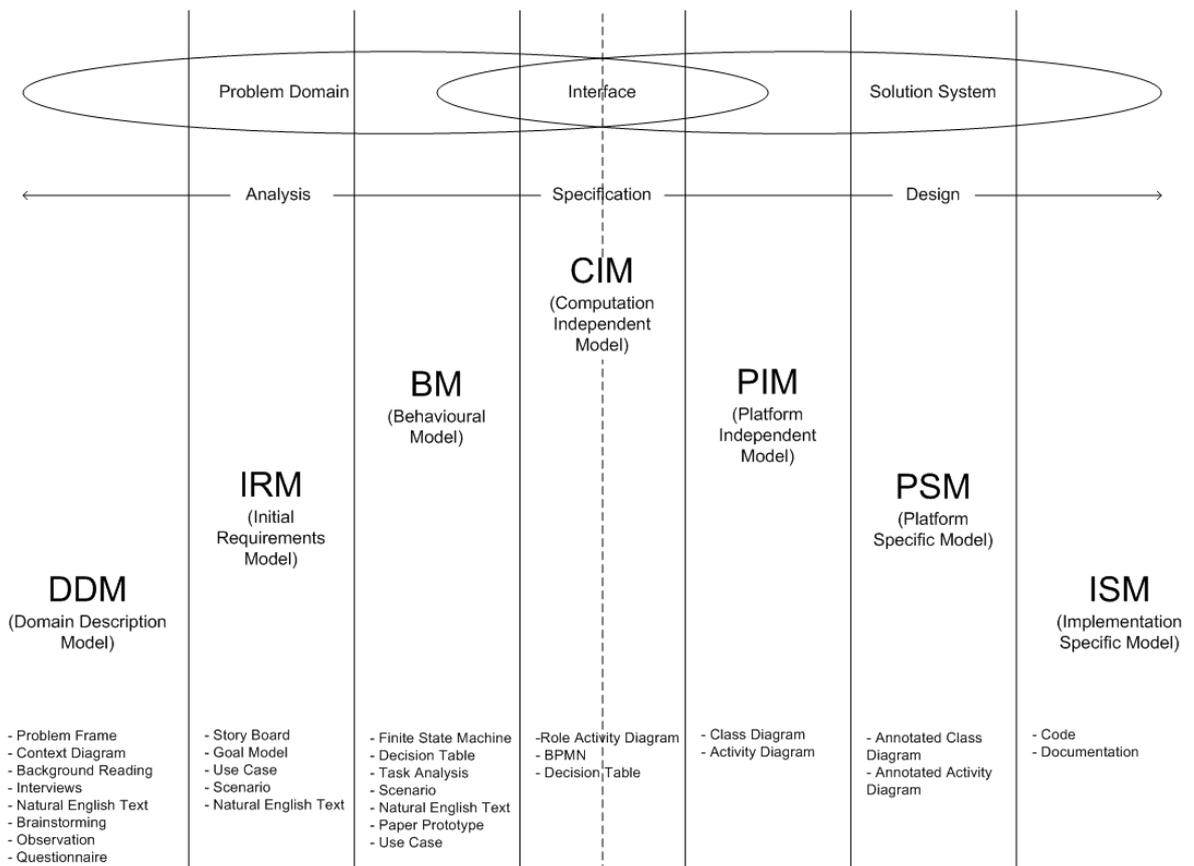


figure 4.2.1, the Extended Model Driven Architecture (xMDA), including pre-CIM activity (Source: developed from Bray (2004), Brown (2004a), Jackson (1995), OMG (2003b)).

Concern has been raised that the MDA neglects the development of the CIM (Ambler 2007). Since “the initial phases of Business Process Modelling projects, process discovery and documentation take up more than one third of the overall project time” (Jeary et al. 2008), it is reasonable to suggest that the MDA should direct greater attention to artefacts that are created upstream from the CIM, that is pre-CIM exploration, in order to reap benefits later in the project. By focussing elicitation on smaller issues, enhanced requirements gathering can be achieved. PD elements can be accounted for in pre-CIM activities of the Extended Model Driven Architecture (xMDA), outside of the software modelling world, as demonstrated in figure 4.2.1. The redefinition of the CIM as such interface and the introduction of pre-CIM activities look to enable MDA abstraction into the root of the PD, facilitating accessibility for the business user.

With emphasis on RE and BPM techniques, rather than concepts of systems engineering, business users are suggested to be able to express requirements to Software Engineers with resulting systems likely to be more akin with customer expectation. Thus, activities are divided into new categories of the Domain Description Model (DDM), the Initial Requirements Model (IRM) and the Behavioural Model (BM), and integrated with the MDA and the redefined CIM. The DDM is used to define the PD context as it is. The IRM is used to define any requirements (both functional and non-functional) to be imposed by the new system. The BM is then used to highlight specific functional requirements and the behaviour of the involved process. All pre-CIM activities are developed in the context of business, and not software. Therefore, xMDA is fully compatible with RE and BPM since no specific tool or technique is defined for any of these activities (only that they exist). For example, a business process model could be used to demonstrate the DDM, IRM and BM. By focusing only in domain specific requirements at this low level of abstraction, ambiguities and misunderstandings are ironed out as early as possible, at the same time as bringing specification closer (Bray 2002). Ideally, definitions at the CIM level will be free from complexities, understandable, and above all, helpful to both business users and Software Engineers. Moreover, such definitions ought to facilitate transformations (manual and/or automatic) to be appropriate for the MDA.

4.3 Summary

From examining the CIM within the MDA it was found that it is an inappropriate interface between business and software domains. That is, it appears to support original opinions in that it is not fit for purpose from a quality perspective of interfacing with the client to represent business requirements.

The theoretical analysis of the MDA in Section 4.1.1 demonstrated that whilst being applicable to the business domain, considerable distance between the MDA and the business user is evident. The neglect of RE concepts and the MDA reliance on notations grounded in software theory, such as the BPMN and the UML, were highlighted, with openings for extending the CIM to reach the business user being identified.

The example case study discussed in Section 4.1.2 supported this view by demonstrating that notions supported by the UML are unhelpful in describing PD elements, defining business requirements and facilitating business user understanding. The distinction between what is required within the system and what is part of the process (i.e. specification and the system boundary) was also identified as an important consideration in realising business value.

It is the task of the Requirements Engineer to transform business requirements into a richly defined requirements document, which in turn should be possible to be reflected in a CIM, enabling the overall presentation to a PIM that is more accurate, and more aligned with the requirements and strategy of the business user. Software modelling and theory are foreign to the business user (OMG 2003b) and therefore, the concentration on the fusion between the business and Software Engineering world is suggested to be the onus of the software developers (Shneiderman 2002), which is central to the distinction of the xMDA pre-CIM activities as described in Section 4.2. As noted in Chapter 1.0, requirements represent the desired effects a software system would have in the PD; the PD being the “part of the universe within which the problem exists” (Bray 2002). The application domain is the part of the universe within which the application (or solution system) will be applied. The interface between the two forms the specification of a software system, and it is this that lays foundation to extending the MDA.

Chapter 5

Situating Requirements within the CIM

In Chapter 4.0 it was found that what is required of the CIM is not in line with the expectation of business; the purpose of aim 2 is to address this.

5.1 Discovery of how other modelling techniques which are accessible to the business user, might be integrated with the MDA in terms of method and notation, with the focus on transformation and traceability

By giving focus to CIM-to-PIM and CIM-to-CIM transformations involving notations derived from the RE domain it is possible to demonstrate the application of requirements techniques within the context of the MDA.

5.1.1 PIM Support for Requirements (CIM-to-PIM)

The 2.0 specification of the UML general purpose modelling language used in the development of software systems brought about several enhancements on the 1.4 specification with relation to the Activity Diagram and the business process (Wohed et al. 2005). The Activity Diagram is an OMG behavioural model solution and provides a notation to model functional aspects of a process with a view do describing how the interaction of co-ordinated activities might take place. The technique is very structured and *well behaved*, providing a sequential view of the process. Activity Diagrams are therefore excellent in Software Engineering at describing processes that are automatic in nature, with objects at heart, for which they were intended to be used. The difficulty in delivering a system from the Activity Diagram is that the business user is not accustomed to concepts defined in the UML and the real-world of the business process is dynamic and experiences less sequential activity than are described in software interactions. From reviewing the available literature in Chapter 2.0, the RAD was found to be a business process modelling alternative, potentially useful for modelling in terms of human interactivity. Since these are common ideals between RE and BPM, and a starting point for business use, focus was given to the RAD as a business notation in investigating the natural fault-line between software systems development in MDA and business. This addresses the idea that techniques better aligned with the business need might be accommodated within the MDA in the development of quality software systems.

The application of the RAD first brought about the re-evaluation of what the notation of the RAD is actually defined to describe. To understand the application of the RAD, roles, associated nodes and the tight relationship with the business process, a RAD metamodel was extended from Badica et al. (2005) by using the notation of the MOF (OMG 2006a) to contain the construction rules and constraints implied by the RIVA modelling methodology described in Ould (2004c). The RAD metamodel is depicted in figure 5.1.1.1.

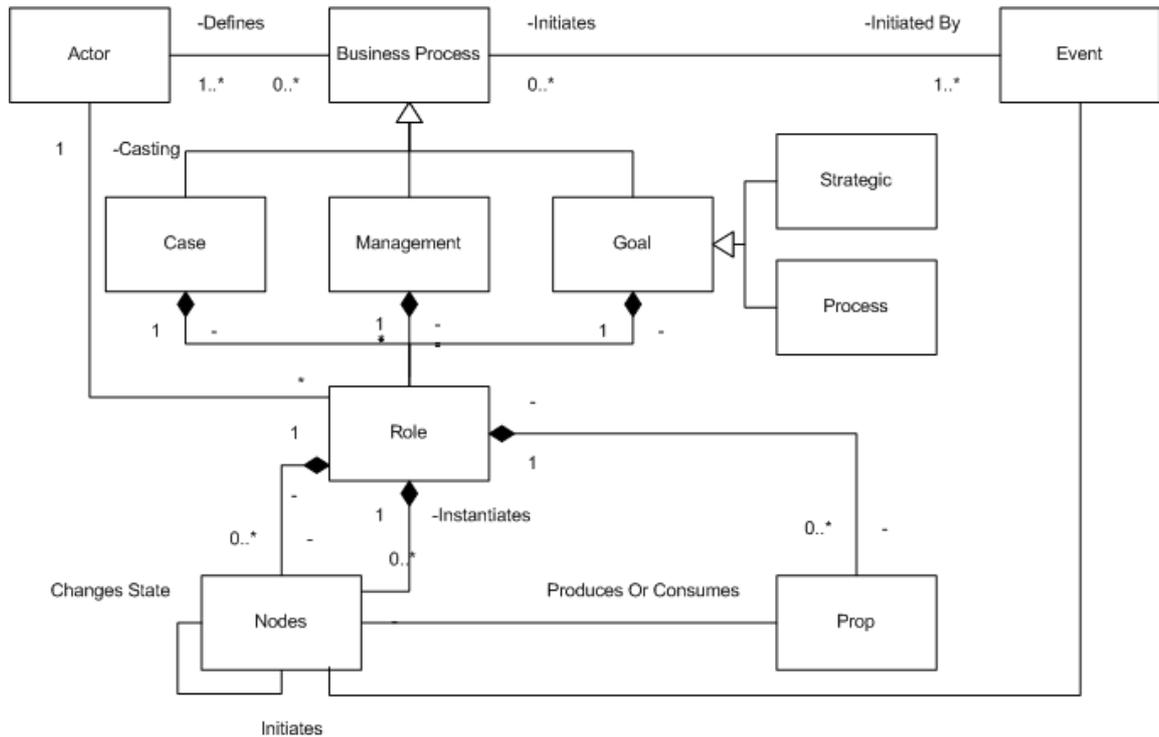


figure 5.1.1.1, the RAD metamodel (Source: developed from Badica et al. (2005), OMG (2006a), Ould (2004c)).

As figure 5.1.1.1 demonstrates, roles are composed directly from aspects of the business process itself, including individual case, management and goal related components. A goal in a RAD is likely to be represented as an end state, for example, *project has concluded* or *customer has completed system acceptance*. This direct relationship to business process is perhaps the unique point of the RAD, since it proposes a model representation directly linked to requirement deliverables of the business process, which might be applicable to CIM and used in the MDA.

The metamodel also proposes that each process has a process owner. When a project has an interdependent relationship with another project, “the person who is acting as implementation owner for each project may also be a requirement sponsor on other projects” (Harrison-Broninski 2006b). This is to ensure that the underlying requirement of one project is implemented succinctly in other associated projects and identifying

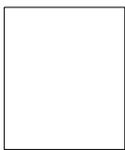
these process owners is an important consideration before moving into the development of single and multi-projects.

The RAD is genuinely related to the *real-world* of business, including concepts which naturally cater for event driven behaviour, which may be an ideal platform for software systems which are tightly linked with the business process to be developed from. The difficulty in integrating the RAD with the MDA is that the intended purpose of the RAD is to describe the business process and not software systems. Research in this area is very limited, however, it is thought that the framework of the MDA might be enhanced by including RAD descriptions at the CIM and/or PIM level allowing for software development to be integrated with business processes. To evaluate the differences between the Activity Diagram and the RAD, an informal comparison of the two notations was made. Specifically, rules and notational elements associated with each were analysed to ascertain the suitability of these notations with respect to CIM-to-PIM transformations in terms of the perceived understanding a typical business user would obtain from published materials. Therefore, the rules were extracted as part of this research from educational publications for the Activity Diagram (Stevens and Pooley 2000) and RAD (Ould 2004c). These publications were purposefully selected since they are designed specifically to educate and aid understanding of such notations in terms of application.

5.1.1.1 The UML Activity Diagram Rules

In this section, rules associated with the Activity Diagram are described, accompanied by a pictorial description of the notational element for further discussion.

5.1.1.1.1 Activity Partition (Swim-Lane)



- AD1. The **Activity Partition** is optional.
- AD2. The notation for **Activity Partition** is a rectangle.
- AD3. The label of an **Activity Partition** instance must not be null.
- AD4. An **Activity Partition** must contain all nodes associated with that partition.
- AD5. An **Activity Partition** may include multiple nodes.
- AD6. An **Activity Partition** may have multiple entry **Transitions**.
- AD7. An **Activity Partition** may emit multiple exit **Transitions**.
- AD8. Multiple **Activity Partitions** may appear in a single diagram.

5.1.1.1.2 Start



- AD9. The **Start** node is not optional.
- AD10. The notation for a **Start** node is a single-filled circle.
- AD11. The label of a **Start** node may be null.
- AD12. A **Start** node may not have any entry **Transition**.
- AD13. A **Start** node must emit only a single exit **Transition**.
- AD14. Only one **Start** node may appear in a single diagram.

5.1.1.1.3 Activity



- AD15. The **Activity** node is optional.
- AD16. The notation for an **Activity** instance is that of a box with rounded corners.
- AD17. The label of an **Activity** instance must not be null.
- AD18. An **Activity** instance must include an **Action**.
- AD19. An **Activity** instance must have at least (but not limited to) one entry **Transition**.
- AD20. An **Activity** instance must emit at least (but not limited to) one exit **Transition**.
- AD21. Multiple **Activity** nodes may appear in a single diagram.

5.1.1.1.4 Transition



- AD22. The **Transition** node is not optional.
- AD23. The notation for a **Transition** is that of an arrow.
- AD24. The label of a **Transition** may be null due to identification being evident from the completion of the preceding **Activity**.
- AD25. Only a single **Transition** may be involved between two nodes in any instance.
- AD26. A **Transition** instance may traverse an **Activity Partition**.

-
- AD27. A **Transition** instance may include an **Action**.
 - AD28. A **Transition** instance may include a **Guard**. A **Guard** is defined to ensure access to the succeeding activity is based on satisfied pre-requisites.
 - AD29. **Guards** are not required to be satisfied.
 - AD30. The label of a **Guard** instance must not be null.
 - AD31. The label of **Decision Diamond** exit **Transitions** must not be null.
 - AD32. The label of **Decision Diamond** exit **Transitions** must be contained within squared brackets.
 - AD33. A **Transition** instance may not receive any entry **Transition**.
 - AD34. A **Transition** instance may not emit any exit **Transition**.
 - AD35. Multiple **Transition** nodes may appear in a single diagram.

5.1.1.1.5 Decision Diamond



- AD36. The **Decision Diamond** node is optional.
- AD37. The notation for a **Decision Diamond** is a diamond shaped box.
- AD38. The label of **Decision Diamond** may be null.
- AD39. The label of **Decision Diamond** may contain an **Activity**.
- AD40. A **Decision Diamond** is required to be satisfied.
- AD41. A **Decision Diamond** instance must have at least (but not limited to) one entry **Transition**.
- AD42. A **Decision Diamond** must emit at least (but not limited to) one exit **Transition**.
- AD43. Multiple **Decision Diamond** nodes may appear in a single diagram.

5.1.1.1.6 Synchronisation Bar



- AD44. The **Synchronisation Bar** is optional.
 - AD45. The notation for a **Synchronisation Bar** is that of a wide horizontal column.
 - AD46. The label of a **Synchronisation Bar** node may be null.
 - AD47. **Transitions** and **Activity** succeeding a **Synchronisation Bar** cannot be executed until all entering **Activity** and **Transitions** are in a state of completion (join).
 - AD48. **Transitions** and **Activity** succeeding a **Synchronisation Bar** are executed in parallel once the **Synchronisation Bar** is in a state of completion (fork).
-

-
- AD49. A **Synchronisation Bar** instance must have at least (but not limited to) one entry **Transition**.
- AD50. A **Synchronisation Bar** must emit at least (but not limited to) one exit **Transition**.
- AD51. Multiple **Synchronisation Bar** nodes may appear in a single diagram.

5.1.1.1.7 Stop

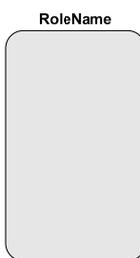


- AD52. The **Stop** node is not optional.
- AD53. The notation for a **Stop** node is a single-filled circle, surrounded by an additional outer circle.
- AD54. The label of a **Stop** node may be null.
- AD55. A **Stop** node must have only one entry **Transition**.
- AD56. A **Stop** node may not emit any exit **Transition**.
- AD57. Only one **Stop** node may appear in a single diagram.

Note: The Activity Diagram does not normally include events.

5.1.1.2 The RIVA RAD Rules

5.1.1.2.1 Role



- RAD1. The **Role** is not optional.
- RAD2. The notation for a **Role** is a shaded block with rounded edges.
- RAD3. The label of a **Role** instance must not be null.
- RAD4. A **Role** instance may be persistent.
- RAD5. The label of a **Role** instance with pre existing instances may be denoted by a ✓.
- RAD6. The label of a **Role** instance with exactly x pre existing instances may be denoted by ✓x.
-

-
- RAD7. The label of a **Role** instance with an indeterminate number of pre existing instances may be denoted by a $\surd n$.
- RAD8. The label of a **Role** instance must appear immediately above or below the shaded block.
- RAD9. A **Role** instance may include multiple nodes.
- RAD10. A **Role** instance must contain all nodes associated with that **Role** instance.
- RAD11. A **Role** instance must contain all appropriate **Props** to enable process completion.
- RAD12. A **Role** instance may overlap another **Role** (provided there is no ambiguity).
- RAD13. A **Role** overlap is a darker shade of grey.
- RAD14. A **Role** instance may contain many 'threads'.
- RAD15. A **Role** instance may instantiate any 'thread' at any time, depending on the event.
- RAD16. A **Role** instance may instantiate another **Role** instance.
- RAD17. Multiple instances of the same **Role** may appear in a single diagram.
- RAD18. A **Role** may have multiple entry **Interactions**.
- RAD19. A **Role** may emit multiple exit **Interactions**.
- RAD20. Multiple **Roles** may appear in a single diagram.

5.1.1.2.2 Independent Activity



- RAD21. The **Independent Activity** node is optional.
- RAD22. The notation for an **Independent Activity** instance is that of a black box.
- RAD23. The label of an **Independent Activity** instance must not be null.
- RAD24. An **Independent Activity** instance may terminate a **Role** instance.
- RAD25. An **Independent Activity** instance must have at least (but not limited to) one entry **State**.
- RAD26. An **Independent Activity** instance must emit at least (but not limited to) one exit **State**.
- RAD27. Multiple **Independent Activity** nodes may appear in a single diagram.

5.1.1.2.3 Looping, Line and Descriptor States (pre/post-conditions)



- RAD28. **State** nodes are optional.
- RAD29. The notation for a **State** is either a line (single or looped) or ellipse.
- RAD30. The label of **State** nodes may be null.

-
- RAD31. **State** nodes may merge together.
 RAD32. Multiple **State** nodes may appear in a single diagram.

5.1.1.2.4 Case Refinement (Alternatives)



- RAD33. **Case Refinement** is optional.
 RAD34. The notation for a **Case Refinement** is that of an upturned triangle, with each 'thread' being represented by its own triangle.
 RAD35. The label of a **Case Refinement** may be null.
 RAD36. The label of a **Case Refinement** is represented as a question.
 RAD37. The label of a **Case Refinement** may not contain an action.
 RAD38. The label of **Case Refinement** nodes may contain a probability figure.
 RAD39. A **Case Refinement** is not required to be satisfied.
 RAD40. A **Case Refinements** can be rejoined into a single state once the **Case Refinement** is in a state of completion (join).
 RAD41. A **Case Refinement** may be abbreviated to a single **Independent Activity** in simple cases.
 RAD42. A **Case Refinement** instance must have at least (but not limited to) one entry **State**.
 RAD43. A **Case Refinement** must emit at least (but not limited to) one exit **State**.
 RAD44. Multiple **Case Refinement** nodes may appear in a single refinement (N-way).
 RAD45. Multiple **Case Refinement** nodes may appear in a single diagram.

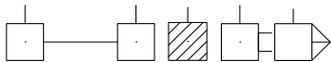
5.1.1.2.5 Part Refinement (Concurrency)



- RAD46. **Part Refinement** is optional.
 RAD47. The notation for a **Part Refinement** is that of an upright triangle, with each 'thread' being represented by its own triangle.
 RAD48. The label of a **Part Refinement** may be null.
 RAD49. A **Part Refinement** is not required to be satisfied.
 RAD50. Threads succeeding a **Part Refinement** are executed in parallel once the **Part Refinement** is in a state of completion (fork).
-

-
- RAD51. **Part Refinement** nodes can be rejoined into a single state once **Part Refinement** is in a state of completion (join).
- RAD52. A **Part Refinement** instance must have at least (but not limited to) one entry **State**.
- RAD53. A **Part Refinement** must emit at least (but not limited to) one exit **State**.
- RAD54. Multiple **Part Refinement** nodes may appear in a single refinement (N-way).
- RAD55. Multiple **Part Refinement** nodes may appear in a single diagram.

5.1.1.2.6 Interaction



- RAD56. The **Interaction** node is optional.
- RAD57. The notation for an **Interaction** is that of a horizontal line between two (or more) white boxes (part-interaction).
- RAD58. The notation to indicate the 'driving' force in an **Interaction** is that of a shaded white box.
- RAD59. The notation to indicate the 'one-to-one' **Interactions** with all members of the same **Role** is that of a white box with twin line indicators.
- RAD60. The notation to indicate the 'one-to-many' **Interactions** with all members of the same **Role** is that of a white box with a 'crow's foot' line indicator.
- RAD61. The label of an **Interaction** may be null.
- RAD62. A single **Interaction** may be involved between two or more nodes in any instance.
- RAD63. An **Interaction** instance may traverse any **Role**.
- RAD64. An **Interaction** instance cannot take place until each involved **Role** is in a ready **State** (pre-state synchronisation).
- RAD65. Once an **Interaction** instance has taken place, each involved **Role** is in a completion **State** (post-state synchronisation).
- RAD66. All involved **Roles** pass through the **Interaction** instance and the same time.
- RAD67. An **Interaction** instance may change the **Prop** of a **Role**.
- RAD68. An **Interaction** instance may include an **Action**.
- RAD69. A conditional **Interaction** may be abbreviated to a single **Action** in simple cases.
- RAD70. An **Interaction** node must have only one entry **State**.
- RAD71. An **Interaction** node must emit only a single exit **State**.
- RAD72. Multiple **Interaction** nodes may appear in a single diagram.

5.1.1.2.7 Role Instantiation



- RAD73. **Role Instantiation** is optional.
- RAD74. The notation for **Role Instantiation** is a crossed box.
- RAD75. The label of a **Role Instantiation** may be null.
- RAD76. A **Role Instantiation** node must have only one entry **State**.
- RAD77. A **Role Instantiation** node must emit only a single exit **State**.
- RAD78. Multiple **Role Instantiation** nodes may appear in a single diagram.

5.1.1.2.8 Trigger



- RAD79. The **Trigger** node is optional.
- RAD80. The notation for a **Trigger** node is a single-filled arrow.
- RAD81. The label of a **Trigger** node may not be null.
- RAD82. A **Trigger** node changes the **State** of a **Role**.
- RAD83. A **Trigger** node may change the **Prop** of a **Role**.
- RAD84. A **Trigger** node may have zero to one entry **State**.
- RAD85. A **Trigger** node must emit only a single exit **State**.
- RAD86. Multiple **Trigger** nodes may appear in a single diagram.

5.1.1.2.9 Replication



- RAD87. The **Replication** node is optional.
- RAD88. The notation for **Replication** is a fan symbol.
- RAD89. The label of **Replication** may not be null.
- RAD90. The label of **Replication** must contain some indication of the number of **Replications** to take place.
- RAD91. The **Replication** node defines a **Part Refinement** in a ‘thread’.
- RAD92. A **Replication** node must have only one entry **State**.

-
- RAD93. A **Replication** node must emit only a single exit **State**.
 RAD94. Multiple **Replication** nodes may appear in a single diagram.

5.1.1.2.10 Undefined



- RAD95. The **Undefined** node is optional.
 RAD96. The notation for **Undefined** is a spring symbol.
 RAD97. The label of **Undefined** may be null.
 RAD98. An **Undefined** node may have zero to one entry **State**.
 RAD99. An **Undefined** node must emit only a single exit **State**.
 RAD100. Multiple **Undefined** nodes may appear in a single diagram.

5.1.1.2.11 Prop

- RAD101. The **Prop** node is optional.
 RAD102. The notation for a **Prop** is a textual list in the **Role** box.
 RAD103. The label of a **Prop** may not be null.
 RAD104. The **Prop** may not interact with any other nodes within the **Role** box.
 RAD105. Multiple **Prop** nodes may appear in a single diagram.

5.1.1.2.12 Stop



- RAD106. The **Stop** node is optional.
 RAD107. The notation for a **Stop** node is a single horizontal line.
 RAD108. The label of a **Stop** node may be null.
 RAD109. A **Stop** node must have at least (but not limited to) one entry **State**.
 RAD110. A **Stop** node may not emit any exit **State**.
 RAD111. Multiple **Stop** nodes may appear in a single diagram.

Note: Events play an important part in the RAD.

5.1.1.3 Comparative

The rules presented in the previous two sections are indicative of the complexity and difference between the two techniques essentially used to capture the same thing from alternating perspectives. The first thing to become immediately evident is that basic rules pertaining to the Activity Diagram and RAD appear more alike than contrasting. They are both concerned with modelling the functionality of a process and follow similar constructs. However, where there are differences, these differences are important, and the focus of this section is on those differences.

5.1.1.3.1 Activity Partition Vs Role & Role Instantiation

The notion of a RAD role is significantly more important than that of an activity partition. The activity partition is the simple segregation of activity nodes, and has an optional existence. In a RAD, the role is central, and in fact *essential* to the process. This is because roles are the only entities that can provide functionality and interaction. A role is the conceptualisation of real-world representations, whereas an activity partition has no such significance, and the use of the RAD notation allows the positioning of such a role within the process (including organisational hierarchies) to be seen as “much better than swim-lanes” (Ould 2004c). A role can persist throughout the lifetime of the process (as roles do in reality) enabling roles to be called upon in alternate processes and even be defined upon how many particular instances of that role might be in existence.

There is no such concept of activity partition instantiation in the UML. In a RAD, notation is available to demonstrate how a role, for example a *Divisional Director*, might instantiate an instance of another role, in this case a *Project Manager*, as and when new projects come to light, which is a direct reflection of real business processes, unlike the notion of the activity partition.

Another important, and perhaps overriding difference, is that activity partitions do not allow for more than one *thread* of activity within a single partition, focus is given rather to providing a sequential string of activity. In a RAD, the notion of a *hanging thread* is available. A *hanging thread* allows for continual change within a process and is activated on event, affecting the state conditions within the process. This is very useful in modelling alternative and exceptional situations. Indeed, this event-driven behaviour is an important reflection on the real-world where people are often observed “dealing with abnormal or unexpected situations” (Ould 2004c) within a process definition.

5.1.1.3.2 Start Vs Trigger

In a RAD, there is no corresponding notation for the start node of the Activity Diagram. In an Activity Diagram, the start node represents the beginning of some *thread* of activity, nothing more. In general, states are used to mark the start of activity *threads* within the RAD which enables this activity to begin once the condition of state is met. Specialisations can involve the undefined notation, where it is considered unimportant to know when a *thread* of activity may be initiated, only that it can be. As previously noted, a role might also be instantiated, which represents the start of a new role, where *threads* are allowed to begin and end with a natural start and stop state, without the need to represent this explicitly. The most explicit representation of a start node in a RAD is that of a trigger.

A trigger is state-based and can be used to begin a thread of activity, as and when, according to the event that might activate that trigger. The notation allows for a trigger to be defined by the user that changes the starting state of a *thread* within a role, enabling it to become active. For example, a trigger might be defined that instigates a *thread* of activity that chases any unpaid invoices at the end of each month. A trigger node is not limited to a single representative in one *thread* (or indeed role), the notation can be applied as many times, and in as many roles that are deemed necessary to represent the process. It is important to note that the start node of the Activity Diagram does not have any corresponding notation to manage the concepts introduced by the trigger node of the RAD.

The “prioritisation of user requirements is important” (Maguire and Bevan 2002) yet little is defined here regarding the prioritisation of trigger events in this context. For example, if a trigger was used as described above to ensure that unpaid invoices are chased at the beginning of every month, an exceptional event may occur and override the trigger (i.e. if the systems were down, there will be no system clock, and therefore the trigger may never activate). Such prioritisation may be important in the definition of the process.

5.1.1.3.3 Activity Vs Independent Activity

Activity and independent activity or action nodes are perhaps the most alike notations found in both techniques, both being used to describe some form of act to be performed. Indeed, as rule AD18 notes “An Activity instance must include an Action”. One defining difference is that an independent activity in a RAD has the power to destroy a role instance completely. It is evident that no such activity in an Activity Diagram has such a corresponding significance since an activity partition has no *real* value.

Phalp (1998) writes that “as with data flow approaches, processes may be decomposed into further processes, and so on”. A consideration for both activity and action is that every instance of each could be decomposed further. Therefore, it is apt to “always show whatever detail is appropriate to *that* model for *that* purpose”

(Ould 2004c), and indeed, the RIVA method provides guidance in achieving this. This is supported in Abeysinghe and Phalp (1997) where suggestion is made to decompose processes into essential entities to enable transformations. However, not much direction regarding abstraction levels within the Activity Diagram is available.

5.1.1.3.4 Transition Vs State & Interaction

An Activity Diagram transition is used primarily to connect nodes together, enabling progress to be followed from the start node, sequentially through to the natural conclusion of the stop node. Since no such notion of sequential flow is relevant in the RAD, the closest relatives are state and interaction nodes.

Once again, it is evident that a transition bears no resemblance to any *real* notion, whereas RAD states and interactions directly relate to real-life processes. A state line is representative of how far a process has progressed relative to time. An interaction expresses the collaboration between two or more roles. "Interactions are very rich in nature and we must be aware of that" (Ould 2004c). A transition, on the other hand, is simply a connector between two nodes. A transition can in fact *only* connect two nodes, whereas an interaction can occur concurrently with a multitude of alternate nodes and state lines can merge into single states, allowing for multiple entry and exit alternatives, thereby being less restrictive. Furthermore, not only can the notation for an interaction facilitate this, but it can also facilitate *one-to-one* and *one-to-many* interactions with all members of the process. All of which is unaccounted for in the Activity Diagram.

A redeeming quality found in Activity Diagram transitions is the inclusion of guards. A guard is defined to protect the succeeding activity sequence by establishing entrance pre-requisites, which are required to be satisfied. In a RAD, this is accounted for by the inclusion of case refinement and the state based nature of the technique. However, pre/post states do not have the same defining quality as conditions placed by guards.

5.1.1.3.5 Decision Diamond Vs Case Refinement (Alternatives)

RAD case refinement operates in much the same way as the Activity Diagram decision diamond, which is very much similar to the decision diamond of early flowcharting techniques. The decision diamond is used when a decision is to be made which could have numerous numbering outcomes; this is represented in the RAD case refinement in the form of a question to which the answer will activate a corresponding *thread* of activity. However, a decision diamond is required to be satisfied; a constraint not included RADs, which operate more akin to the aforementioned guards. If a pre-requisite cannot be satisfied, that *thread* will simply never activate. The case refinement can also be simplified in notation to a single independent activity, but only in simple cases.

5.1.1.3.6 Synchronisation Bar Vs Part Refinement (Concurrency) & Replication

The synchronisation bar and part refinement both cater for the parallel execution of multiple activity *threads* via *join* and *fork* mechanisms. Since the RAD also caters for the merging of states, the additional option for *rejoin* is offered.

The single most identifiable difference in terminology between the two alternate techniques is that of replication. The replication fan is used to define a part refinement in a RAD, to signify that the task is repeated for each and every available consideration. For example, *for each employee, conduct review*. "RADs are about the concurrent activity in the real-world" (Ould 2004c) and not the sequential view of software system design.

5.1.1.3.7 Stop Vs Stop

The stop symbol in an Activity Diagram has a greater significance than that of a RAD. In an Activity Diagram, the stop node signifies the end of that sequential string of activity and must be included on any Activity Diagram. In a RAD, it is not important to signify the end of a *thread* since conclusion is naturally reached via an end state, unless it is considered important by the modeller to explicitly illustrate it, for which the stop notation is used. Since multiple *threads* are allowed across multiple roles, multiple stop signage may appear anywhere, whereas in a single Activity Diagram, a single start and stop node is required.

5.1.1.3.8 Undefined & Prop

Undefined and the prop are notions that are present in the RAD, but have no representation in the Activity Diagram. The Undefined node is a notation of a spring symbol which signifies that a state will occur, irrespective to the measure that may (or may not) have brought the process into that state or that the *thread* moves into a state to which significance is unimportant to the process. The symbol can be used at the beginning, end, and throughout a process.

The prop is a resource a role might create/update, interrogate and/or pass between roles during an interaction. These props are integral to the role since, without them, tasks will be difficult, if not impossible, to complete. The integrity of the prop also helps to ensure that every role operates as a private information space with enough information to participate in the required process.

5.1.1.4 RAD Application for the MDA

It has been demonstrated that by simply looking at the basic rules involved with two alternate techniques used to model a process, albeit business or system, the difference between the notations can be significant. On the extreme it would be very difficult, if not impossible, to model concepts and notions imposed in one by utilising the other. With this division in mind, consider further the difficulty presented when tasked with transposing a CIM (RAD) into a PIM (UML). It is challenging to imagine a static view of a behavioural model, yet the UML accomplishes this in software design at the PIM level via the Activity and Class Diagrams.

As found in Chapter 4.0, BPM and Software Engineering ideals are somewhat conflicting. In Software Engineering, structures described by Dijkstra (1968) are used, where everything is neatly organised; the real-world is much less coherent and applying such modelling techniques to business processes is inadequate (Ould 2004c). Furthermore, there is no use in modelling human behaviours, collaborations and interactions that occur in the business process but have no significance within software systems. So, not only is it suggested that the CIM be computationally independent (a business process model alone in fact will not do), it also ought to be independent of human processes that are not to be realised in the final system, since inclusion of such items will lead to ambiguities and unnecessarily complex models. It is argued here for the CIM to facilitate modelling of real-world disorder whilst not neglecting notations to which software engineers are accustomed to; the CIM is the perfect candidate for a gateway between the *real* and *software* worlds, and ultimately delivering requirements into the design of software systems.

5.1.1.5 Transformations

As previously noted, the MDA is rooted in transformation automation and focuses on the decoupling of the business logic of applications from the underlying technology that provides them. Experience has shown that most MDA transformations take place between PIM and PSM Class Diagrams (Sheena et al. 2003), the behavioural model is offset from the outset. The Activity Diagram is a prominent behavioural MDA artefact that may (or may not) be used in any MDA transformation. A PIM may be transformed into multiple PSMs, for example, a single PIM object may translate into an SQL Database Table definition, an EJB Entity Bean and a Remote Interface at the PSM level. With each new level, more detail is added to the model in question until eventually final code, or other technical models (such as SQL DDL, IDL Interfaces, Deployment Descriptors etc) are output (McNeile 2003).

Transformations have traditionally been the painstaking hard work of developers to transform source and target models by hand. With the advent of MDA, and associated transformation languages like the QVT, OCL and ATL, the transformation can be automated via transformation rules. A course grained component model

retains association and navigation but abstracts away the details; a fine grained component model is required for coding (Kleppe et al. 2003).

It is important to note that such “transformations may never be fully automated” (Berrisford 2004). Forward engineering is defined by elaboration (adding detail) whereas reverse engineering is defined by abstraction (removing detail). In consideration of CIM transformations, reverse engineering is suggested by the “composition and suppression of detail” (Berrisford 2004) whereas forward transformation is not realistic “from a purely conceptual CIM” (Berrisford 2004) as was demonstrated by the case study investigation included in Section 4.1.2. It was described that for forward engineering to be possible, there must be a familiarity with software notions in defining the CIM, which raised the issue of specification and the system boundary for consideration. Furthermore, the argument of *Translationist* Vs *Elaborationist* posed in Section 2.1.3 is important for further consideration of what might be the expectation in reference to the output of transformations.

5.1.1.6 Translation

The *Translationist* (mainly involved with specialist real-time and embedded systems) produces only a PIM with great emphasis on translation rules by which the output PSM and code are prescribed to adhere to. No changes are made beyond PIM level. According to the OMG, models should exhibit system behaviour that is able to be tested and simulated (McNeile 2003; Soley 2006), which of course is fully realisable given a *Translationist* approach. *Translationists* typically apply state machines and activities, which result in enactable models. By having an enactable PIM, requirements can be ironed out early in development process. The *Translationist* argument is very compelling and is ideally where the state of the MDA is envisioned to be in the future. However, the technology for the provision of a pure approach to *Translationist* MDA for complex business applications remains unavailable (Frankel 2005) and state machines are inadequate at describing concurrent process activity (Gupta 2007b).

5.1.1.7 Elaboration

Elaborationists account for the mainstream, developing business information systems by using MDA to produce skeleton models, then elaborating on those models to produce the user-defined effect required. A good MDA tool supports the synchronisation of models whereby if the elaboration is made at the PSM level; all other levels will be kept in sync (McNeile 2003). The *Elaborationist* application of the MDA is made a reality with tools such as ArcStyler and OptimalJ, enabling developers to update the system specification at various levels of abstraction, whilst keeping all pertinent models in sync and allowing for customisable transformation rules (Uhl and Ambler 2003).

The identification of the *Translationist* and *Elaborationist* themes in MDA development appears to stem from the fact that UML is inefficient at providing the required “precision to enable complete code generation” (Meservy and Fenstermacher 2005). The *Elaborationist* view is the current solution to this, in that experienced programmers can simply take the skeleton template provided by the MDA tool and plug in required extensions. Little evidence is provided of work relating to CIM-to-PIM transformations. A semi automatic approach using OCL transformation rules has been identified but requires further development (Leonardi and Mauco 2004), perhaps using an enactable GUI. Clarification of business models and transformations is an important first step in this process, conceivably using the RAD.

5.1.1.8 RAD to Activity Diagram

A transformation between the RAD and the Activity Diagram is hereby considered to be the most useful direction of transformation, since this would be transforming a purely behavioural business model into a UML software artefact and is supported by an exploratory study which suggested that such a translation is possible “in particular cases, but [relies] on the ability of the translators to establish and maintain... equivalence between the two [notations]” (Odeh et al. 2002). From applying this type of transformation in practice, it was found that when considering such transformations for use within the MDA it would, at best, produce a collection of Activity Diagrams that match the set of separate *threads* contained within a RAD. The Activity Diagram notation is simply not rich enough to cater for such a transformation, which means transformation rules themselves would have to contain further information about what to do with elements that cannot be represented in an Activity Diagram. Stepwise software could be produced which asks the modeller appropriately constructed questions to allow for an Activity Diagram to be constructed directly from a RAD, for example, how an interaction might be represented via control flow, which *threads* are to be included in the same process (if they even can be) and what is to be done with the nodes that have no corresponding Activity Diagram notation.

5.1.1.9 Activity Diagram to RAD

Again, from researching this type of transformation, it is difficult to escape the loss of richness that occurs. A single Activity Diagram results in a simplistic single *thread* RAD. This might be useful but the RAD is not an artefact of software engineering, nor is it that of the MDA. Therefore, there may be little significance in producing a RAD from an Activity Diagram unless perhaps to demonstrate to the business user the understanding of a system construction in progress. Moreover, by creating a single *thread* RAD, the richness lost from not being able to deliver *hanging threads* and other RAD notions leaves reasoning futile. An example relating to a simple order processing system is given in figure 5.1.1.9.1.

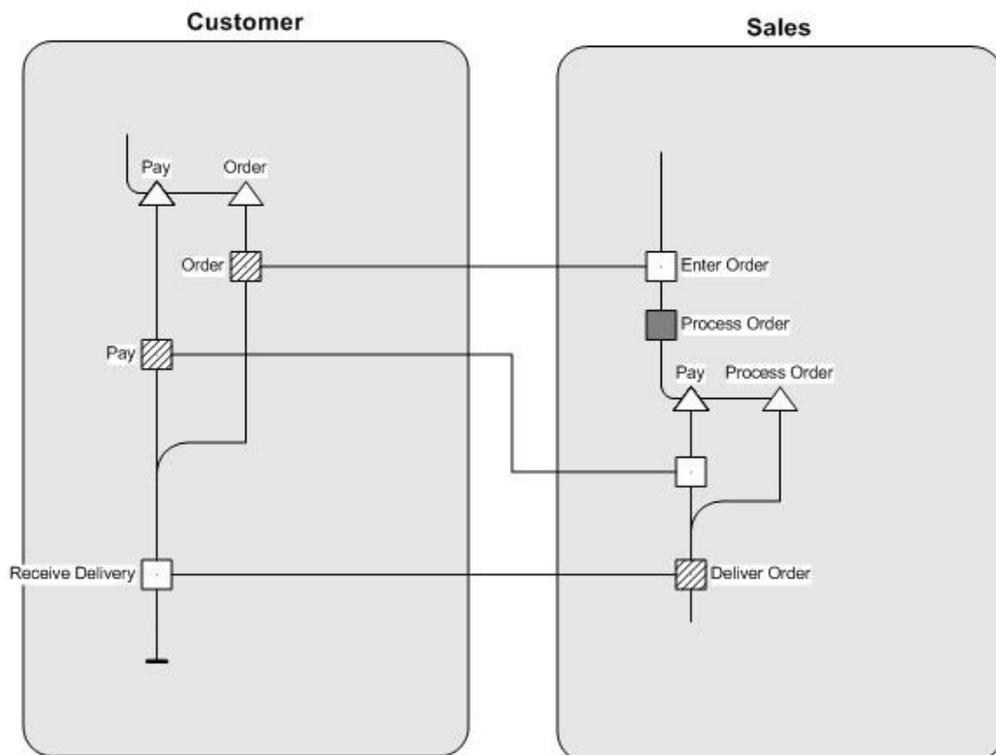
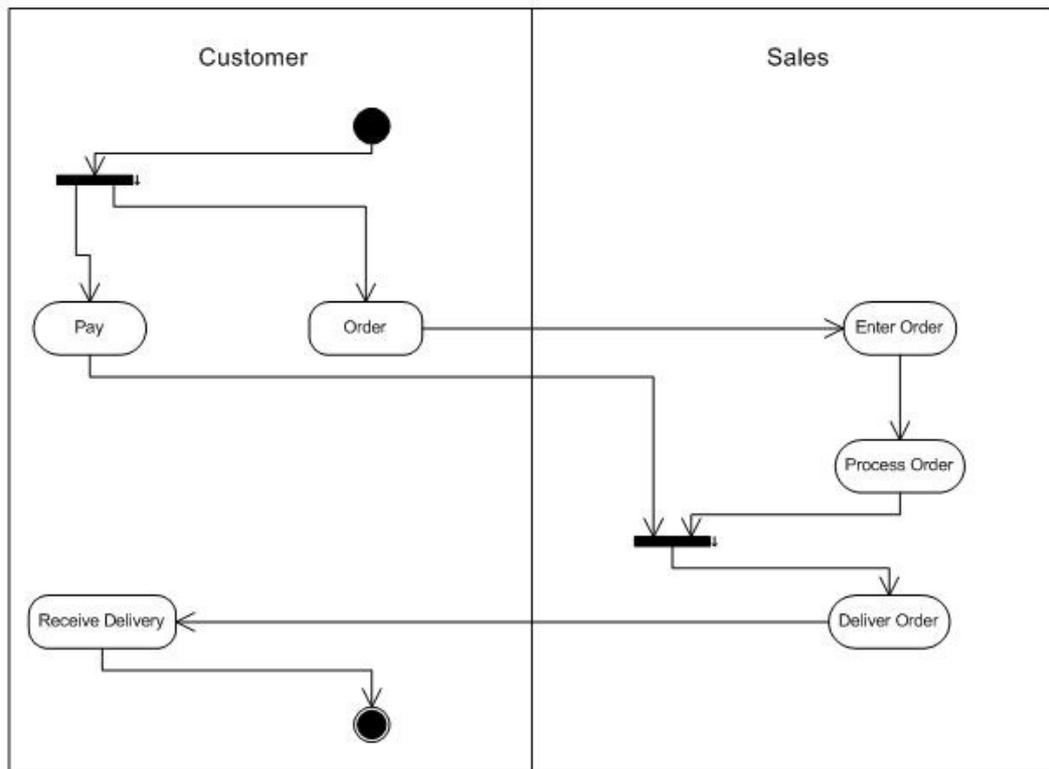


figure 5.1.1.9.1, UML Activity Diagram transformation into RIVA RAD.

This example is demonstrative of just how basic transformations of this type can be. Interactions are revealed by analysing the control flow between activity partitions, with a forced *driving* node being inferred. A start node will never be recognised in a RAD; therefore, every *thread* will remain hanging. Nothing is to be said of the persistence of role instances and nodes that are not included in the Activity Diagram notation, such as the prop, which will never be realised. Moreover, the transformation produces a restrictive view with poor construction, which may not even correctly represent the process to which application the model is prepared.

5.1.1.10 Role Utility Diagram (RUD)

If the elaboration ideal is accepted within the MDA for PIM-to-PSM transformations, then it could be associated with those relating to CIM-to-PIM transformations by creating a static view of a business process model, hereby termed a Role Utility Diagram (RUD), which might in turn transform by elaboration into a UML Class Diagram later in the development process.

The RUD is a conceptual model developed as part of this research and can be created automatically from a RAD by following a set of rules. There are three main components to the RUD, they are the static view of the involved role, the props that are used by that role and the output associations delivered by an activity of that role. Figure 5.1.1.10.1 demonstrates this via the application of a traditional musical jukebox system, focussing on the *customer* role.

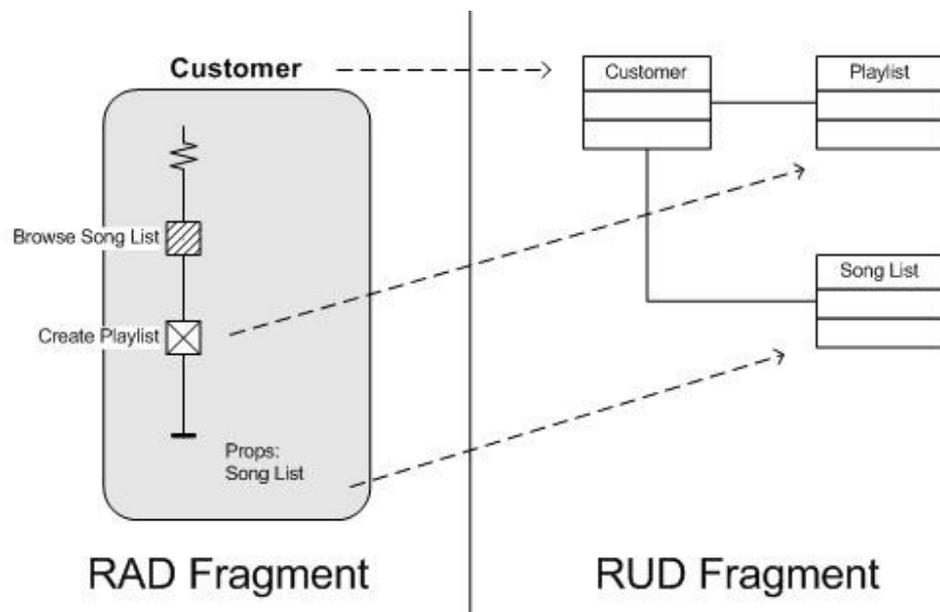


figure 5.1.1.10.1, RAD fragment for jukebox example and associated RUD fragment.

As might be expected, the automated transformation from a *complete* RAD may produce a multiple of RUD fragments. This is because each RAD role is transformed individually. Since this is the case, theory normally used at the PIM-to-PSM abstraction levels can be drawn upon here. That is of model merging. A model merging tool could be used to merge the RUD fragments into a single RUD, based on the elements and associations developed from fragments that were previously input. This is demonstrated in figure 5.1.1.10.2 by taking the RUD fragment created in figure 5.1.1.10.1 and combining it with another fragment, in this example, *payment processing*, following techniques described in Grimm et al. (2007).

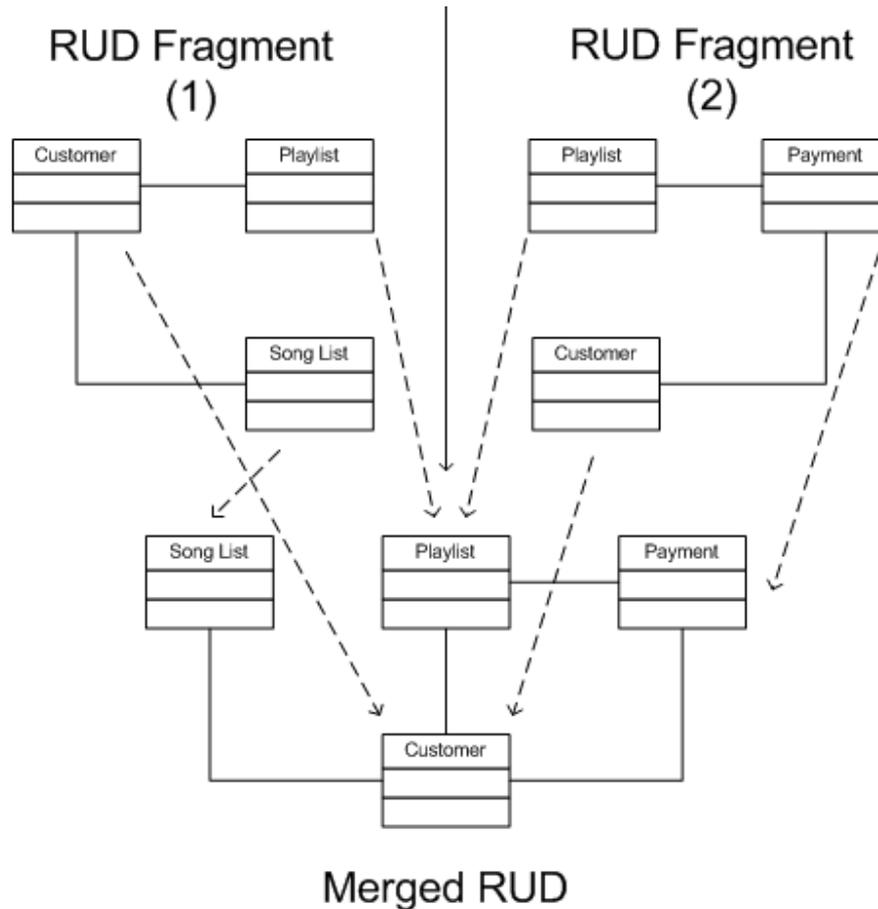


figure 5.1.1.10.2, RUD fragments combined via model merging.

The higher order prevails even in the broken pieces... at least you aren't lulled into a sense of false security by some... merely fabricated order (Huxley 1971).

The delivered RUD is suggested to be the foundation of a Class Diagram, constructed automatically by following transformation rules at the CIM level and complementary to the RAD providing a software representation of business process logic. There is no notion of attribute or operation, nor should there be since they are thought to be a design concern and unimportant at this conceptual level. RUD elements and

associations could be fed into an appropriate tool for modelling the PIM if defined in a universal language such as the XML. This is most helpful in that the eventual Class Diagram (PIM) would be traceable directly to the business model at the CIM level, and any change in the business model (CIM) could result in a change in the Class Diagram (PIM), and vice versa.

5.1.1.11 Conclusion

This investigation has highlighted that the MDA is falling short of expectation and with each downstream transformation comes a loss in upstream richness, and important requirements may become lost in translation. Whilst the UML specification 2.0 has made considerable improvements in introducing a behavioural model for use within Software Engineering via the Activity Diagram, such improvement may not be sufficient in terms of business accessibility and the MDA. Important differences between techniques involved in the MDA and the BPM have been highlighted by this research in looking directly at the rules associated with the Activity Diagram and the RAD, and the feasibility of transformations between the CIM and the PIM business and software technologies. If a move in this direction were to be made within the MDA, engineers could produce a technical PIM derived from a format, for which the business user has a greater appreciation, and vice versa, thereby bridging the gap between technology and business processes.

Rules for the transformations included in the research are not published in academia and further investigation in this area is required. However, the rules represent a basis that might be incorporated in any automatic software development that included the Activity Diagram, Class Diagram and RAD. RAD to UML translations are unavailable, they may not even be possible in consideration of notions such as specification and the system boundary as discussed in Chapter 4.0, since the RAD has no mechanisms to support such notions. Evidence was found in support of the idea that the MDA has yet to be applied to RE concepts; the transformation between RADs and Activity Diagrams were found to be less successful, however, an elementary process for defining a static view of the RAD has been discussed. By presenting a CIM in the RAD and RUD forms, systems could be delivered directly out of the requirements, human-driven processes, and be traceable back to them. Furthermore, the RAD is founded on the *Process Trinity* with business strategy at the heart and therefore such strategies should extend into any IT systems that are produced from them providing for the strategic alignment with IT. The technique ensures that objects are created from originating events that necessitate the need for them. Models that are natural to the PIM might be a transformed output from those implemented at the CIM level via clearly defined metamodels at both levels (Berrisford 2004). The RAD metamodel defined in this chapter could therefore be used as the basis of such a transformation involving the RAD and the MDA.

5.1.2 CIM Support for Requirements (CIM-to-CIM)

Section 5.1.1 focussed on the extent to which the MDA could accommodate CIM-to-PIM transformations from a requirements definition. This chapter extends this investigation by considering CIM-to-CIM transformations and draws on the findings of Chapter 4.0 that the MDA does not explicitly consider requirements and specification as part of the CIM. Current MDA research centres upon the use of the BPMN for the CIM definition. As previously suggested, it is proposed that whilst there are many models and notations available, those that are significantly supported by the OMG, such as BPMN and the UML, may not be best for use by non technical stakeholders. With specific emphasis on the value of BPMN for Business Analysts, this research provides an example of a typical CIM. A requirements approach to specification is then adopted, which proposes to be beneficial to the CIM phase with the goal being to further the utility of the MDA by embedding it with RE theory.

The objective of the standardised BPMN is two-fold. Firstly, to be understandable to the business community within which it is designed to operate, which is provided for in a simplistic *flowchart* manner to which the business user is already accustomed. Secondly, to be transferable to the software community in a format that is rich enough to be defined and executed (i.e. via the BPEL4WS specification). This is challenging and met by the proposal of *mapping* from BPMN notation to BPEL or the like and is how the “BPMN creates a standardised bridge for the gap between the business process design and process implementation” (OMG 2008a). It is important to understand that a simplistic business process diagram does not contain sufficient detail for direct mapping to BPEL4WS and therefore, “graphic elements of BPMN will be supported by attributes that will supply the additional information required to enable a mapping to BPEL4WS” (OMG 2008a). However, the definition of such attributes would in reality be likely to be completed by the technical expert, rather than the Business Analyst, since a complexity is introduced beyond the *Modus Operandi* for business use, as found with the UML in Sections 4.1.2 and 5.1.1.

The case study in this section takes the form of a simplified travel reservation system adapted from Silver (2008d); the BPMN model for this system is shown in figure 5.1.2.1.

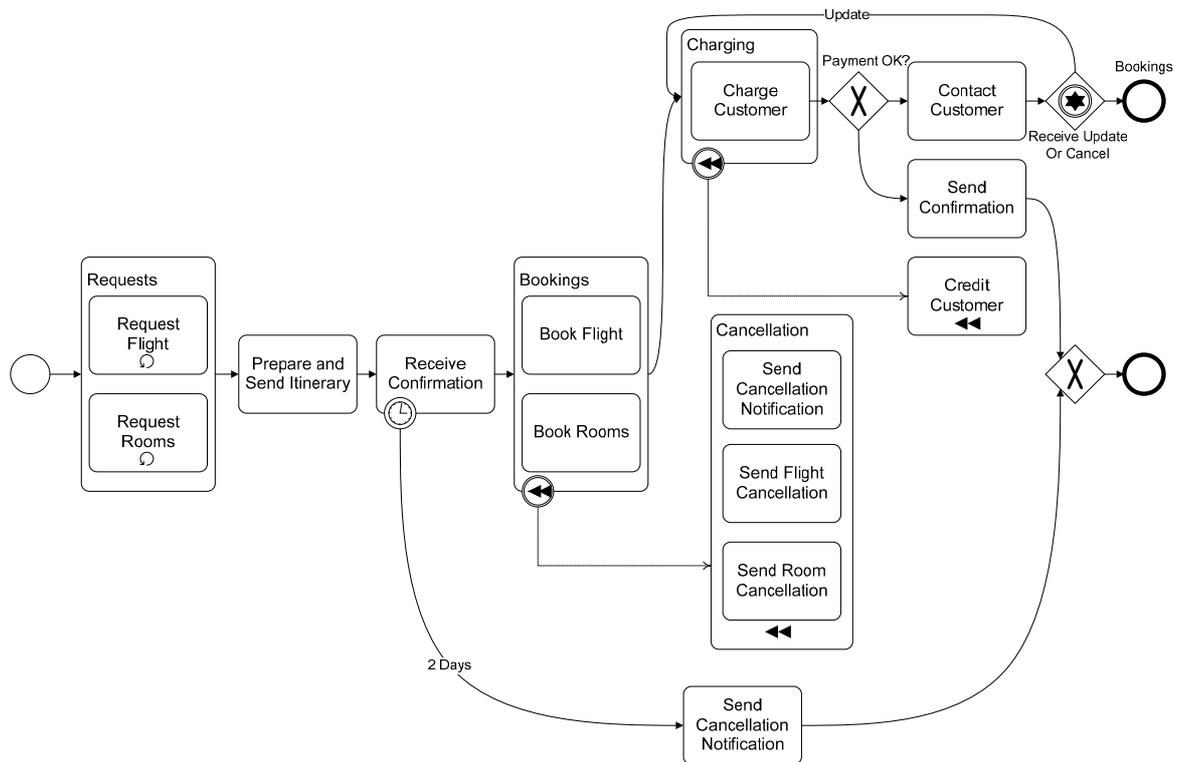


figure 5.1.2.1, CIM of a travel reservation system represented in BPMN (Source: adapted from Silver (2008d)).

In summary, the CIM demonstration involves a fictional scenario and represents a travel reservation system whereby flight and hotel details are input, itineraries created and verified, payments accounted for and holidays are booked with the travel agents. This is a simplistic process (which has been outlined here in a single paragraph of 26 words) yet involves a somewhat convoluted and confusing diagrammatic definition via the BPMN. Moreover, because the components of the BPMN are based upon Software Engineering semantics, the accuracy and correct understanding of the notation is imperative in the definition.

Business Analysts “do not start with textual requirements: it is too complex to come out right away with them. Instead, they usually start with simple graphic diagrams” (Rivkin 2008), supported by VIDE (2009). A common technique for defining specification is the Use Case. Here, the BPMN outlined previously is accounted for within the simplistic technique, with further discussion and analysis given beyond. A set of Use Case specifications are produced where the *users* of the system and are associated with the *tasks* they complete in interacting with the system (Stevens and Pooley 2000). A Use Case diagram is presented in figure 5.1.2.2, along with its matching Use Case description in table 5.1.2.1, which follows the recommended guiding principles prescribed by the CP style rules, which match the 7C’s criteria discussed in Cox et al. (2001), Phalp (2002). CP Style rules offer the user direction and guidance on the format of Use Case descriptions.

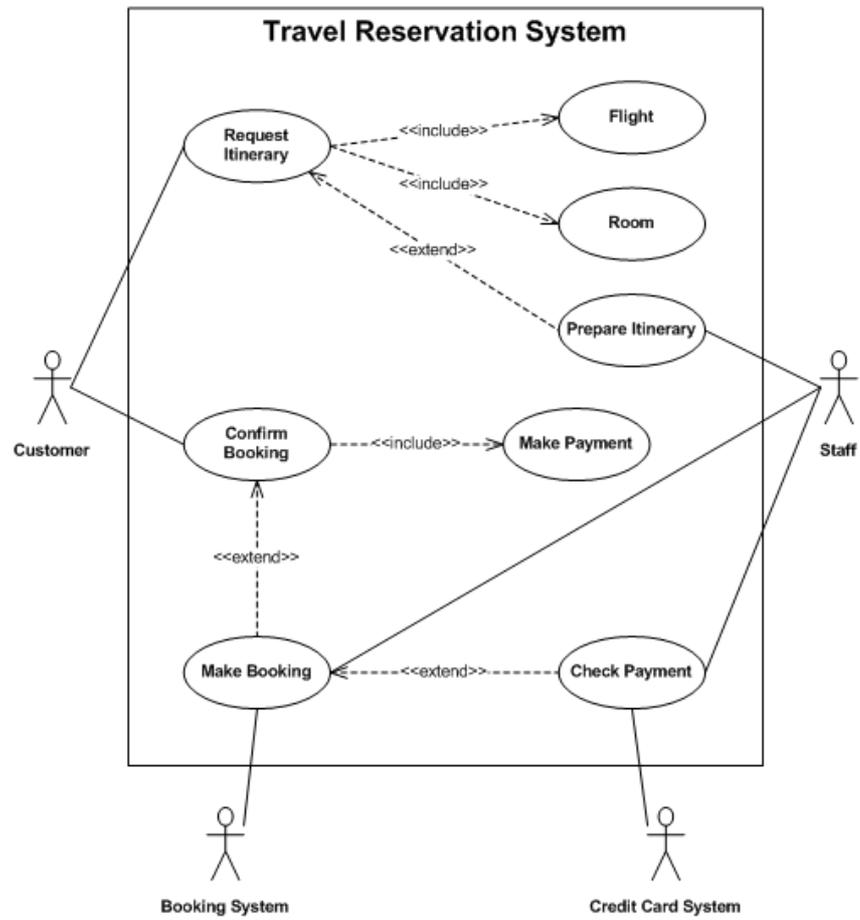


figure 5.1.2.2, travel reservation system represented in a Use Case.

Use Case Title: Travel Reservation System
Actors: Customer, Staff, Credit Card and Booking System
Context / Purpose: To enable the booking of a holiday
Pre-Condition: Staff is available to accept itinerary request
Event Flow:
<ol style="list-style-type: none"> 1. Customer requests Itinerary from Staff 2. <<includes>> Customer selects flight details 3. <<includes>> Customer selects room details 4. Staff prepares Itinerary 5. Staff sends Itinerary to Customer 6. Customer sends Confirmation to Staff 7. <<includes>> Customer makes Payment 8. Staff makes Booking 9. Staff checks payment with Credit Card System (Constraint: Credit Card System is available) 10. Staff sends Confirmation
Alternatives:
<p>Alternatives to this Use Case could be that the itinerary fails to reach the customer or the customer fails to update staff regarding payment problem. However, such an alternative has been ruled out due to insufficient elicitation and will be discussed in the subsequent section.</p> <ol style="list-style-type: none"> 11. If Confirmation fails, Staff sends Cancellation Notice to Customer (Constraint: Wait 2 days) 12. If Payment fails, Staff notifies Customer 13. Customer updates Staff 14. Customer cancels Booking
Exceptions:
<p>Exceptions to this Use Case could be that there is a system-wide failure; the credit card or booking system is unavailable. However, such exceptions have been ruled out due to insufficient elicitation and will be discussed in the subsequent section.</p> <ol style="list-style-type: none"> 15. If Booking fails, Staff sends Cancellation Notice to Customer 16. Staff cancels Flight and Room with Booking System (Constraint: Booking System is available) 17. Staff sends Cancellation Notice to Customer 18. If Charging fails, Staff credits Customer
Post-Condition: Booking confirmation or cancellation notice is sent to the customer

table 5.1.2.1, travel reservation system represented in a Use Case Description.

The BPMN is primarily a software notation. The use of a software-oriented notation for the purpose of modelling business processes is likely to make the notation seem inadequate at representing notions particular to business processes. Furthermore, it is not logical for business to utilise “diagramming techniques that are used by software developers since they are designed to notate and model all manner of things that business managers don’t need to be concerned with in order to manage the business” (Harmon 2005).

Initially, it became evident that there would be some difficulty in distinguishing elements that were important and required by the system, from those that were not. The BPMN model has no system boundary view as provided by the Use Case, and it was unclear as to whether the customer or staff member would enter the initial booking request, which was clarified from examination of the Use Case diagram.

In the BPMN demonstration, notions of role or actor, and interaction do not exist at all, thus do not transfer into the Use Case diagram and description. This highlights the insignificance that BPMN places on such notions, which are of course natural to the business world and requirements methods. The Use Case definition provides a base which was different in comparison to the BPMN model. Although many identified cases remain the same or similar to BPMN tasks, the Use Case was found to be better aligned with the reality of the situation. Roles, interactions and collaborations between parties are easily identifiable, for example, the *check payment* case involving the staff member and the credit card system.

In the MDA, it is considered important that the business user be able to understand, validate and apply the CIM, to help ensure that requirements are correctly met in the final software product. A study in Peixoto et al. (2008) demonstrates how using the BPMN in business can be just as difficult to understand as UML Activity Diagrams (Peixoto et al. 2008). The Use Case is viewed as a naturally simplistic tool, and therein lies the beauty; being useful to communicate to the heart of the business user, rather than introduce the complexities that the BPMN offers and, although simplistic in nature, the Use Case specification is being “increasingly integrated with model elements” (Hansz and Fado 2003) of the MDA. However, as discovered in Section 2.1.4, Use Cases are not central MDA artefacts and are unsuitable for the generation and enactment of code (McNeile 2003) and “the task of moving from Use Cases to design classes is neither obvious nor simple” (Cox and Phalp 2007).

From the Use Case specification, it is clear to see that the travel reservation system employs a great deal of human interactivity. The BPMN solution, however, presents a wholly computerised and sequential view of that interactivity at a particular level of abstraction. Because of its nature, the “BPMN does not match the reality of human behaviour” (Harrison-Broninski 2006a) and therefore is unable to model human-driven processes efficiently.

A mechanism was not available to incorporate non-functional requirements within the context of the BPMN diagram. For example, a requirement might request that the designed system be compatible with the existing system (such as the *credit card* and/or *booking systems*). This is a platform specific consideration that is beyond the scope of the CIM and PIM concern. However, this can easily be accommodated within the supporting documentation of constraints in Use Case descriptions to ensure such non-functional requirements are carried through beyond the CIM.

The eventual system is likely to issue an email address, or perhaps SMS functionality, to cater for the *send* and *receive* requests between the *staff* and *customer*. This requirement is not specified in BPMN; it is only from the analysis of the Use Case description that this issue is raised.

Alternative and exceptional Use Case discovery is part of the Use Case exploration. On review, it becomes evident that the BPMN contains no indication to what might happen should the itinerary fail to reach the customer, the customer fail to update the staff regarding a payment problem, a system-wide failure be experienced or the credit card and/or booking systems be unavailable. Exceptions are delivered in BPMN by relating to an event that might require the cancellation of a booking or an error requiring the customer be credited for amounts charged. Use Cases naturally raise such questions as cause for concern to be verified by the stakeholder involved and provides sufficient elicitation from a RE perspective in the construction of specification.

Techniques such as the BPMN are unsuitable for defining human behaviours and biased to supporting technological implementations (Harrison-Broninski 2005c). Put simply, business and software analysts think they understand BPMN in the same manner (Silver 2008b), but specifications can be understood quite differently by those involved (Cook 2004a; Phalp and Shepperd 1994). From reviewing BPMN samples distributed in training materials, significant errors were found and highlighted in Silver (2008e, 2008f). Semantics related to sequence flow, gateway attributes, intermediate events, sub-process boundary protocols, compensation events, transaction sub-processes, cancel events, link events, state based synchronisation, etc. are all illustrative of the complexities involved with the BPMN. If the tools and teaching materials cannot get the notation right, it is difficult to see how business users may make anything more of BPMN than simple flowcharting. It is agreeable that "there isn't much educational material out there that shows people how to use BPMN correctly" (Silver 2008e). However, once plugged into the MDA, CIM level errors could be a critical project management concern.

The study demonstrated how complexities relating to the BPMN may not be beneficial to the Business Analyst in representing the requirements for software systems, especially when looking to illustrate the CIM within MDA, because the BPMN provides an inadequate representation of human behaviour. It is highlighted that the UML and MDA code generators are not the remedy for the ills of the development world (Thomas

2004); tools and notations available to the Business Analyst, as well as the standards that govern those technologies, must be embraced, not only by the OMG and the Software Engineering world, but perhaps more importantly by the business users.

5.2 Implications

The findings presented thus far support further trials using alternate modelling techniques, including the RAD. Research into how different a PIM might be defined may hold some value in the future of specification via the CIM. It is suspected that mechanistic and humanistic processes might both require specification for BPM, but only those that can be mechanised for the MDA. Either way, an understanding of the nature of business processes is central to the specification, since humanistic processes presented incorrectly at the CIM level may result in a deformed architecture which does not facilitate support of the business process it was defined to augment. The remainder of this chapter considers the implications of these findings in terms of MDA notation and tooling.

5.2.1 Notations

The BPMN was designed to make the implementation of executable models easier, driven by one notation, rather than be flexible to the multitude of notations. It is said that the BPMN provides “the power to depict complex business processes and map to Business Process Management execution languages” (OMG 2008a), however, limiting the modelling of the business domain to a single notation provides a weakness in that concepts which could have been adopted in other notations may no longer be used. Moreover, by definition, if the BPMN is based on software concepts, the technique is no longer as meaningful to the Business Analyst as it is to the software producer, leading the Business Analyst to learn to think, for example, in terms of object rather than process.

The solution suggested here is to open out the MDA and allow the accessibility of a greater range of modelling notations and standards support, rather than giving focus to just the chosen few. The BPMN and the UML are not intended to be supportive or facilitate domain specific modelling definitions in terms of the Business Analyst (Brahe and Bordbar 2006). A greater acceptance of other notations and standards support is suggested to enable the MDA to fulfil the ideology of facilitating truly interoperable, portable and reusable models. For example, the BPMN only allows messages to interact with two single entities, whereas with human-driven processes, interaction is usually between many (a conference call, for instance), which is allowed using other diagrammatic notations such as the RAD. As previously mentioned, the BPMN is not capable of producing a *role*, and therefore, requirements pertaining to such a notational element cannot be modelled correctly for that reason and systems can be delivered incorrectly. In the BPMN, roles have been

aligned to a grouping of activities or process chains via *swim lanes* and *pools*. Roles essentially account for the responsibility and purpose that characterise the people and systems that they represent, and are not just a collection of mechanised actions and functions; the RAD has a simple foundation, with a “natural” graphical notation, which allows them to be introduced to the business user with minimal training (Harrison-Broninski 2005a).

From examining transformation techniques in the MDA, specifically those relating to CIM-to-CIM and CIM-to-PIM, a RAD could be transformed into the UML for the use in software systems development via the MDA. Harrison-Broninski outlines formally how each component in a RAD can be represented in the UML, although with less *specialised usability* and concepts pertaining to the RAD and HIM tools (Harrison-Broninski 2005b). Other researchers also focus on such transformations. For example, a proposed methodology for transforming models of the BPMN into Use Case notation goes some way to looking beyond the OMG’s CIM and towards the PIM (Rodriguez et al. 2007a). Secure Business Processes are defined by using an extension of the BPMN Business Process Diagram and the Business Process Security Profile. QVT transformation rules are then used to capture the BPMN and transform it into elements within UML Use Cases (Rodriguez et al. 2007b). For example, *pool to actor*, *activity to use case* and *security requirement to use case* (Rodriguez et al. 2007a).

Ultimately, the task of retaining the richness contained within requirements models that may be lost in a transformation process remains a difficult one unless the requirements model is held integral to the MDA. The BPMN does not include the technology suitable to retain the richness of information that is provided for by requirements models. An alternate suggestion for investigation could be to include, or extend the notations of the BPMN to account for notions that carry real meaning in requirements models. For example, as previously discussed, the *swim lane* notation could be adapted to reflect that of a *role*. However, this notion is dismissed in other research, where the view is that such richness simply cannot be captured by the BPMN via extension (Harrison-Broninski 2006c), supported by Bushell (2005).

5.2.2 Tools

MDA tools and processes are required to be agile enough to adapt and facilitate changes. Models allow ideas to be “shared in abstractions” (Soley 2006). In order for the MDA to be successful within industry, tools should be accessible to both the software developers that create implementations and the business users whose requirements necessitate them. Many tools are available in the market and purport to pursue MDA ideals. However, it has been seen that “most MDA tools are geared to programmers and software developers rather than non-technical stakeholders” (Kanyaru et al. 2008b) and that such tools remain in an evolutionary state (Leonardi and Mauco 2004).

As previously highlighted in this chapter, the BPMN has associated problems. Difficulties are also apparent in industry in that the BPM field has not yet matured (See Section 4.1.1). Tools are available, Common-Off-The-Shelf (COTS) and custom created, to accomplish all manner of process analysis, design and execution tasks. It is noted that business users are not embracing the BPMN to the degree that the OMG and MDA proponents would hope for. One reason suggested for this is that “the tool vendors themselves don’t follow the spec” (Silver 2008c). It has been highlighted that well known distributors of BPMN software tools and training facilities (analysis derived from a review that includes Savvion, Tibco, Appian and Intalio’s Open Source Modeller) neglect the BPMN specification (Silver 2008c, 2008e). Supporters of BPMN therefore must work tirelessly to ensure that tools and training integration incorporate the correct definition of the BPMN standard as prescribed by the OMG (2008a). It is also noted that “modelling tools obviously don’t include validation routines that weed out illegal diagrams” (Silver 2008c), which may also benefit consideration. Formal semantics could be defined and, via automation, rules could be generated and model semantics verified which would “ensure precise specification and... assist developers in moving towards correct implementation of business processes” (Wong and Gibbons 2008). However, the technique, supported by Microsoft research, draws the definition of semantics in CSP via an abstracted syntax described by the mathematical state base in the notation Z . The mathematical base and process algebra syntax is very complex and unhelpful for the business user in the definition and verification of requirements (Wong and Gibbons 2008). As previously noted in Section 2.2.4, Abeysinghe and Phalp (1997) demonstrate how CSP can be combined with the RAD to provide end-user accessibility, “whilst retaining the formality of CSP” (Abeysinghe and Phalp 1997).

Eclipse is an open source Integrated Development Environment that supports the MDA. Central to Eclipse is the Eclipse Modelling Framework, which is a “model-driven metadata management framework” (Frankel 2005). In marketing Eclipse to industry, the unique selling point is highlighted to be the modelling and code generation capabilities, little focus is given to the consistency and automation of the Integrated Development Environment; which is the real advantage (Frankel 2005). Perhaps this is because the Integrated Development Environment capabilities are too complex and difficult to market for business use. Indeed, by giving attention to modelling and code generation, interest can be gained by organisational managers and investments in technologies can be made. However, if the market is purely represented by technologists, then this should not be the case. In reality, Eclipse does not have the necessary tools to generate applications such as an order processing system, “it is better suited... for modelling a tool’s metadata and generating code that manages the metadata” (Frankel 2005). The Eclipse marketing strategy therefore, could be considered to be aimed at increasing the business interest, without any real promise to those users, delivering directly to the technologists working behind the scenery.

The focus is for MDA vendors to “provide integration with requirements and testing artefacts within the tool... if models can be tested against accurate specifications, then problems are caught upstream where they

can be handled” (Hansz and Fado 2003). Requirements and testing experts “often stay outside the modelling tools. They need to be brought in” (Hansz and Fado 2003). From the perspective of the MDA, they are currently outside of the architecture as a whole, beyond the tooling task. The Business Process Developing Life Cycle (BPDLC) is defined as an approach to eliminate the Business/Systems Analyst knowledge gap, by formalising business requirements in terms of a Basic Business Process Flow (BBPF) (Rivkin 2008). Another solution is the Topological Functioning Modelling for Model Driven Architecture, via UML Use Cases and Conceptual Class Diagrams (Osis et al. 2007). The underlying architecture is that “functionality determines the structure of the planned system” (Osis et al. 2007), however, to limit the definition in such a way perhaps neglects some important pre-CIM and design considerations that should be made in the derivation of a suitable concept tool. Another drive to include requirements within the CIM is presented in the VIDE initiative (VIDE 2009) and adopts an interactive environment, based on BPM ideals (Phalp and Jeary 2010) that eases the business user into the MDA process, without getting overly involved in the technicalities related to the architecture. The tool provides a “development environment” (VIDE 2007) consisting of several palettes, guiding the user from pre-CIM-to-PIM via a stepwise methodology (VIDE 2008a) and represents a useful step in the right direction of making the MDA accessible to the business user, despite having a reliance on the BPMN for defining the CIM.

5.3 Summary

The relationships and differences between business and software techniques, and a discussion regarding the notational and tooling implications involved, gave valuable insight into how requirements techniques may be applied to the MDA, and what needs to be done to achieve that aim. The suggestion for extending the MDA has previously been identified with the solution being that any such extension support specification without any loss in the versatility of the MDA, providing a conduit to interoperable, portable and reusable software models. By including RAD definitions in support, the MDA could be enhanced and business user interface facilitated. A foundation Class Diagram could be derived based on the original process specification or RUD as described in Section 5.1.1.10, provided system boundary elements are addressed. A loss of richness was clearly identified when attempting to transform from the business model and therefore further research is required in discovering how to retain the richness provided by business models, perhaps by including the RAD as a support model at the PIM level or through a series of model evolutions. By examining the transformation process between CIM level artefacts, it was found that the BPMN is semantically rich in software knowledge and not an appropriate notation to communicate ideas to the business user or to produce specifications from as the notation was found to be void of any system boundary concept. A simple Use Case could cater for such a system boundary view and be better aligned in demonstrating user requirements and resolving the ambiguities and complexities associated with the BPMN in terms of the business user. However, the Use Case is not a central artefact of the MDA, nor is it complex enough to retain the richness required of

the PIM. Promising results were found in Section 5.1.1 in terms of transforming from the RAD to the UML. Provided the concern for specification is addressed, the RAD may prove useful at interfacing between the business and software users as a single CIM level notation, being both rich enough for business process definition and simple enough to facilitate understanding of that process. Therefore, further research is required to discover how business notations, such as the RAD, might be supported within the MDA; moreover, how the framework of the MDA could be extended to support varied and combined notational standards at the CIM level, whilst retaining the central ideals of the MDA.

Chapter 6

Extended MDA

In this section, ideas based upon findings made in Chapters 4.0 and 5.0 for extending the MDA framework to make the connection between business and software users are discussed, with a solution in satisfaction of aim 3 being provided in Sections 6.3 and 6.4. The consideration is focussed upon what is useful for and applicable to both business and software users alike (Musschoot 2010).

6.1 Extending the MDA

The CIM is representative of the *enterprise* and includes the definition of business rules, facts and terms (Hendryx et al. 2002). However, due to the irregularity of the real-world described through Chapters 2 to 5, it has been seen that MDA transformations are focussed only on those involving the PIM and PSM. The MDA does not place enough emphasis on CIM development (Ambler 2007; Kabanda and Adigun 2006; Karow and Gehlert 2006; Phalp et al. 2007) and the exclusion of the CIM within the MDA transformation process is demonstrated in figure 6.1.1. It is seen that no strong connection exists between the CIM and the PIM and there is little research given to transformations involving the CIM, that is CIM-to-CIM and CIM-to-PIM (Kherraf et al. 2008), supported by the findings of Chapter 5.0.

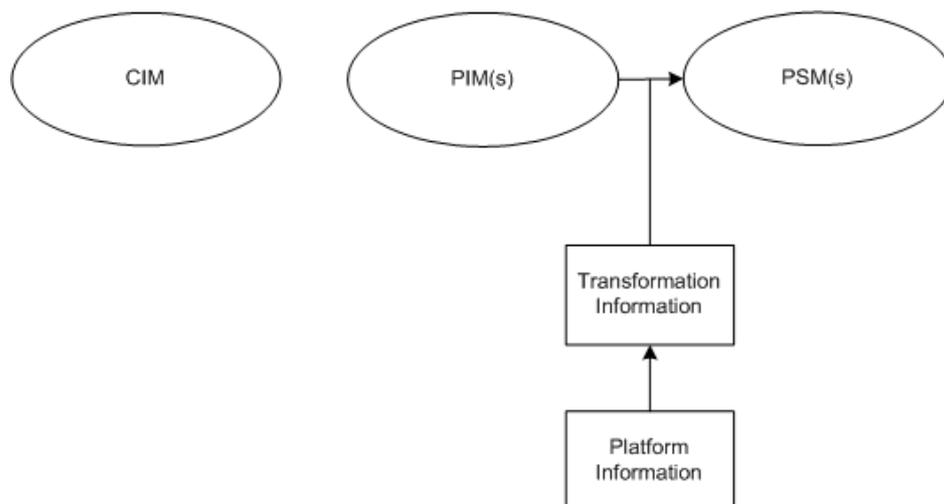


figure 6.1.1, MDA Viewpoints (Source: developed from OMG (2003b)).

To align the paradigms of business and software within the MDA, it is appropriate to use RE and BPM technologies centrally within the framework. The UML appears to be the *de facto* standard for the PIM and support is given by the MDA to the adoption of the BPMN specification (OMG 2005, 2008a; White 2004) for CIM definition. The UML and BPMN are plagued with software engineering concepts and it is difficult to replicate those of RE using them. Notions natural to the business domain, such as *role* and *interaction*, are not easily represented using UML or BPMN, and that is key. Arguably, the UML may not be the best generic technique for mapping concepts to programming languages, let alone from a business domain into those of the programming world (Génova et al. 2005). From research conducted thus far, it is thought that extending the MDA definition to account for the dislocated CIM could involve a combination of two suggestions in accounting for specification within the CIM. The first being that particular models and modelling techniques are specified for use in making the connection, the other is that a more abstract framework extension be defined for the MDA, within which any number of techniques might be employed, so long as essential themes are addressed. Each of these ideas is given further deliberation in turn.

Section 5.2 highlighted that the nature of business processes is of important consideration in determining techniques sufficient for requirements modelling purposes. It is also noted that the business process does not necessarily account for the specification of a software system. For the definition of functional requirements at the CIM level, the BPMN may be perfectly sufficient to model processes that have a mechanistic nature, provided the notation is understood and implemented correctly. Other processes have a complicated nature and may be better represented by the RAD, perhaps with the BPMN definitions in support, or to enable transformations from RAD to BPMN or the UML. Support could be given in defining a less complex version of the BPMN, which could then be translated into the BPMN for the definition of a mechanistic software system, avoiding identified complexities. The simplistic notation used could be akin to flowcharting methods that the business community is used to, but the important thing is for elements within the notation to be directly transferable. For non-functional requirements, visual test cases might be suggested to be the key MDA artefact.

The second suggestion is to conceptualise an accessibility extension to the MDA framework to include requirements elicitation and specification as central to the CIM (see figure 6.1.1). Preliminary investigations into ideas of extending the architecture via pre-CIM activities (DDM, IRM and BM) and the formalisation of the CIM in terms of available transformations (see Chapters 4.0 and 5.0) have been conducted by dissecting the expectation of each MDA phase in terms of inputs and outputs. The description of an extended MDA framework would likely follow discoveries made and defined by xMDA in Section 4.2, highlighting the importance of defining the system in terms of functions and constraints, without describing a technique or notation to do so. The idea retains the versatility of the MDA in being interoperable, portable and reusable since the inclusion of any notation constrains these central concepts.

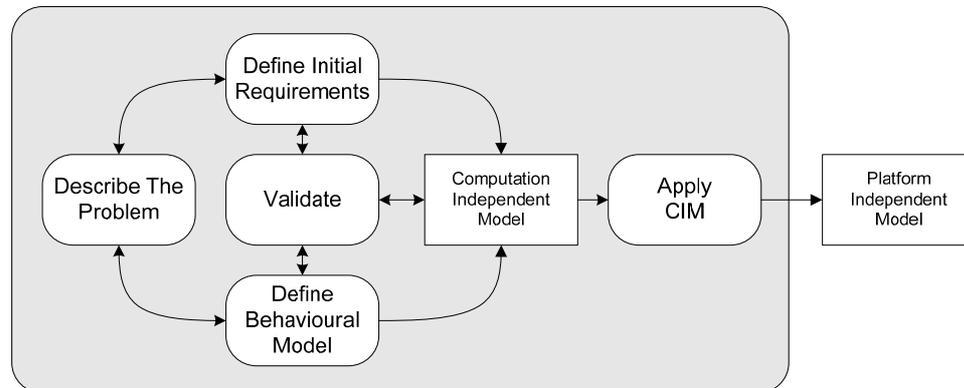


figure 6.1.1, requirements and specification included as part of the CIM definition.

RE and BPM focus on business processes, and associated roles and notations offer freedom to the business user, often allowing an extensibility which cannot (or at least is very difficult to) be interpreted by software modelling tools. One way to support the move to design from a process model is to provide a requirements phase where modellers can specify a system that addresses a given business need in a language common to the business user, with resulting systems being akin to customer expectations on quality requirements and the business process. CIM definitions do not currently account for the unfamiliarity of software development paradigms to the business user, and mechanisms for transferring requirements knowledge are unavailable.

So far, the importance of extending the MDA to include RE has been argued. The MDA is broad enough to accommodate many methodologies from the common waterfall approaches to more recent methods such as *Agile Software Development* and *Extreme Programming* (Kleppe et al. 2003; OMG 2003b). With an increasing demand for systems to match the requirements prescribed by business managers at the start of a project, there is an even greater need for projects to reflect multiple changes throughout the development process. Perhaps the most defining characteristic of the MDA is conceptual simplicity. The MDA fosters an agile environment which goes some way to addressing the concern that development projects are not reactive enough to the changing environments. Developers have been using models for years, business users even longer. The UML is now the operating standard to which MDA is associated, facilitating transformations between models with the ability to be flexible to changing requirements.

6.2 Importance of Specification

The ability to efficiently design appropriate computer systems and enable them to evolve over their lifetime depends on the extent to which... knowledge can be captured (Greenspan et al. 1982).

In 2004, the Standish Group reported that “about 20% of IT projects are cancelled before completion and less than a third are finished on time and within budget with expected functionality” (Kappelman et al. 2006). Among the factors relating to the requirements definition, the differing ideals of software developers and business consumers and the inability for projects to be reactive to requirements change, have been instrumental in a multitude of failed projects (Al-Neimat 2005; Kappelman et al. 2006; Lavagno and Mueller 2006; May 1998; Phalp et al. 2007; Poernomo et al. 2008; STSC 2003).

A requirement is a desired relationship among *phenomena* of the *environment* of a system, to be brought about by the hardware/software *machine* that will be constructed and installed in the environment. A *specification* describes machine behaviour sufficient to achieve the requirement... Specifications are derived from requirements by reasoning about the environment... (Jackson and Zave 1995).

Note: The use of the term *machine* is preferable over that of *system* due to associated ambiguities.

It follows in Jackson (1995) that, the analysis of the PD is related to the design of the solution system via the specification (see figure 6.2.1) – literally the intersection of the problem and solution domains (Jackson 2000).

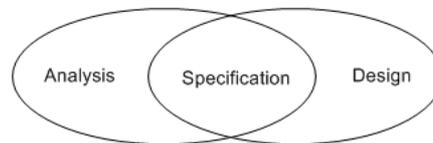


figure 6.2.1, systems of prime concern and development activities (Source: developed from Jackson (1995)).

In specification based software development, user needs are defined by requirements, from which a specification of system behaviour is created to address (Jackson and Zave 1995). PD elements that are not to be realised in the solution system are extracted during specification, imposing neither informality on the solution system nor formality on the description of the PD (Jackson 2000). Once authorised, the specification is transformed into the resulting code for a system to achieve the original user need (Ince et al. 1993); hence, RE is important to the development process (Castro et al. 2002). Ultimately, specification should attain the qualities of being unambiguous, consistent, complete and incremental (Gupta 2007c). In reality, experience has shown that business processes and requirements are often misunderstood, or even entirely neglected, leaving resulting systems incomplete in meeting stakeholder requirements (Bray 2002; Kappelman et al. 2006; Lavagno and Mueller 2006; May 1998). As described in Section 4.1.1, this is frequently because developers require business stakeholders to understand requirements and process logic in technical terminology, rather than logic native to business. Insufficient attention has been given to defining

requirements in complex projects, those which address unfamiliar problems, and the penalties in terms of cost, quality and the time it takes for projects to be completed have all been well demonstrated; issues pertaining to RE have been found to be instrumental in such failures (Bray 2002; Hansz and Fado 2003). It holds therefore, that time is well spent in defining requirements in the development process (Brooks 1975; Greenspan et al. 1994; Kleppe et al. 2003; Sommerville 2004; STSC 2003; Wiegers 2000).

Specification is defined by the boundary between system and user (known as the system boundary). The system boundary distinguishes those functions required by the system from those that are not; and is important in the determination of requirements (as demonstrated in Chapters 4.0 and 5.0) because it defines where the software element is situated in comparison to the overall system (Nuseibeh and Easterbrook 2000). The abstraction is a fine line between interactive human and software elements and, when working with a PIM level language such as the UML, the imposition of object orientation compounds this elemental distinction. The system boundary is a consideration when addressing business processes in that the notations used must deliver the requirements of software systems, and not just process. System views of business processes must therefore be accounted for by the MDA in order that requirements are defined correctly, so that PD elements are not realised in the design of software systems. In consideration of specification in the context of the MDA, a framework extension is described in Section 6.3, supporting a method facilitating specification outlined in Section 6.4. "A design's form, behaviour and function typically progress iteratively from concept to detailed realisation" (Ohata and Butts 2005) and therefore, in order that models are refined and requirements and PD elements accounted for, the framework and method support an iterative nature (see Section 6.4.5 for further discussion).

6.3 xMDA Framework

As noted in Section 6.1, upstream transformations remain relatively unsuccessful and there is no extensive study of this. Those that attempt the task report a loss of richness in process models. Other researchers look to incorporate requirements within the MDA. In Poernomo et al. (2008), a methodology is suggested that can react as requirements change, "accurately reflect them to code and ensure that the successful completion of the IT system will add value to the business" (Poernomo et al. 2008). However, no verifiable data is provided and the solution seems conceptually too complex for business to adopt. The CIM-to-PIM transformation process is inadequately described and the resultant diagram is not necessarily a PIM, since no account for design has been made. In Martin and Loos (2008), an integration language based upon the BPMN is discussed to provide both the simplicity to the business user and enough complexity to the software engineer in order that programs might be created "automatically" (Martin and Loos 2008). This is difficult because by introducing automatic transformations from the CIM to code constrains the CIM and renders the PIM redundant. Design features are shifted into CIM construction, leaving a large part of the development process

down to non-technical business users and the BPMN is complex; it may not be the best notation for business users to demonstrate business processes, let alone design and implement software solutions. In Kherraf et al. (2008), a typical transformation process is identified and demonstrated via a case study. Although significant in furthering the understanding of the value and feasibility of such transformations, no consideration is given to the fact that analysis and design models are from two opposing environments (Kanyaru et al. 2008a). The difficulties encountered in transforming elements from the business to software domain are also reflected in the findings of Génova et al. (2005), Nuseibeh and Easterbrook (2000), which supports the argument here.

Therefore, a solution is suggested which incorporates the two ideas presented in Section 6.1, extending the MDA framework with requirements and specification, and supporting the extension with a technological method, to solve two overarching problems. Firstly, the need to provide a generic software process structure into which different uses of requirements and specification can be accommodated, and secondly, the need to provide a specific set of techniques for expressing requirements and specification within the MDA. Figure 6.3.1 extends figure 6.1.1, illustrating how the MDA could appropriately situate RE within the MDA and connect the CIM to the PIM, ensuring the distinction between the problem and solution domains.

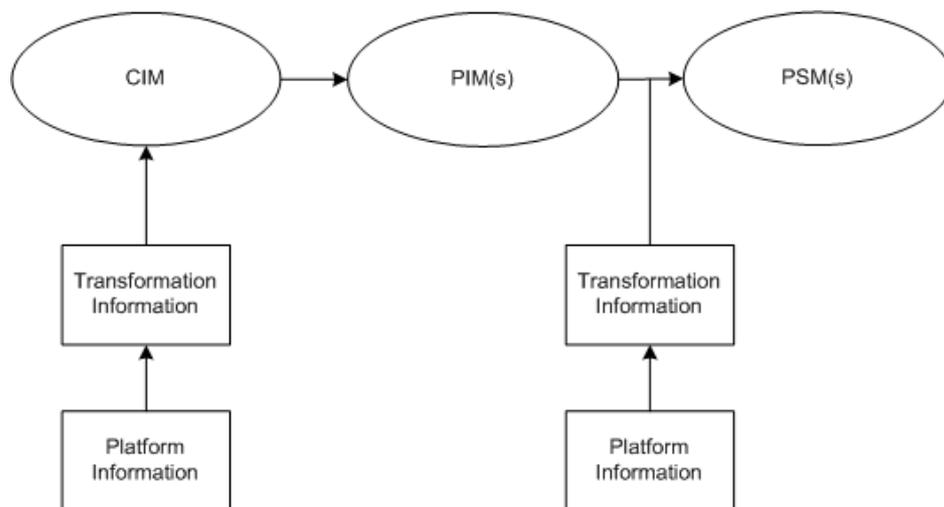


figure 6.3.1, *xMDA framework (Source: developed from OMG (2003b)).*

In software development, “attention is focussed at the application level, where the emphasis is on achieving desired functionality” (Beeson et al. 2002). The inclusion of a CIM that focuses only on software languages such as the UML or BPMN is unhelpful, and leads to the neglect of requirements and specification, giving focus only to such functionality. The system boundary is currently not an explicit consideration of the CIM, which is accounted for in the xMDA framework. It is argued here that, for the transformation of the CIM to be useful post-CIM, and to adequately make the connection to the software domain, it must represent specification as described in Section 6.2, and not a mere abstraction on the PD. However, analysis and design are not considered to be isolated from specification, rather, that a part of both reside within it, supported by

Gunter et al. (2000). That is, specification requires a degree of analysis as input and that specification defines a degree of design as output; therefore the xMDA framework integrates Jackson's systems of prime concern directly into the xMDA with the inclusion of three phases within the CIM - *Environment*, *Shared* and *Machine* (see figure 6.3.2).

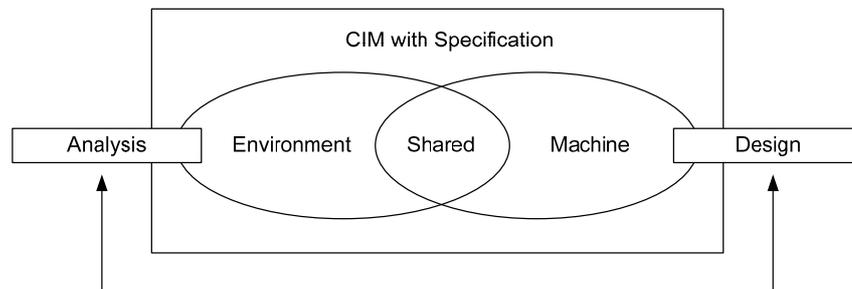


figure 6.3.2, the integration of the xMDA with Jackson's systems of prime concern (Source: developed from Gunter et al. (2000), Jackson (1995)).

The *Environment*, *Shared* and *Machine* phases are proposed to enable only elements relevant to the machine to be transferred into design from the analysis of the PD. This is achieved by applying information about the transformation (i.e. mapping rules) that are informed by the target platform metamodel (i.e. the UML) and extracting elements in the process. This represents a true interface by taking elements in the real-world that are to be realised in the software domain and transferring them into it, allowing for traceability to be accounted for in the documentation trail.

Ultimately, the goal is to facilitate MDA accessibility by allowing business users to define requirements in *any* format to which they can understand and apply; which retains the versatility of the MDA in being an interoperable, portable and reusable solution. This then must be matched to an xMDA representation that is sufficient for the derivation of software systems requirements in a manner to which the Software Engineer is akin to and also retains the richness of definition provided by the business user. The CIM must embrace ideals from both the world of BPM and Software Engineering, and these ideals are somewhat conflicting as noted in Section 5.1.1.4. The more complex upstream becomes, the more chance there is of errors being transferred into implementation, and therefore simplicity is a highlighted ideal in defining the framework.

6.4 xMDA Application

To augment the applicability of the framework described in Section 6.3 and contribute in part to achieving aim 4, a specific method, suitable for expressing requirements and specification within the generic xMDA framework, is required. In order that such a method is defined, suitable notations must be identified. There is

real benefit in “adopting a single notation instead of battling with many competing ones” (Spinellis 2010). However, no single notation was found to address the *Environment*, *Shared* and *Machine* concerns of the xMDA. Therefore, one would need to be created specifically.

With the advent of suitable environments the formal specification will be retained throughout the software lifecycle as the key reference point for design, implementation and maintenance... One approach to developing a requirements elicitation and formalisation method would be to adopt or adapt an existing informal requirements analysis method (Finkelstein 1987).

Section 5.1 demonstrated the potential of the RAD as a candidate for use within the MDA in facilitating the move from business to software models. The RAD is used primarily for “requirements capture and validation” (Phalp et al. 1998). It is grounded firmly in business, rather than software concepts, and given that “software development is an integrated collaboration between machine and people” (Ould and Roberts 1987), a compelling case is made for the notation as a modelling language that is capable of describing human/machine interactivity. It would be inefficient to focus only on designing systems based on structured processes such as those described by the BPMN when the human interactivity the system is designed to support is actually rather more complex. It is said to be “impractical” (Hogg 2009) to model all exceptions during design as this will lead to a complex design set that cannot account for the exceptions it endeavoured to account for in the first place. However, whilst the software domain remains in structured applications, unstructured processes will never be fully realised by software support systems. Humans are structured with enough information to make many complex decisions. Therefore, it is argued here that by utilising the RAD, focus can be laid upon roles that include ad hoc activity and can transfer into enabling operations in design, provided the notation is adapted to account for specification in terms of *Environment*, *Shared* and *Machine* phenomena. The xMDA method outlined in this chapter utilises these three phases to describe how an analysis RAD can be transformed from analysis into design, maintaining traceability and thus supporting the alignment of business models in the MDA.

6.4.1 Moving from Analysis into Specification

Analysis is the software development phase in which the PD (business environment) is investigated in order to provide statements of intent (requirements) to inform a subsequent specification of a software system solution to be applied within that PD. At this point, it may be helpful to be reminded of the systems of prime concern and development activities as given previously in figure 6.2.1 and below in figure 6.4.1.1.

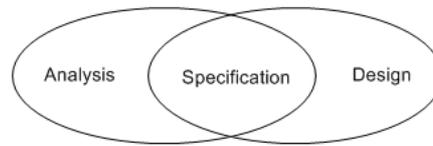


figure 6.4.1.1, systems of prime concern and development activities (Source: developed from Jackson (1995)).

By capturing and exploiting relevant properties of the problem domain, the developer refines customer's requirement into a programmable specification... most specifications are extremely obscure unless accompanied by the refinement history and a statement of the original problem (Jackson 2000).

The first phase of the xMDA method (which is called the *Environment RAD*) thereby ensures that adequate attention is placed on connecting specification with the analysis of the PD (see figure 6.4.1.3).

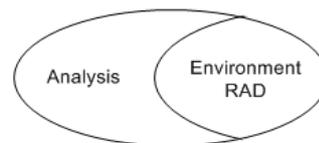


figure 6.4.1.3, Environment RAD of Specification.

Although it is possible to use any business process model to describe the *Environment RAD*, it is not simply the case. The *Environment RAD* is informed by the PD analysis, initial requirements and behavioural definitions (DDM, IRM and BM) described in Section 4.2 and referenced in figure 6.1.1. Therefore, subsequent models are aligned with requirements defined within the *Environment RAD* as a result of analysis.

6.4.2 Accounting for Specification

Difficulties often occur in transferring environment objects into specification and design. "The most common error occurs when the analysis model is supposed to represent the real-world (domain model), but is then used as a specification of the system to be constructed (software model)" (Génova et al. 2005), supported by Nuseibeh and Easterbrook (2000). By defining systems in such a way, design decisions unwittingly become imposed on upstream models, whereas such decisions ought to be addressed manually by the software analyst. Specification-based software development is plagued by the problem that the connection between the requirements and specification is not explicit, and it is suggested that specifications might be derived directly from analysis (Johnson 1988). Therefore, the second phase of the xMDA method (known as the *Shared RAD*)

is focussed on the interface between the environment and the machine, taking *the Environment RAD* as input (see figure 6.4.2.1).

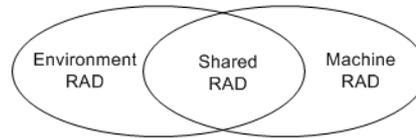


figure 6.4.2.1, *Shared RAD of Specification.*

RADs represent business *activities* (actions or interactions) across *roles*, and explicitly considers the representation of the “resources the role needs” (Ould 2004c) via *props*. Harrison-Broninski (Harrison-Broninski 2006a) argues that a role therefore characterises a *private information space*, where data is used by that role to realise internal and instantiate external activity. It is argued here that the information space of a role can be divided between that of the *environment* (that part of the role in which human activity exists) and the *machine* (that part of the role in which machine activity exists) in order that the specification can represent the interface between the environment and the machine, without losing the connection to the original role and associated process. Retaining this connection is the essence of this method and offers an advantage over other methods in facilitating a better alignment with the business process through specification into design. Therefore, the RAD notation is adapted to increase the information space of each role.

Once completed, relationships between the environment and the machine become evident; known as *shared phenomena* (Gunter et al. 2000; Jackson and Zave 1995). Such phenomena can be questioned; posing interesting questions for both analysis and design.

6.4.3 Moving from Specification into Design

Since all xMDA method transformations focus on a single notation, there are real benefits in that mappings are preserved throughout the process and designs are offered based upon the analysis that necessitated them. “A specification is a starting point for programming” (Jackson and Zave 1995) and to move the specification closer to design, a final phase (the *Machine RAD*) is therefore utilised to focus purely on the machine elements revealed by the *Shared RAD* (see figure 6.4.3.1).

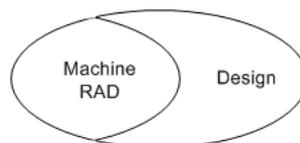


figure 6.4.3.1, *Machine RAD of Specification.*

The *Machine RAD* is used to influence the design of the solution system by providing an appropriate interface from which first-cut design models can be extracted (see Section 6.4.4). Here, environmental elements are completely removed from the *Shared RAD*, leaving only *Machine-to-Machine* relationships within and between roles to be analysed. The central aim here is to complete the removal of environment elements in order that, through the specification process, they do not appear in design. There is no requirement for this model to be transformed into a Use Case (or any other intermediate notation) since the method is thorough enough to negate the need (although the technique supports this as a transformation should that be the will of the designer, see Section 7.6.3 for further discussion).

6.4.4 Class Discovery, Transformation and Platform Information

Once completed, further analysis of the *Machine RAD* can be conducted to reveal hidden information about candidate *entity*, *interface* and *control* classes in moving to design, and Transformation and Platform Information in the form of *transformation rules* can be applied to the *Machine RAD* to derive first-cut models to be used as a starting point for design (see Chapter 7.0 for further information).

6.4.5 Iteration

“Business changes inevitably lead to changing requirements” (Sommerville 2004) and therefore, changes to the specification and solution system. Requirements elicitation and analysis are iterative in nature (Sommerville 2004) but accounting for iteration in resolving requirements can be a problem in software process modelling (Dowson 1987c; Tully 1987). Therefore, in order that models are refined and requirements and PD elements are appropriately accounted for, it is important that iteration is supported in a modelling and transformation method that is “comprehensible for all stakeholders during the respective development phase” (Karow and Gehlert 2006). With the MDA, it is possible to revisit models generated for use at any phase (Garrido et al. 2007); a concept central to the xMDA method. However, checking for consistency between specifications will always involve an element of human participation (Tully 1987).

6.5 Summary

This chapter has argued the need for extending the MDA to account for RE and proposed an extension which places Jackson’s (1995) theory of specification at heart. In outlining a method to support the xMDA framework, it is important to consider that every situation is different and some techniques may have a greater applicability than others, depending on any particular given situation and PIM level technology. Such

methods are not intended to simplify difficult development problems, rather they are intended to guide the user through suitable steps to provide sufficient support (Finkelstein 1987). The xMDA method is proposed to support the xMDA framework. This is suggested to provide a CIM level interface between business and software users and guidance on the successful application of the method to the MDA in defining requirements and realising software solutions. It is possible that *any* notation could be used, provided each described xMDA specification phase (*Environment*, *Shared* and *Machine*) is addressed within the CIM and relevant Transformation and Platform Information is available. "Clearly, it would be naive to claim that the profound problematics of knowledge representation can be overcome by diagrammatic considerations alone. Nevertheless, every little improvement helps" (Harel 1988).

The validation of the xMDA framework is addressed in the following chapters, with the next chapter illustrating the xMDA method in detail. Once established, Chapters 8.0 and 9.0 investigate the value of the xMDA framework and supporting method for business and software use in the academic and commercial context.

Chapter 7

xMDA Illustration

In the previous chapter, the underlying theory behind a method to support the xMDA extension was described. In this chapter, a case in point is used, illustrating the application of the xMDA method.

7.1 Order Processing Worked Example

Described below is a sample fictitious case that is used as focus for this illustration. In the subsequent sections, each xMDA method phase is applied and discussed in detail.

The XYZ Company is a medium sized business that is looking to develop a software system to support a small scale mail order business. Because of the logistics involved with the enterprise system that is already in place, the new support system is to be interoperable with other systems and have a portable and reusable architecture. The following describes the activities of the mail order process.

The marketing department is responsible for the creation and despatch of all marketing materials.

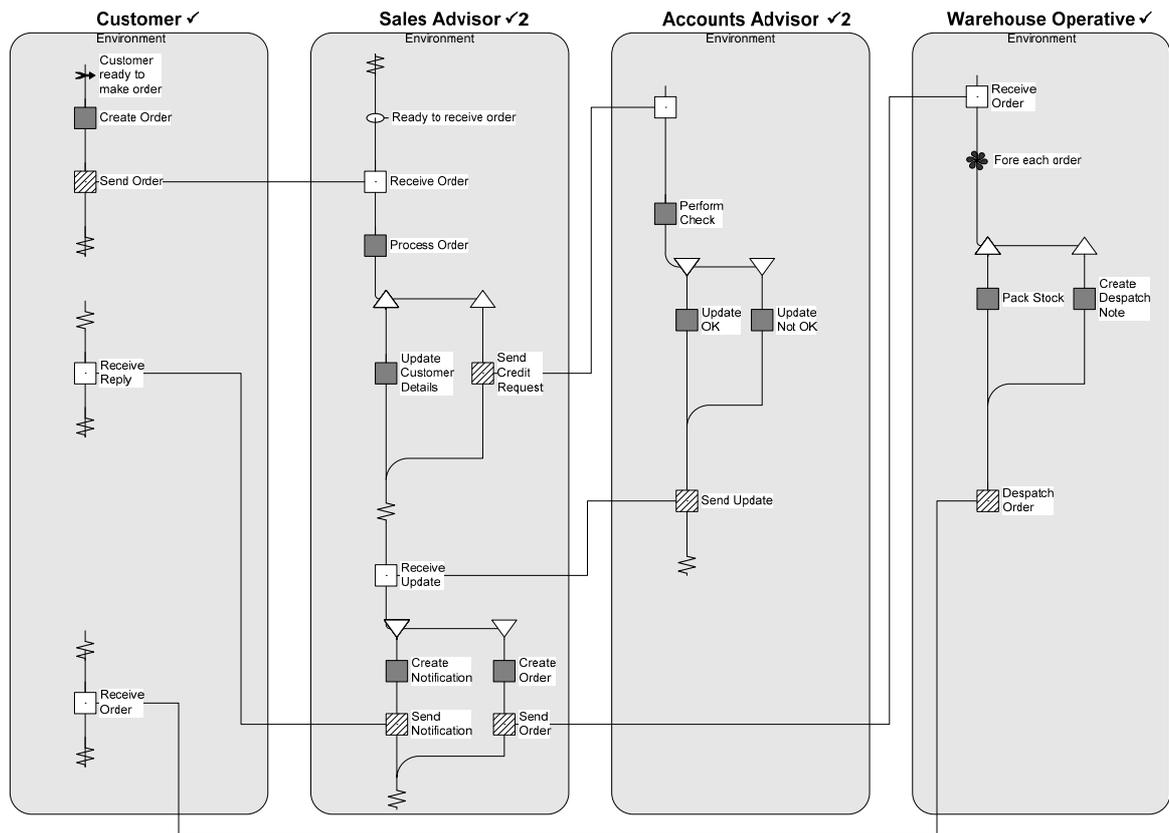
Once a customer is ready to make an order, they fill out the order form and mail it into the company. Once received, a sales advisor processes the order. The customer details must be kept on file for future reference and credit checks are made to ensure the customer account is in credit before any order can be authorised.

The accounts advisors are therefore required to respond to any sales order request by providing an update on client credit worthiness. If a client has poor credit, the sales advisor notifies them of the situation, otherwise an order is created and sent to the despatch team; who pack and despatch the order, along with required despatch note.

A copy of the despatch note is forwarded to the accounts advisors for billing purposes, which is already supported by the accounting system. There are two sales advisors and two accounts advisors available to the mail order enterprise. The warehouse team numbers varies with seasonal demand and is unlikely to have any impact in the design of the new support system.

7.2 Environment RAD

The first phase of the xMDA method involves extracting detail from the analysis of the PD and representing it in a RAD. In this example, the PD description was analysed, resulting in the construction of the RAD given in figure 7.2.1.



Note: Enlarged version appears in Appendix I, figure 1-1.

figure 7.2.1, Environment RAD for the Order Processing example.

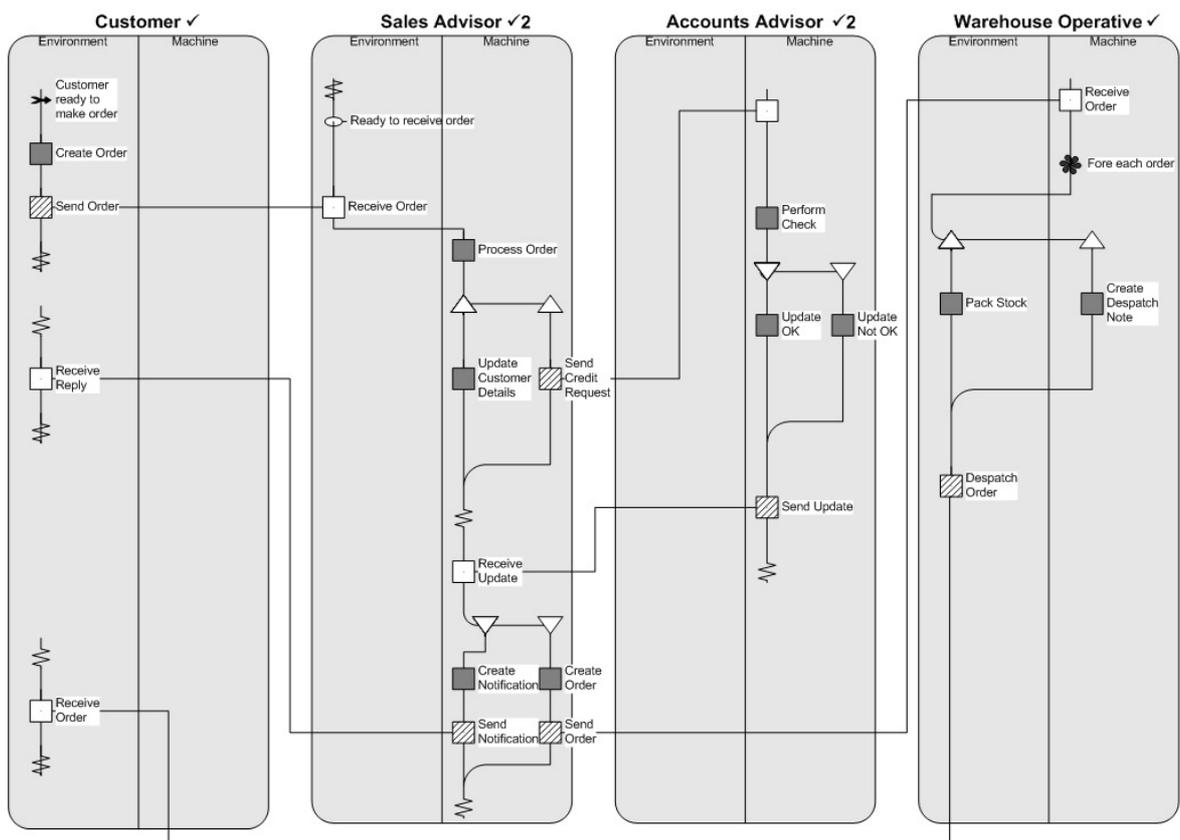
Typically, the *Environment RAD* is constructed in conjunction with other tools for requirements elicitation and analysis, such as the DDM, IRM and BM described in Section 4.2, and therefore the *Environment RAD* is the output from such elicitation and analysis. Thus, it is considered to be more of a complete offering of the phenomena required for specification rather than a simple process model (although it could act as one).

Here, *Environment-to-Environment* relationships are identified and considered. By examining interactions, such as the *Send Order* and *Receive Order* interaction between the *Customer* and *Sales Advisor*, aspects of the environment can be considered in isolation of machine complexities. Relationships are compared and contrasted with those defined in the PD and initial requirements to ensure the model is correct and complete,

with any inconsistencies being addressed and resolved with the stakeholder via iterative mechanisms. The *Environment RAD* serves as input to the second phase of the xMDA method.

7.3 Shared RAD

This next phase addresses issues central to specification. The *Environment RAD* is refined to reflect the behaviour of the system to be developed with respect to its operating environment. Each individual user or system role is divided into separate conceptual spaces of the *environment* and *machine*, particular to that user or system. The role is divided by a boundary line, with elements pertaining to either the environment or machine being placed in the respective space, separating the tasks a role might be required to complete in the real-world or via the machine. The output of this process is known as the *Shared RAD*. Figure 7.3.1 illustrates the *Shared RAD* for the *Order Processing* case.



Note: Enlarged version appears in Appendix I, figure 1-2.

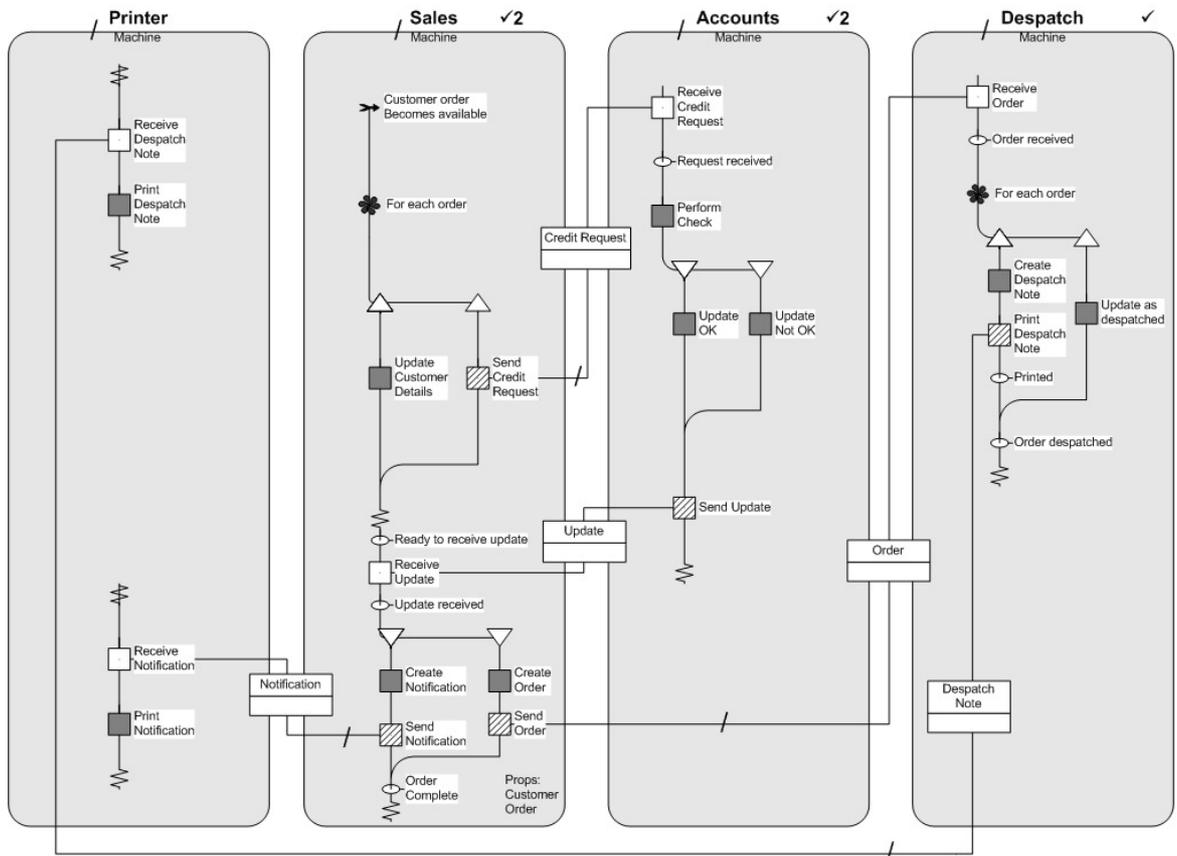
figure 7.3.1, *Shared RAD for the Order Processing example.*

Each user or system role reflects the requirements for that role; and the interfaces between the environment and the machine. Any element included within the environment space of a role signifies a relationship between the role and the external environment. Conversely, any element included within the machine space of a role specifies a relationship between the role and the machine. For example, since the order form is created and sent in the real world (external to the system), the actions and interactions that fulfil those requirements appear in the environment side of the *Customer* role. If the order processing system was online, those actions and interactions would appear on the machine side of the role, describing an interface between the *Customer* and the machine. This is useful since it explicitly highlights human and machine interaction by role, and across roles, in the context of the process. Therefore, rich detail about the environment, machine and the boundary in between, is carried forward in specification, allowing for specification and design to share a better alignment.

Here, shared phenomena (*Environment-to-Machine* and *Machine-to-Environment*) associations can be interrogated. This is important to ensure that the correct interfaces present the user with sufficient access to the machine and that the machine represents required elements, enabling accurate support for specification when moving to design. The *Shared RAD* serves as input to the third phase of the xMDA method.

7.4 Machine RAD

This final phase involves moving the specification closer to design. Machine elements and shared phenomena from the *Shared RAD* are interrogated; each connection is investigated further, along with the stakeholder (iterating where necessary), with a view to de-coupling the environment from the machine and ensuring that remaining connections are as complete and correct of requirements as possible. The *Machine RAD* is the resulting artefact (see figure 7.4.1).



Note: Enlarged version appears in Appendix I, figure 1-3.

figure 7.4.1, Machine RAD for the Order Processing example.

The de-coupling can result in a renaming of roles to better align them with the machine. From analysing the *Machine-to-Environment* relationship of *Send Notification* and *Receive Notification* interaction between the *Sales Advisor* and *Customer*, it was discovered that the interaction is to actually be one with the *Printer*; from which the printed *Notification* would be despatched to the client in the environment of the real-world. Similarly, investigation of the *Warehouse Operative* role revealed a relationship with the *Printer* role whereby the *Despatch Note* was printed. In de-coupling the *Environment-to-Machine* relationship that existed within the *Sales* role (from *Receive Order* to *Process Order*, the *Customer Order* prop was associated with *Sales*, and the trigger and replication notation was used; this replication notation was also added to the *Despatch* role in a similar fashion; replacing the cross-boundary part refinement.

This *Machine RAD* is proposed to replace the need for moving to Use Cases in that the information contained within it is a representation of the behaviour of the system to be designed, based on the roles specified to use the system and the requirements associated from the analysis of the PD. Since the method forces the user to

make explicit consideration of machine elements in the process of guiding the user from analysis into design, the additional effort is suggested to be worthy over that of other descriptions (such as the Use Case). Of course, some requirements (such as those which are non-functional or abstracted away in the process model) may require further investigation and therefore it is proposed that other mechanisms for requirements and traceability are employed alongside this method; specifically the early derivation of test cases related to the requirements of the system to be developed.

7.5 Class Discovery

Part of the description of the xMDA involved providing an optional mechanism to offer class discovery from the *Machine RAD*. Since no such mechanism exists in practice, one has been defined specifically for the xMDA task. In object oriented programming, three stereotypes are “often reserved for object identification” (Cox and Phalp 2007). They are *Control*, *Entity*, and *Interface* (Edlich et al. 2006; Harrington 2000). Although others can be added (for example *GUI* and *Database* as described in Cox and Phalp (2007)), the stereotypes identified by this research are given in table 7.5.1.

Stereotype	Description
Entity	Data access and retrieval class
Interface	Connection class
Control	Functional communication management class

table 7.5.1, stereotype descriptions for class discovery (Source: developed from Cox and Phalp (2007)).

An analysis technique is defined here which utilises the stereotypes described above. The *Tri-Step* analysis consists of directly interrogating the *Machine RAD* for stereotypes and can provide interesting insight into the discovery of candidate design classes of each class type. Entity, interface and control classes can be discovered from a complete *Tri-Step* analysis of the *Machine RAD* and each stereotype is given consideration in the subsequent sections, with the complete Tri-Step analysis for the worked example following. This analysis is suggested to be useful in finalising models output from the application of Transformation and Platform Information at design-time to aid understanding of provided models. Once classes are identified, attributes, services and structures (such as inheritance, aggregation and association) can be considered for application (Cox and Phalp 2007).

7.5.1 Entity Classes

An entity class is used to represent the data elements of a system. Candidate entity classes can be derived by examining *Machine-to-Machine* interactions, with potential hidden classes being uncovered by using a similar

technique as described in Phalp and Cox (2001). For example, the *Send Credit Request* and *Receive Credit Request* interaction in figure 7.4.1 between *Sales* and *Accounts* roles reveals an interaction that contains an exchange of data (hereby called a data-interaction). Interrogation of this interaction can result in the entity *Credit Request* (Credit Request <<entity>>) being discovered, which contains customer details for the credit checking process. These data-interactions are highlighted in figure 7.4.1 with the explicit representation as UML Classes. Such data discovery can also be achieved by examining independent activities (known as data-actions) and state transitions (data-states) of the *Machine RAD*. For example, the discovery of a *Notification* (Notification <<entity>>) class can be made from examining the *Create Notification* data-action; the discovery of the *Customer Order* (Customer Order <<entity>>) can be made from examining the data-states involving the *Customer Order* prop in the *Sales* role, and so on.

7.5.2 Interface Classes

An interface class is used to represent a system element that facilitates the connection of classes to other classes, or to the outside world; enabling the input and output to and from users and alternative systems to be connected. Candidate interface classes can be derived from the *Machine RAD* in three ways. The investigation of the imposed connection between the user or system role and any machine element; any connection that exists within a role to a prop; and any interaction between roles, can result in the derivation of candidate interface classes. For example, a graphical user interface can be identified for the *Sales* role, replacing the role with the Sales Graphical User Interface (GUI) (SalesGUI <<interface>>); an interface could be extracted from the *Sales* role to *Customer Order* prop (i.e. to interface (Customer Order <<interface>>) with the Customer Order database); and in order for *Sales* to interact with *Accounts*, an interface to connect those classes is likely to be involved (Sales / Accounts <<interface>>). Interfaces can be denoted with the / symbol on the *Machine RAD* (see figure 7.4.1).

7.5.3 Control Classes

A control class is used to represent a system element that manages communication; typically between entity and interface classes. Candidate control classes can be derived from the control processes of a RAD interaction and the logic flow of RAD state transitions. For example, the driving class behind the interaction *Send Credit Request* and *Receive Credit Request* is the control class (Sales <<control>>), *Sales* being the driving role having elicited relevant information for the *Credit Request* and made the original initiation to interact with the *Accounts* (Accounts <<entity>>) class (denoted by the standard RAD notation of the square box with right-sided upwards diagonal shading for identifying driving roles). The control of logic flow within a user or system role is by each individual role; any other control is a concern external to the machine and ought to be considered in that context.

7.5.4 Tri-Step Analysis

This analysis has been conducted here for the case presented at the start of this chapter, resulting in the potential classes highlighted in table 7.5.4.1; descriptions of relationships are made following the technological syntax described in Cox and Phalp (2003), Cox et al. (2005b) where domain 'DO' is responsible '!' for {x} phenomena.

Relationship	Description	Class Name	Type
Send Credit Request	SA!{send credit request}	Credit Request	<<entity>>
		Sales	<<control>>
		Accounts	<<entity>>
		Sales / Accounts	<<interface>>
Print Notification	SA!{print notification}	Notification	<<entity>>
		Sales	<<control>>
		Printer	<<entity>>
		Sales / Printer	<<interface>>
Send Order	SA!{send order}	Order	<<entity>>
		Sales	<<control>>
		Despatch	<<entity>>
		Sales / Despatch	<<interface>>
Send Update	AC!{send update}	Update	<<entity>>
		Accounts	<<control>>
		Sales	<<entity>>
		Accounts / Sales	<<interface>>
Print Despatch Note	DE!{print despatch note}	Despatch Note	<<entity>>
		Despatch	<<control>>
		Printer	<<entity>>
		Despatch / Printer	<<interface>>
Sales to Customer Order	SA!{access}	Customer Order	<<entity>>
Sales to Customer Order	SA!{access}	Customer Order	<<control>>
Sales to Customer Order	SA!{access}	Customer Order	<<interface>>
Printer to Machine	PR!{access}	PrinterGUI	<<interface>>
Sales to Machine	SA!{access}	SalesGUI	<<interface>>
Accounts to Machine	AC!{access}	AccountsGUI	<<interface>>
Despatch to Machine	DE!{access}	DespatchGUI	<<interface>>

table 7.5.4.1, potential design classes derived from a Tri-Step analysis of the Machine RAD for the Order Processing example.

It is feasible that this analysis could be automated to some degree and included in any software support for the method, provided user interaction is available to delete, amend or add classes (this could also be useful for the management of classes in design).

7.6 Transformation and Platform Information

7.6.1 Transformation Information

The next step of the xMDA method involves the application of Transformation Information to relate the *Machine RAD* with the desired platform (in this case the UML – See Section 7.6.2). The choice of particular notations to illustrate the xMDA orchestrates the selection of Transformation and Platform Information. Similarly, as with defining the *Tri-Step* analysis mechanism for class discovery, Transformation Information to relate the UML platform with the RAD does not exist. Therefore, part of describing the xMDA method necessitates the creation of this information, which is the focus of the remainder of this chapter.

Metamodelling is “a common technique for defining the abstract syntax of models and the inter-relationships between model elements” (Sendall and Kozaczynski 2003), supported by FT (2007). The OMG describe a four-layered architecture for Software Engineering with the UML and the MOF (OMG 2006a, 2007c) (see table 7.6.1.1).

Level	Description
M3: Meta-meta	“a self-defined language to define other languages at level M2” (MOF)
M2: Meta	“a set of domain specific metamodels” (UML/Java metamodel)
M1: Model	“Any model [that] is compliant with a specific metamodel” (Class)
M0:	“Instance level [describing an] execution” of M1 (Object)

table 7.6.1.1, the four-layered architecture of the OMG (Source: developed from FT (2007), Kusel et al. (2009), OMG (2006a, 2007c), Peltier et al. (2000), Sheena et al. (2003), Thiemann (2009)).

This four-layered architecture underpins model theory (FT 2007) whereby a model is defined as an entity which conforms to a metamodel, which in turn conforms to a meta-metamodel. The declarative QVT-Relations (QVT-R) language forms part of the QVT standard central to the MDA and “allows for the creation of [a] declarative specification of the relationships between MOF models” (Giandini et al. 2009). This means that the QVT-R can be used to define transformation relations at the M2: Meta level between models conforming to the MOF meta-metamodel.

In order to “work on the basis of this theory, specifying a metamodel which describes... domain concepts (for instance UML proposes Use Case, Activity and Class concepts) is needed” (FT 2007). Therefore, since the output from the xMDA method is the *Machine RAD* in the modified RAD notation, a RAD metamodel conforming to the meta-metamodel of the target platform (in this case the MOF) is required. Only one known example of a RAD metamodel exists in Badica et al. (2005), which was extended in Section 5.1.1. Here, this

extension is continued, forming the *SimpleRAD* metamodel in line with the RIVA modelling methodology given in Ould (2004c) and the MOF defined by the OMG (2006a) (see figure 7.6.1.1).

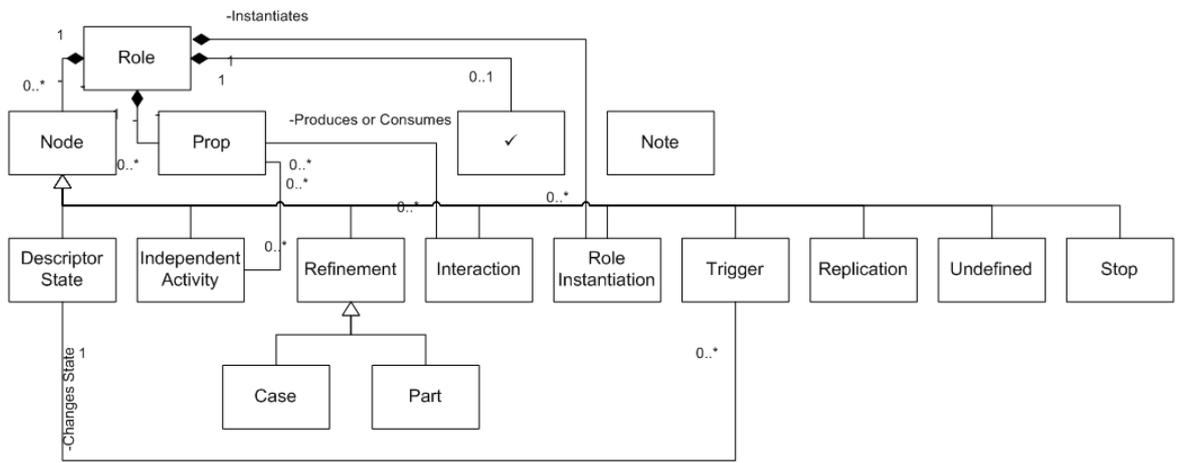


figure 7.6.1.1, *SimpleRAD* metamodel (Source: developed from Badica et al. (2005), OMG (2006a), Ould (2004c)).

Tools that can be used to define metamodels of domain specific languages include Dome, GME, MetaEdit+, ParadigmP (Sendall and Kozaczynski 2003); the eMOF textual representation demonstrated in FT (2007) could also be used to demonstrate metamodels.

7.6.2 Platform Information

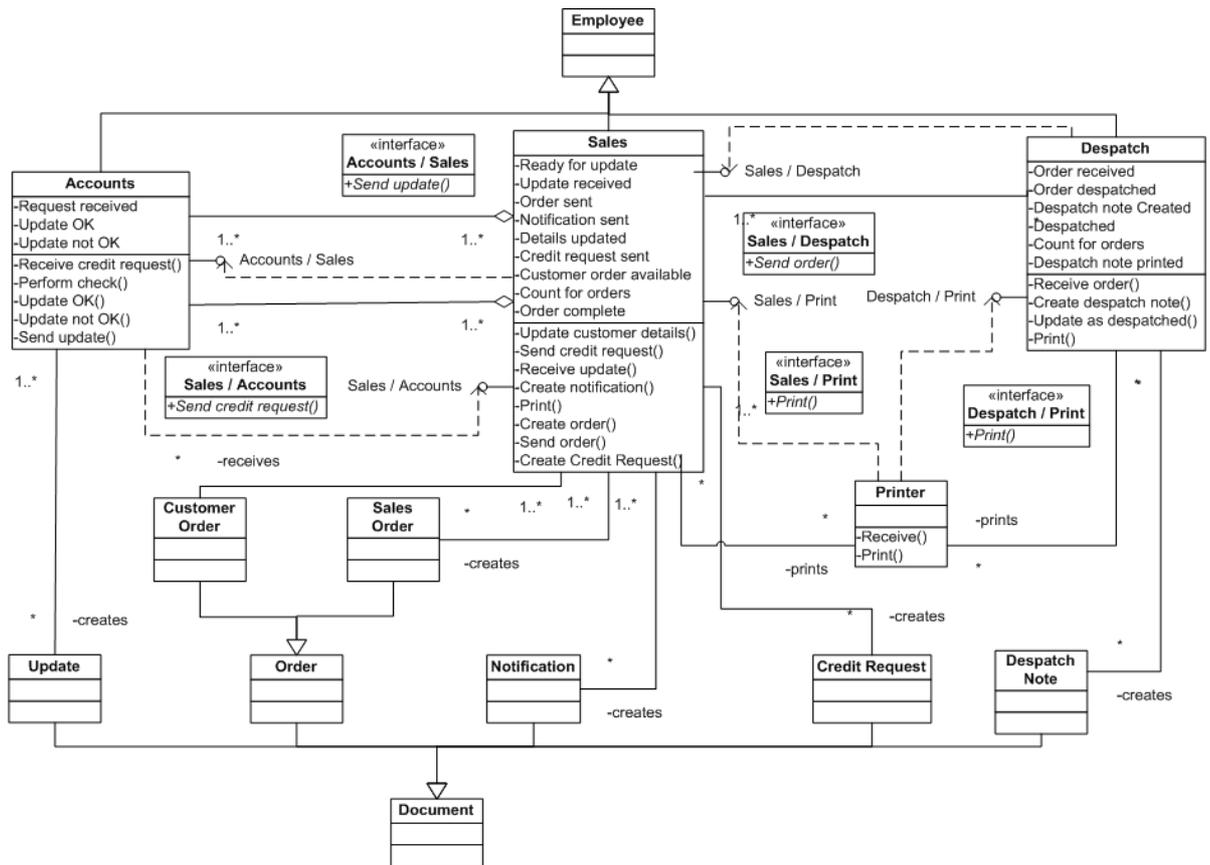
Transformation Information alone is not enough to facilitate moving from CIM-to-PIM; the application of Platform Information is required to draw components of one from the other. This section draws on a simplified UML metamodel to define transformation rules at the meta-level from the RAD to the UML. Focus here is given to the Class Diagram to verify the application of Platform Information because it is considered to be the most useful model for PIM description (OMG 2003b) and relates to discoveries made in Section 5.1.1.10 where these concepts first appeared in CIM-to-PIM transformations involving RUD fragments to create a foundation Class Diagram. The UML is defined here by the *SimpleUML* metamodel and associated with the *SimpleRAD* metamodel (see figure 7.6.2.1).

RAD	UML Class Diagram
Role	Class
Independent Activity	Operation; Class & Association if Object results from Activity
Looping, Line and Descriptor States	Attribute (only applicable for Descriptor States)
Case Refinement (Alternatives)	Attribute
Part Refinement (Concurrency)	Attribute; Composition if refinement objects are descendent from resulting object
Interaction	Association; Operations in Role Classes; Aggregation if Role interactions are exclusive to only a single Role
Role Instantiation	Role Class; Operation in Source Role Class; and Association
Trigger	Attribute
Replication	Count attribute
Undefined	Check context; May define alternate Class Diagram
✓	Multiplicity
Prop	Class; Association with Role Class
Stop	Attribute in originating Role Class
Note	Note

table 7.6.2.1, initial transformation rules to map from the RAD to the UML Class Diagram.

7.6.2.1 Class Diagram

To demonstrate how the described Transformation and Platform Information is applied, the rules described in table 7.6.2.1 were applied the *Machine RAD* given in figure 7.4.1 which was derived as part of the xMDA method. The Class Diagram in figure 7.6.2.1.1 resulted from this application process.



Note: Enlarged version appears in Appendix I, figure 1-4.

figure 7.6.2.1.1, UML Class Diagram for the Order Processing example.

As per the rules defined in figure 7.6.2.1, the resulting foundation Class Diagram to be passed to design contains the four central classes derived from the *Printer*, *Sales*, *Accounts* and *Despatch* roles of the *Machine RAD*. The prop *Customer Order* has also resulted in a class, which has gained inheritance since both the *Sales Order* and *Customer Order* are deemed to inherit from an *Order* super-class type. This type of manual embellishment is useful, allowing other roles to be related as *Employee* class types and included to demonstrate how a first-cut Class Diagram can be used for, and to inspire, design related decisions post-CIM. A further example of this is the inclusion of the *Create Credit Request* operation in the *Sales* class to account for the data object that is required to result from the *Sales* class before the *Send Credit Request* interaction can take place. Ideally, this sort of decision would be made further upstream when examining the process. However, it might not be until the *Tri-Step* analysis, or well into design, that the importance of the creation of the *Credit Request* object be recognised. Either way, this demonstrates the significance of this type of manual enhancement to the first-cut Class Diagram. Independent activities relate to the operations of those classes where, on occasion, new classes have been generated where a data objects result from that operation (for example, *Despatch Note* being created by the *Despatch* class). Attributes have manifested from descriptor

states (for example, *Request received* in the *Accounts* class), part refinements (for example, *Credit request sent* in the *Sales* class), case refinements (for example, *Update OK* in the *Accounts* class), triggers (for example, *Customer order available* in the *Sales* class), and replications (for example, *Count for orders* in the *Despatch* class). Interactions have defined associations between classes, with aggregation being present in certain cases where interactions are exclusive between single roles (for example, between *Sales* and *Accounts* classes in the *Send Credit Request* relationship). Multiplicity has been defined from the ✓ notation (for example, 1..* *Sales* instances may be involved with * *Despatch* instances). The Class Diagram has been further elaborated to account for a selection of classes discovered from the *Tri-Step* analysis conducted in Section 7.5.4, specifically the interface classes, to demonstrate how these might be included and the Class Diagram refined at design-time.

7.6.3 Transformation Rules

To extend this application, the process was repeated to account for UML Activity and Use Case diagrams to demonstrate how the solution could support other CIM-to-CIM and CIM-to-PIM transformations, provided necessary Transformation and Platform Information is defined. As noted in Section 6.4.3, the *Machine RAD* negates the need for moving to the Use Case diagram in defining specification although the transformation may be useful to those requiring Use Cases to support understanding and communication in the development process. The complete set of transformation rules which are described as part of this research in the direction of the UML from the RAD are presented in table 7.6.3.1, along with a natural English description and example of the provided transformation rule.

Natural English	RAD	UML Class Diagram	UML Activity Diagram	UML Use Case	Example
Noun referring to human or system	Role	Class	Activity Partition	Actor; Relationship to Use Cases derived from that Role	Project Manager
Verb with direct object (noun)	Independent Activity	Operation; Class & Association if Object results from Activity	Activity	Use Case. Chunk of activity may define a single Use Case	Writes report
Clause where sequence is defined (before or after)	Looping, Line and Descriptor States	Attribute (only applicable for Descriptor States)	Transition; Synchronisation Bar if flow is split	Check context; May define extend/include relationships	Ready to write report; Writes Report; Ready to send report; Sends Report
Clause joined with "or" conjunction	Case Refinement (Alternatives)	Attribute	Decision Diamond; Guard	Relationship; Check context; May define extend/include relationships	Write report or Delegate task
Clause joined with "and" conjunction	Part Refinement (Concurrency)	Attribute; Composition if refinement objects are descendent from resulting object	Synchronisation Bar	Include relationship	Writes and Sends Report; Write report a) and report b) to be contained in report c)
Sentence containing Role subject and object nouns with verb; optionally modified by adverb	Interaction	Association; Operations in Role Classes; Aggregation if Role interactions are exclusive to only a single Role	Activity; Transition	Source and destination Use Case; Relationship. Chunk of activity may define a single Use Case	Project Manager quickly sends Report to General Manager and Contractor
Verb with Role noun initiating new Role noun	Role Instantiation	Role Class; Operation in Source Role Class; and Association	Activity Partition; Activity and Transition	Actor	Project Manager selects Contractor
Noun referring to an event that starts a process	Trigger	Attribute	Start; Note	Note	Complaint is received
Determiner associated with activity	Replication	Count attribute	Decision Diamond; Guard; Transition (loop) encapsulating replicated activity	Note	For every application received, assess it
Where sequence is undefined (before or after)	Undefined	Check context; May define alternate Class Diagram	Check context; May define alternate Activity Diagram; Transition; Stop	Check context; May define alternate Use Case; Relationship	The project manager writes report; The project manager owns a car
Determiner associated with Role noun	✓	Multiplicity	Note	Multiplicity	There are 500 employees; The Project Manager
Verb and noun consumed by Role noun	Prop	Class; Association with Role Class	Note	Check context; May define alternate Actor; Use Case; Note	Uses database
Sequence terminating verb	Stop	Attribute in originating Role Class	Stop	Note	Project Ends
Adjective modifying noun	Note	Note	Note	Note	Project Manager is logged in

table 7.6.3.1, complete set of initial transformation rules to map from the RAD to the UML.

It is agreeable that such rules are perhaps better to be considered as *guidelines* “since it is possible to find counterexamples to them” (Chen 1983) and therefore the involvement of the user is suggested to enhance the application by determining the extent to which the rules should (or should not) be followed. Therefore, it is important to note that although elements are declared as related, no such declaration is made on equivalence and any software implementation of these rules ought to support user interaction in terms of model modification and/or a *wizard* mechanism for the verification of the treatment of model elements. This is to say that meta-level elements might be derived from one another, but are not suggested to be in direct replacement, and it is not the intention here to evaluate the extent to which elements might be used in replacement. Therefore, the remainder of this research will focus on the transformation rules given in table 7.6.3.1 and the application of such rules, rather than a discussion on how elements might exhibit such equivalence.

7.7 Summary

This chapter has defined and demonstrated an approach to the xMDA that utilises the RAD to move incrementally from analysis to design, through a series of model refinements, and described a set of transformation rules based upon Transformation and Platform Information involving the *SimpleRAD* and the *SimpleUML* metamodel definitions.

To be a specification, it is suggested in Jackson and Zave (1995) that focus be given to shared phenomena, that the onus of phenomena control is on the machine, and that events and/or states are used to represent event restrictions. This is achieved by this method since each phase is focussed on revealing and advancing connections between the environment and the machine. The *Machine RAD* ensures that control is central to the machine; providing interface to the environment where it is not. The very nature of the RAD is that it is event based, using state transitions to move the process from one state to the next. Therefore, requirement constraints are represented in the RAD via events and pre/post states.

Rules for the transformation of RADs into Use Case, Activity and Class diagrams ensure that what is defined in the business model is received by the systems specialist in a language to which they are accustomed to (the UML), making the xMDA method useful not only for business use in defining the CIM, but also to the software engineer in design. These rules represent a basis for the development of software support that includes the RAD and the UML. Furthermore, they have considerable implication in terms of the MDA in forming a real connection between the CIM and the PIM phases.

The resulting artefact is a complete method consisting of three related viewpoints that are founded upon requirements elicitation and analysis, and an analysis technique with a set of rules to transfer that knowledge into the heart of software development, within the xMDA framework. By presenting specification in the CIM

via RAD, it is argued that a starting point for object oriented system design can result from the definition of human-driven processes; and be traceable back to originating requirements, without the need to define specification via Use Cases. The method ensures that by defining metamodels between the CIM and the PIM, classes can be transformed from founding events into design. It is further suggested that this in turn will also account for the alignment of business strategy with the IT process, since the RAD is defined and formed by the nature of the *Process Trinity*. The remaining chapters give focus to verifying the value of the extended mechanisms of the xMDA in practice, via the xMDA method.

Chapter 8

Moving from Analysis to Design via xMDA

Using techniques to assist the transition from analysis to design is important in ensuring the alignment of business requirements and software implementations. However, this is a commonly difficult and misunderstood task. The xMDA framework and method, proposed and described in Chapters 6.0 and 7.0, are suggested to enhance experience and understanding for and between the business and software interest. This chapter looks to address aim 4 in part validating the provided solution in academia to learn whether or not the technique is viable, and what challenges might be involved in comparison with other available techniques. Three alternate approaches (including the xMDA method) in moving from analysis to design were presented to 47 Honours level students of BPR on the Software Systems framework at Bournemouth University, with the results of student participation being highlighted and discussed. It is thought that Business Analysts, with limited or no knowledge of Software Engineering and a basic impression of RE, may share similar experiences with the students, since they also are perceived to have both a limited knowledge of and an appreciation for the move from analysis to design, and therefore, the research may be transferable to this setting. The argument here is that, if the xMDA method is found to be viable and accessible to the students, it may be practical in industry due to those similarities. Of course, much of what is learned *is* subjective. However, it is also valuable in gaining an overall sense of the impression that users with rudimentary BPM and RE knowledge might have regarding the viability and accessibility of techniques.

8.1 Academic Application

As described in Section 6.2, requirements are the desired effects in the environment to which the solution system is proposed to bring about; the specification is a description of the solution system that can fulfil such requirements and are defined by “reasoning about the environment” (Jackson and Zave 1995). This reasoning is commonly termed *analysis*. Analysis, specification and design notations are often orthogonal (Cox and Phalp 2007; Phalp 2002) and can represent different views of the software process (see figure 8.1.1).

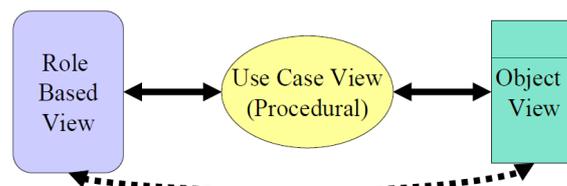


figure 8.1.1, orthogonal notations and the software process (Source: Phalp (2002)).

The *Role Based* view typically represents an analysis model (for example, a RAD) where activities are spread across *roles*; the *Use Case* or *Procedural* view can represent a specification model (for example, a Use Case) where *actors* are spread across *activities*; and finally, the *Object* view commonly represents the internalisation of the solution system (for example, a UML Class Diagram) where *Objects* represent the integration of *activities*. Phalp (2002) highlights that these orthogonal domains relate to the capacity of models and notations to represent clear and consistent mappings between domain.

Solution mechanisms to overcome this difficulty have been described by various authors and focus here is given to the two that are taught on the BPR unit at Bournemouth University, and the xMDA method described as part of this research. Process Oriented Systems Design (POSD) was defined in Henderson and Pratten (1995) and represents a structural mechanism for describing a system, that can act as an intermediate technique in moving from a RAD to a Use Case (Phalp 2002). A second technique known as the SystemRAD, developed by Dr. Keith Phalp of Bournemouth University, is used in a similar manner. This approach introduces the system boundary to the RAD, which facilitates transformations into Use Cases. The final technique has already been presented in great detail through Chapters 6.0 and 7.0 and suggests that the RAD could contain enough information to describe specification, negating the need for the Use Case, whilst allowing for the generation of system level models based on specification with Use Case derivation if required.

Other approaches used to remedy the issues in moving from analysis to specification include enhancing the Use Case definition with Enactable Use Case tools (Kanyaru 2006; Kanyaru and Phalp 2005; Kanyaru and Phalp 2009) that facilitate the representation of behavioural dependencies via states; the B-SCP framework for strategic alignment (Bleistein et al. 2006), which focuses on the integration of business strategy, context and process; and the Problem Frames approach, presented in Cox and Phalp (2003), Cox et al. (2005b), which attempts to guide the application of Problem Frames described by Jackson (1995) to process models.

8.2 Thematic Analysis

The three methods of moving from analysis to specification and design were presented to 47 Honours level students of the BPR unit on the Software Systems framework at Bournemouth University in November 2009 and informed consent was given by participants to be involved in this research. There was no grouping in consideration of student selection and response was related to the completion of the second part of the coursework for that unit; therefore, the sample selected was randomised (Deveaux et al. 2005). The students were both male and female, each having at least one year industrial experience prior to attending the BPR unit. Each approach was given a two hour timeslot where the methods were demonstrated to the students and

opportunity provided for the students to interact and ask questions about the usage and application of the methods. The two BPR unit approaches were taught by Dr. Keith Phalp; the author attended as a guest speaker on the 26th November 2009 to outline the xMDA method, with the presentation slides being made available to the students via the internet after that date. The question that was posed for coursework submission relating to this research was as follows - the complete assignment brief is given in Appendix II:

Discuss the issues and solutions encountered in moving from analysis (the process models) to specification and design, and mechanisms that you would use to ensure alignment of the business process model (and business needs) and the IT system.

Manuscripts were written by the students without researcher intervention in the students' own time with an expectation of an estimated 5-6 hours in total to be spent on completing this part of the assignment, constrained by a one month submission date and a maximum one thousand word limit. The identity of individual participants is protected and, therefore, raw data associated with this output is excluded from the main text of this deliverable. However, three samples of student feedback (with the identity of the individual students being protected) are included in Appendix II, figures 2-1, 2-2 and 2-3, to allow the reader to gain a sense of responses received. This research consists of drawing upon student discussions and discovering the factors which were perceived to be influential and identifying the personal experiences of and attitudes towards given solutions in the transition from analysis to specification and design.

Responses were imported into QSR International's NVivo software which was then used to systematically organise and analyse student feedback using NVivo's classification techniques to code themes. Central themes relating to the posed question were discovered from the analysis and spread of codes across responses and used as the basis for Template Analysis (see the respective sections of Appendix II for results and observations relating to this analysis). Grounded in data extracted from the written experiences of the BPR students, the purpose of this type of qualitative research is to focus on those individual experiences to gain an understanding (rightly or wrongly) of the central themes perceived by the student to be associated in moving from analysis to design via specification via Template Analysis.

8.3 Methods

In Section 2.2.4, the RAD is described with concepts of *role*, *activity*, *assertion* and *entity* as central to this notation. The modelling notation is useful since it caters for the dynamics of real processes, allowing for them to be flexible. Because of this ability to describe real-world processes in a rich context (including behavioural dependencies) and being simple enough to understand and implement, they are a candidate analysis tool

adaptable to methods used in moving from analysis to specification and design. The three methods described in this section all propose the use of the RAD as integral to each approach.

8.3.1 Process Oriented Systems Design (POSD)

Experience in the modelling of large scale enterprise and distributed systems has shown that notations (such as the RAD) suffer from abstraction issues in that models can become complex and convoluted (Henderson and Pratten 1995). Changing environments leads to a change in software systems that support the business process and such changes are often inadequately reflected. A visual model known as the POSD diagram is proposed to enable a business user interface (Henderson and Pratten 1995).

It is most likely that we would show only the top-level most abstract models to the business process owner, reserving the more detailed (and demanding) models to our own technical uses (Henderson and Pratten 1995).

In a POSD, *behaviours* are denoted by boxes, with touching boxes showing direct relationships between those behaviours (known as promises). Behaviours can sit within other behaviours (as sub-behaviours) or overlapping other behaviours (although this may lead to “cluttered diagrams” (Henderson and Pratten 1995)). A simple POSD can be used to create further abstractions by positioning behaviours appropriately in accommodating change, reviewing improvements and facilitating communication with Business Analysts (Henderson and Pratten 1995).

The POSD can be used to represent an abstraction on a RAD to hide the detail of the underlying process and simplify understanding. Phalp (2002) suggests that the POSD notation can be act as an intermediate technique in moving from an analysis RAD to a specification Use Case, enabling the discovery of Use Cases directly from the analysis and grouping of *Shared Behaviours* of a RAD (Phalp 2002). The method involves creating the base-level analysis RAD and a simple POSD representation of that RAD by abstracting away the detail of the RAD and grouping role activity into POSD behaviours, where promises are formed from RAD interactions. Additional viewpoints are then used to expand upon the POSD by adding connections representing those promises between behaviours (from examining the interactions between roles) and grouping related activities within roles (from examining the actions within and interactions between roles). These connections and groupings are then bundled together according to relevance, illustrating *Shared Behaviours* which are then matched to Use Cases (Phalp 2002). The alternate viewpoints are suggested to help guide the user to the Use Case from the RAD (Phalp 2002).

8.3.2 SystemRAD

Another method that can be used to guide the user from analysis to specification and design is known as the SystemRAD. This method, yet to be formally described, is similar to the POSD method in that it proposes the transformation from a base-level analysis RAD into a procedural Use Case via the SystemRAD modification.

The SystemRAD extends the RAD notation by providing a simple *System* role mechanism to capture behaviours that relate between the environment and the system being defined. This way, the system boundary is introduced to the RAD and all interactions and actions that might involve the system are included across the *System* role. The system boundary is an important consideration in the determination of requirements (as detailed in Section 6.2) because it defines where the software element is situated in the overall system. The abstraction is a fine line between interactive human and software elements. By adding the *System* role, an understanding of the system and how roles are to interact with it can be made.

Once the initial analysis RAD is completed, it is represented in a system context via the SystemRAD mechanism. The *System* role can then be mapped directly to the Use Case system boundary, with relevant Use Cases being created from the behaviours included within the role and roles interacting with the *system* role mapping to Use Case actors. Colour coding and numbering have been suggested to facilitate a visual understanding of the elements mapped in the process.

8.3.3 xMDA Method

The xMDA method (detailed in Section 6.4 and Chapter 7.0) suggests that a modified RAD can contain enough information to move directly from analysis through specification and into design by not focussing on procedural descriptions for specification, thereby negating the need to transform to Use Cases.

Strategic traceability is proposed to be accounted for through three modelling refinements, known as the *Environment RAD*; the *Shared RAD*; and the *Machine RAD*. These viewpoints mirror those described in Jackson (1995), being descriptions that are true of the PD, machine and of both. The *Environment RAD* is concerned with taking observations and information from analysis and depicting them in a standard RAD, focussing on the requirements and interactions that occur within a PD and are useful to specification. The *Shared RAD* takes the *Environment RAD* as input and splits the private information space of each role into two, distinguishing specifically the requirements and interactions that occur within the environment, those that occur (or are required to occur) within the machine, and those that occur between the two. This is because to “make the model work properly, so that it contains useful information about the domain, you also have to establish correspondences between the individuals in the machine and the individuals in the domain” (Jackson 1995). By focussing on the interactions between environment and machine, interfaces between them can be

identified and process problems resolved via further analysis. The final refinement extracts all environmental concern from the *Shared RAD*, giving focus to only the behaviours that are to occur in the machine and the interfaces required to the external environment and the involved users; the *Machine RAD* facilitates the RAD into design.

Designed specifically for the MDA, the method includes a mechanism to generate system level diagrams from the *Machine RAD* into UML Use Case, Activity and Class Diagrams, if required, associating the CIM with the PIM of the MDA, which is a concern highlighted in Fouad et al. (2009, 2011), Jeary et al. (2008).

8.4 Discussion

As previously noted, the objective of the analysis conducted in Appendix II was to learn from the personal experience of students in determining the issues felt to be central in moving from analysis to specification and design, and from exposure to the xMDA method, if the method would be received as viable and accessible to the students in consideration of those issues. The xMDA method received considerable student support in both the attitude and the limited experience gained of the method by the students, suggesting that it could be viable for practical consideration and addresses key concerns in moving from analysis to specification and design. Insights resulting from the analysis contained within Appendix II are considered in this section for discussion, specifically for the xMDA method.

8.4.1 Tool Support

Difficulties in moving from analysis to specification and design were identified by the students; these difficulties are compounded when the process is automated. This is because the move from analysis to design has never been considered to be an automatic one (Gustavson 2004; Kanyaru et al. 2008a); there are many unknown variables to be considered in the early stages of development and it is well documented that errors at this level can be costly (Brooks 1975; Greenspan et al. 1994; Kleppe et al. 2003; Sommerville 2004; STSC 2003; Wiegers 2000). It is agreeable that some aspects of the xMDA method could benefit if the logic complexity could be hidden from the user. Therefore, a semi-automatic approach would be considered in the form of tool support.

It is thought that tool support could be implemented in either one, or combination of, the following two ways. Firstly, heuristics could be used to enable the automatic generation of template *Environment*, *Shared*, and *Machine RADs* and transformations for users to work on, which would include an automatic traceability mechanism to ensure that each specification viewpoint is kept in sync. Secondly, since "human intervention is

always needed” (Gustavson 2004), the tool could operate a *wizard* mechanism that asks the user to verify how the tool should treat particular elements of input diagrams with all views being created as output from the *wizard*.

This could of course lead to the introduction of errors and ambiguities, away from the initial requirements defined by the Business Analyst, since it will be easier to make mistakes using tool support. Furthermore, whilst tool support reduces the complexity of the method for the Business Analyst; an increase in complexity and cost may be experienced in learning the tools and methods; the opposite of which is the key motivation for a light-weight approach to specification. A balance between the benefit of such an approach and the relative complexity needs to be met, which should be a consideration for further research.

8.4.2 Enterprise and Distributed Processes

No academic or practical application of the xMDA method is currently available because the method is new. Concern was raised about how the method might be applied to large scale business processes and systems, which may involve many requirements. Such systems are likely to involve requirements that are interdependent in and between projects and therefore it may be difficult to resolve dependency issues. Moreover, the management of large scale RADs could be cause for concern. These issues are not peculiar to RADs; almost every modelling notation experiences this in some form, with some (such as the DFD) finding sufficient solution in nesting and levelling (Cachia 2005).

The suitability of the method to large and distributed processes could be addressed by the investigation of such abstraction mechanisms. It is uncommon to find an individual that has the complete view and understanding of the process because of complexity (Dawkins 1998) and therefore, it may be impractical to expect a single model to do the same. The current view of a process could be recorded in the RAD by modelling different user or system roles directly. That is, focussing on a particular part enables the management of large and complex processes. Since it is uncommon in such a situation to find an individual that has a complete process view, it is plausible that a process architect (or leader) would be able to focus on an entire process by examining individual users (process owners and stewards) (Bilodeau 2010) and available system documentation. Abstraction can be used “...to describe systems in terms of hierarchies of their actual physical components” (Owen 2009a); a RAD can transform directly to a Context Diagram (Cox and Phalp 2003; Cox et al. 2005b), which would be perfectly viable for abstracting away from necessary detail in such an application.

8.4.3 Design Architecture

As previously noted, stakeholders commonly believe that they understand models in the same manner by which the software developer does and much effort and cost is associated with becoming a skilled modeller, and in aligning the paradigms of business and software (Cook 2004a), supported by Berrisford (2004), Phalp and Shepperd (1994). Since the RAD notation has been designed to model the business process, and not to design a computer system, it might at first seem foolish to consider the application of a such modelling technique to something that it clearly was not designed to do.

Analysis and design models are views derived from different environments. As found in Chapter 2.0, there is a real danger of confusing models of different environments and specifying a software system upon them (Génova et al. 2005; Nuseibeh and Easterbrook 2000). In Chapter 5.0, Use Cases were found to be less than adequate in retaining the richness contained within process models; the RAD was found to be simplistic enough to interface with stakeholders, yet rich enough to detail a specification with less information being lost once the system boundary is accounted for as illustrated via the xMDA method in Section 6.4 and Chapter 7.0. It is argued that the application of the xMDA method enables the real-world structure to be retained, forcing the user to consider alternate views on specification, thereby maintaining alignment via a single notation. This is challenging to imagine, but not impossible.

Computers systems are no longer data-centric entities and requirements are being placed on software engineers to produce systems which mimic human-driven behaviours, such as contract negotiations by web services (see WS-Policy, www.w3.org/Submission/WS-Policy/). It is therefore suggested to be appropriate to expect software systems to able to reflect the behaviour of business process roles and interactions; the challenge being to account for all of this, whilst accommodating architectures of the software process.

8.4.4 Object Orientation

Once produced, it is argued that the output of the xMDA method could be applied as specification to the design phase of any chosen software architecture, using any platform technology. The method has so far been verified using object oriented techniques.

In Section 6.4 and Chapter 7.0, the xMDA method was related to the MDA approach to software development, as defined by the OMG (2003b), with a view to better connecting business problems to IT solutions (Jeary et al. 2008). Transformation and Platform Information in the form of rules were derived to facilitate the generation of UML Use Case, Activity and Class models with promising results. The difficulty of applying the RAD to the MDA is that it was never intended to describe software; hence the need for Transformation and Platform Information to enable the RAD to be described in a design language. In this

case, analysis of metamodels for both the RAD and the UML provided a set of guiding rules to facilitate the transformation of models.

Whilst able to adequately account for the object oriented paradigm, it is suspected that other approaches might better account for structured techniques. The xMDA method was designed specifically with the view to output object oriented UML within the MDA framework. Many students considered that choice of method might be dependent on project and the involved software processes, or that a combination of techniques might better facilitate the maintenance of alignment in transformation since they can provide alternate viewpoints. Therefore, further investigations are required to examine the applicability of the method to procedural and combined solutions if it is to be applied in that manner. This is important in assessing the flexibility of the method in combination with others, and discovering whether it is really feasible to suggest a method of specification that discounts Use Cases.

8.5 Summary

From the analysis conducted in Appendix II and the related discussion presented in this chapter, it is clear that there are a number of difficulties in moving from analysis to specification which are reflected in the themes identified in consideration of the evidence; such difficulties are compounded in consideration of any move that might involve automatic transformations.

It was found that the students considered that methods and applied notations need to be both simple enough that stakeholders may understand and verify that requirements are being addressed through the development process, and rich enough that detail is not lost in transformation, thereby providing for an appropriate level of business/IT alignment. The RAD is rich with state based information and accommodates a sufficient description of behavioural dependencies; without modification, the Use Case does not. Therefore, the RAD is suggested to be a perfect candidate for modelling processes in analysis. The Use Case on the other hand is found to be less than adequate in describing the level of detail to ensure alignment; other techniques are required to capture rigorous process detail before moving to Use Cases and even then, much of this detail could be eventually lost.

The focus of results found and observations made in Appendix II was clearly on the POSD and the xMDA methods. This could either have been because they were best demonstrated, most useful or simplistic approaches. The RAD neglects the system boundary which was found to be an important consideration in selecting an appropriate method to move from analysis to design, since specification is about the interface between them. Only two methods make a real conscious effort to apply this notion to the RAD, the SystemRAD and the xMDA method. Of these, the demonstrated use of the SystemRAD focussed on deriving

a Use Case from the RAD and the xMDA method focussed on the development of specification via the RAD; both approaches lack practical application and experience. Complex relationships in requirements definition are important and it is vital that these relationships are carried through specification; it was found that there is a lack of richness supported in Use Case specifications and approaches such as the xMDA method could discount them due to the thoroughness of the technique.

A challenge was presented in that it is difficult to move from a process model (such as a RAD) to a procedural specification (such as a Use Case) or an object model in design whilst maintaining the alignment between models. This is suggested to be alleviated by the application of the discussed methods. Much attention was placed on the xMDA method and considerable support from the students was received in comparison with other available techniques. This could suggest that the xMDA method is viable in making the transition from analysis to design and also addresses the key considerations, such as being accessible to stakeholders, in the process. The approach is current and focussed on the RAD, thereby facilitating analysis and specification into design via a single notation. With the application of a number of steps, the user is forced into considering viewpoints and making decisions about specification. This also ensures that information is not lost and remains aligned with business ideals since specification is based directly on the analysis RAD and mechanisms are included to complete the move to design; it is built for the derivation of system models, enabling the interoperability with MDD.

However, a high percentage of students placed expectation in solutions being derived by the application of approaches in combination, according to project and process. This is perhaps the accepted reality since there is unlikely to be a *one size fits all* solution (Jackson and Zave 1993). This is supported in Jackson (1995) where it is highlighted that there is “a big temptation to believe that you can describe the application domain and the machine all together, in one combined description... But if you only make one description, you’ll surely be tempted to put things into it that describe only the machine, and to leave out things that describe only the application domain” (Jackson 1995). Alternate modelling methods can offer alternate views from alternate domains, thereby facilitating a more complete solution.

As previously mentioned, the findings made here are subjective in nature and therefore the credibility of the research is reliant on the appreciation of the participators’ perspective in an academic setting; that is the scope of the research is limited to those involved and therefore, so is the credibility. A large consideration for all methods described was the lack of practical application. Therefore, finding the xMDA method to be viable and accessible from the analysis of student feedback, there is a requirement to extend this research into a commercial setting, to which Chapter 9.0 is directed.

Chapter 9

xMDA and The Club Company

In Chapter 8.0, ideas relating to the method presented in Chapter 7.0 were verified in an academic setting, demonstrating that the method is both learnable, and applicable in moving from analysis to design. However, since the investigation focussed on individual perceptions, there was a vital need to ascertain if the method could also be practical in industry. Therefore, to complete the achievement of aim 4, the commercial validity of the described extension to the MDA framework is required to be assessed. Since there is a lack of a sufficient enterprise level test environment, the xMDA method has been applied in a commercial case study with a view of discovering whether the specification method is really feasible for application to commercial processes. In Chapter 8.0, tool support was also found to be a questionable area to which this chapter looks to address by applying the transformation rules of the xMDA method to the QVT, a specification defined and supported by the OMG for transformations within the MDA, and then using the derived QVT definitions in a tool developed in part of the VIDE initiative (VIDE 2009) to demonstrate how system models might be generated by xMDA application.

9.1 Commercial Application

The Club at Meyrick Park, Bournemouth, Dorset, is a historic golf course dating back to 1894. Nowadays, facilities that extend the golf course and clubhouse include the health and fitness gym, aerobic, holistic & spin classes, swimming pool, sauna & steam rooms, sun beds, treatments, café & bar, club shop and accommodation at *The Lodge*. Focussing on *Membership Sales*, part of this research was to produce a complete set of process models for internal training and documentation purposes, using the RAD notation. These models were agreed to then be used to verify the application of the xMDA method to demonstrate whether or not the method was practical in consideration of real business processes; the results and discussion of which are presented in this section.

Several elicitation meetings with the Membership Manager of The Club at Meyrick Park resulted in a total of eight process models being created to represent the entire *Membership Sales* process. These process models were initially employed internally by the Membership Manager to train new starters. On introduction, the Membership Manager advised that the models were simple to use and easy to understand, being “very helpful” in outlining the business processes to new starters. Beyond this initial training application, the

process models were found to be useful for the new starters, acting as a checklist to be referred to when uncertain about how certain parts of the processes ought to proceed. The models created as part of this research are currently active within the organisation and are given in Appendix III, along with a summary of discussions relating to the application of those process models within the organisation with the Membership Manager, who was directly involved with the original elicitation process in creating the process models for the complete *Membership Sales* process. It is important to note that some interacting roles are not given in the original process models at the request of the customer, since some relationships were deemed obvious and the visualisation of those relationships was found to be unnecessary. Therefore, the models contained within Appendix III are considered to be a *Sales Advisor* view of the complete process, illustrating how abstraction can be facilitated via the RAD. However, since interacting roles are vital to the xMDA method; the case study of this chapter includes the interacting roles as validated by the customer. For clarity, and to illustrate how the application of the xMDA method can facilitate abstraction by concentrating on the smaller parts of the complete enterprise process, focus here is given to one of those process models. The process model which will remain the focus of the rest of this chapter is the *Follow Up Call Process Model (05/004)* which was selected as the most complex of those available to best demonstrate the method in practice.

The *Follow Up Call Process Model (05/004)* involves the situation where a *Sales Advisor* is required to follow up on a potential member (or prospect) with the view to attaining membership to the club. Triggered by the need to follow up a prospect, the model illustrates that for each prospect to be followed up, the sales advisor must first check to see if the prospect's details have been retained on *Brightlime* (an existing prospect database). If the prospect details are not present, a new record is created. Once the details exist, the sales advisor is in the position to contact the potential client. Upon making contact, one of four routes can be taken. The prospect could be signed up as a member (this is catered for in an alternate process model - 05/003), an appointment or a return can be arranged for the prospect to visit the club (process model 05/002), an additional follow up call could be required, in which case the same process is followed (illustrated by a looping line state), or the sales advisor may blow the prospect out, by sending a letter and updating the prospect database.

This chapter continues by taking the *Follow Up Call Process Model (05/004)* and applying the xMDA method as discussed in Chapter 7.0.

But a method, to be worthy of the name, must at least decompose the development task into a number of reasonably well-defined steps which the developer can take with some confidence that they are leading to a satisfactory solution (Jackson 1982).

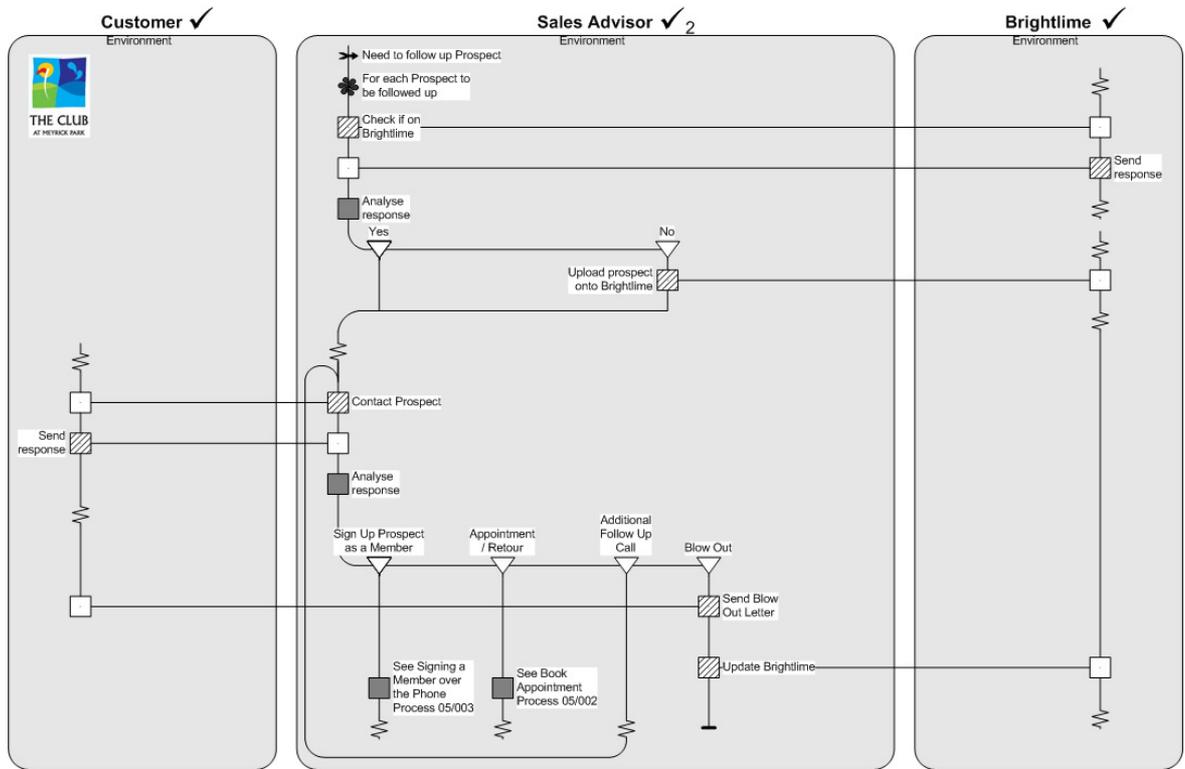
To recap, the steps of the xMDA method, detailed and illustrated in Section 6.4 and Chapter 7.0, are as follows:

-
1. Define *Environment RAD* from analysis.
 2. Derive *Shared RAD* to examine shared phenomena.
 3. De-couple environment concerns to create *Machine RAD*.
 4. Complete *Class Discovery* via *Tri-Step* analysis (optional).
 5. Apply Transformation and Platform Information to create UML representations.
 - 5.1 Use Case Diagram.
 - 5.2 Activity Diagram.
 - 5.3 Class Diagram.

The first three parts of this method involves moving from analysis through specification by focussing upon phenomena relating to the environment, that which is shared between the environment and the machine, and that of the machine via a series of RAD based model incarnations. The first being the *Environment RAD*, which represents the process in an environmental context, regardless of machine use; the second being the *Shared RAD*, which highlights the boundary between the environment process and the machine use; and lastly, the *Machine RAD*, which captures only those mechanisms of the process required to be implemented by the machine.

9.1.1 Environment RAD

As previously mentioned, the point of the *Environment RAD* is to capture the process model in terms of the environment to which it is subjected. The *Environment RAD* presumes that a degree of analysis and elicitation has already taken place (in order to create the original process model from which the *Environment RAD* is created). The *Environment RAD* for the *Follow Up Call* process is given in figure 9.1.1.1.



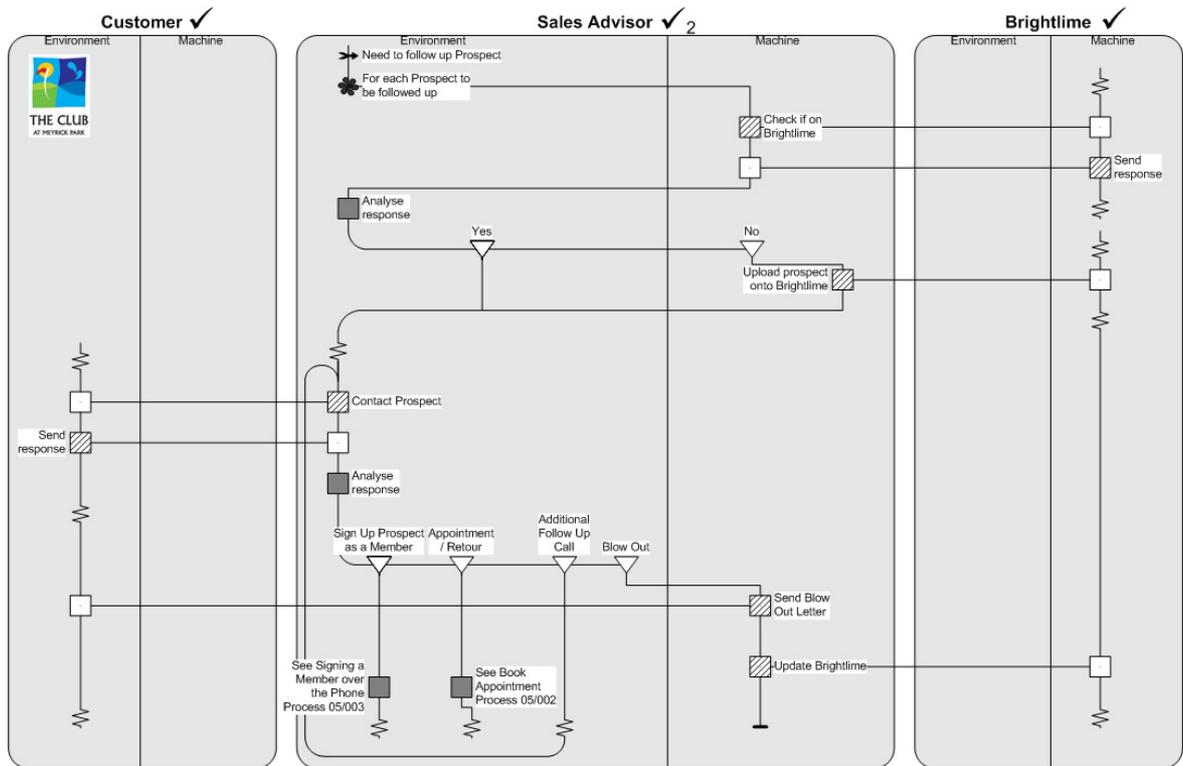
Note: Enlarged version appears in Appendix IV, 4-1.

figure 9.1.1.1, Environment RAD for the Follow Up Call process.

As can be seen in figure 9.1.1.1, the *Follow Up Call Process Model (05/004)* has been elucidated by the inclusion of the *Customer* and *Brightlime* as roles to capture the opposing sides of interactions with the *Sales Advisor* role. Further to this, along with customer elicitation, the *Check if on Brightlime* independent activity of the *Follow Up Call Process Model (05/004)* was found to actually represent an interaction between the *Sales Advisor* and *Brightlime* roles; this illustrates how the inclusion of all existing roles in a RAD can help to tease out important process semantics.

9.1.2 Shared RAD

Once the environment view has been captured in the RAD, it can be used to distinguish a phenomenon which exists, or should exist only on the machine. The approach for completing this involves interrogating each process element and making a manual, pragmatic decision based on available information as to whether such element exists in the environment of the real-world, or whether it ought to be represented by the machine. The *Shared RAD* for the *Follow Up Call* process is given in figure 9.1.2.1.



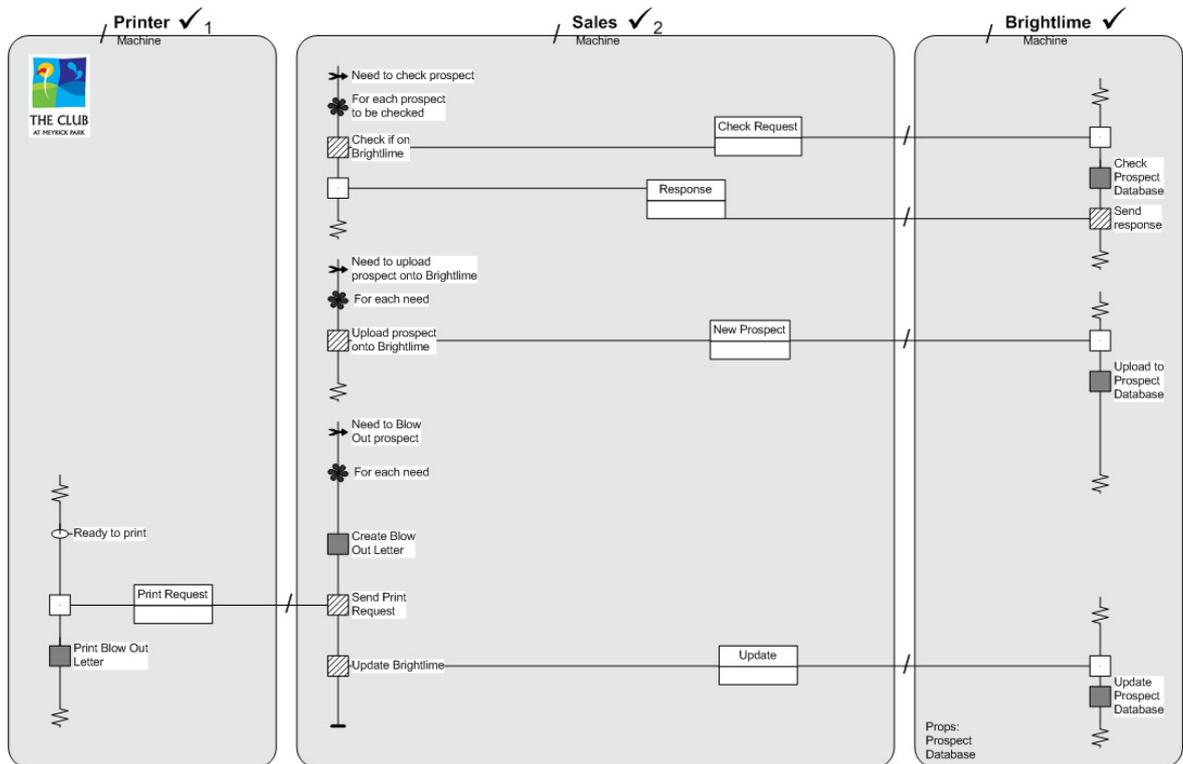
Note: Enlarged version appears in Appendix IV, figure 4-2.

figure 9.1.2.1, Shared RAD for the Follow Up Call process.

The *Shared RAD* forces the user to shift the focus from the process and operating environment to question relationships that exist between the environment and machine, and where necessary, making significant adjustments to the model. In figure 9.1.2.1, it is clear that only certain activities and roles are required to be supported by the system that is to be developed. For example, the *Blow Out Letter* is sent by the *Sales Advisor* to the *Customer*. If this was a *Machine-to-Machine* relationship, further interrogation may reveal that this communication is via e-mail. However, the relationship that exists is one of *Machine-to-Environment* (or *Environment-to-Machine*, depending on perspective) which is most interesting because further analysis can reveal how the process might move from the machine to the environment and exactly what is required of the machine for specification; as it stands, the *Customer* role has no explicit relationship with the machine and therefore more is to be discovered. *Environment-to-Environment* relationships, whilst interesting from a process management and engineering perspective, serve no purpose in this part of the xMDA method since they should have already been addressed in analysis and the *Environment RAD*. However, interrogation of these relationships can reveal significant process flaws that require addressing before moving to the next stage, iterating back to previous activity.

9.1.3 Machine RAD

In the previous section, the *Shared RAD* revealed interesting relationships between the environment and machine elements in the role context. The *Machine RAD* is directed at focussing purely on the part of the process that resides with the machine. That is, the final stage of specification, directed at completing a picture of what is required of the machine by the roles involved. The *Machine RAD* for the *Follow Up Call* process is given in figure 9.1.3.1.



Note: Enlarged version appears in Appendix IV, figure 4-3.

figure 9.1.3.1, Machine RAD for the Follow Up Call process.

Here, further elicitation has been conducted to discover the true nature of the relationships highlighted between the environment and machine by the *Shared RAD*. In the previous section, it was noted that the *Customer* role had no interaction with the machine in the reception of the *Blow Out Letter*. In much the same way as was discussed in the illustrative case study given in Chapter 7.0, further elicitation found that this letter is actually created and printed by the *Sales Advisor*, and is then sent by the *Sales Advisor* via standard mail in the operating environment. That is to say, the relationship between *Customer* and *Sales Advisor* was in fact *Environment-to-Environment*, with the discovery of another role, *Printer*, which is used to facilitate the printing of the *Blow Out Letter*, created by the *Sales Advisor*. It is agreed that an experienced Systems Analyst may have spotted such an adjustment without using the xMDA method; the point being, that the method

forces the user into making such consideration and minimises the risk of overlooking important issues, which the MDA does not address. Other significant adjustments designed to disassociate the machine from the environment include the use of triggers to enforce rules on active state which can be used to instigate a particular part of the process, should that required state be reached.

At this point, roles may be renamed to represent a generic, or more useful, terminology that may have greater application for the design phase. For example, *Sales Advisor* has been changed to *Sales*, reflecting the associated department rather than individual role. The *Printer* could have been labelled as the *Sales Printer*. Although, since more than one *Sales Printer* might materialise in the overall process, to be more specific, it could be helpful to use the printer name or resource address. However, this is down to the vision of the modeller and point of application rather than a requirement of the method; for clarity here, *Printer* will suffice. This *Machine RAD* has also been annotated with further information which will be discussed in the subsequent section.

9.1.4 Tri-Step Analysis

Once the *Machine RAD* has been created, it can be used to discover candidate design classes of classically object oriented *Entity*, *Interface* and *Control* types. This is defined by the *Tri-Step* analysis of the *Machine RAD*, which was created as part of this research and described in Section 7.5. Here, the *Tri-Step* analysis for class discovery has been applied to the *Machine RAD* given in figure 9.1.3.1, results of which are represented by annotations upon the *Machine RAD* and in the following table using the syntax provided in Cox and Phalp (2003), Cox et al. (2005b).

Relationship	Description	Class Name	Type
Check if on Brightlime	SA!{check if on Brightlime}	Check Request	<<entity>>
		Sales	<<control>>
		Brightlime	<<entity>>
		Sales / Brightlime	<<interface>>
Send Response	BR!{send response}	Response	<<entity>>
		Brightlime	<<control>>
		Sales	<<entity>>
		Brightlime/Sales	<<interface>>
Upload Prospect	SA!{upload prospect}	New Prospect	<<entity>>
		Sales	<<control>>
		Brightlime	<<entity>>
		Sales / Brightlime	<<interface>>
Print Blow Out Letter	SA!{print blow out letter }	Print Request	<<entity>>
		Sales	<<control>>
		Printer	<<entity>>
		Sales / Printer	<<interface>>
Update Brightlime	SA!{update Brightlime}	Update	<<entity>>
		Sales	<<control>>
		Brightlime	<<entity>>
		Sales / Brightlime	<<interface>>
Brightlime to Prospect Database	BR!{access}	Prospect Database	<<entity>>
Brightlime to Prospect Database	BR!{access}	Prospect Database	<<control>>
Brightlime to Prospect Database	BR!{access}	Prospect Database	<<interface>>
Printer to Machine	PR!{access}	PrinterGUI	<<interface>>
Sales to Machine	SA!{access}	SalesGUI	<<interface>>
Brightlime to Machine	BR!{access}	BrightlimeGUI	<<interface>>

table 9.1.4.1, potential design classes derived from a Tri-Step analysis of the Machine RAD for the Follow Up Call process.

From analysing the *Machine RAD* in terms of *Entity*, *Interface* and *Control*, potential design classes have been uncovered. For example, from examining the *Machine-to-Machine* relationship of *Print Blow Out Letter* between the *Sales* and *Printer* roles, a *data-interaction* reveals the *Print Request* (*Print Request <<entity>>*) class. This is illustrated in figure 9.1.3.1 with a UML Class sitting on the interaction line. Candidate *Interface* classes are denoted by the / symbol in figure 9.1.3.1 and were discovered by looking at the connection between the roles in three ways. Firstly, the connection between the role and the machine provides graphical user interfaces for the roles to access, e.g. *SalesGUI* (*SalesGUI <<interface>>*). Secondly, the connection between the roles and any props they access revealed that *Brightlime* is, or may be required, to interface with the *Prospect Database* prop (*Prospect Database <<interface>>*). Thirdly, the connection between the roles themselves revealed that interacting roles may be likely to be required to interface via some common medium, for example the *Sales* class maybe be required to go through some interface (*Sales / Brightlime*

<<interface>>) to connect to the *Brightlime* system (which may be on an alternate platform). Since a *Tri-Step* analysis considers all relationships, it is possible to also distinguish further information about the direction of control in such relationships. For example, by examining the interaction *Print Blow Out Letter*, the *Sales* role can be seen to drive the interaction and manage the communication between the *Sales / Printer* <<interface>> and the *Printer* <<entity>> via the notation, and thus, the *Sales* <<control>> class is discovered; this is shown graphically in figure 9.1.3.1 by the standard RAD notation for identifying driving roles in interactions (square box with right-sided upwards diagonal shading). As previously noted, each individual role is responsible for controlling any internal logic flow; external control is a concern is beyond the context of the machine.

9.1.5 Transformation and Platform Information

The benefit of what has been discovered so far is that a specification has been developed from modelling the business process in terms of the business user in a language that they can understand and validate. This is suggested to help ensure business involvement in defining software systems to suit the business need. The next part of the xMDA method involves taking Transformation and Platform Information and applying it to the *Machine RAD* to output models useful in design (PIM), and to which the potential design classes discovered from the *Tri-Step* analysis complement. In this case, Transformation and Platform Information in the form of the transformation rules (introduced in Section 7.6.3) which support the *SimpleRAD* and *SimpleUML* metamodels have been applied to create Use Case, Activity and Class diagrams from the *Machine RAD* that resulted in Section 9.1.3.

9.1.5.1 Use Case Diagram

As noted previously in Section 7.6.3, the *Machine RAD* replaces the need to move to a Use Case specification. However, transformation is supported specifically for users wishing to augment documentation with the definition, alleviating concern raised in Chapter 8.0 of defining a method that does not incorporate the Use Case and demonstrating the flexibility of the xMDA framework. Rather than taking each rule and creating a literal translation of the RAD into the UML Use Case, interactive mechanisms of the method are invoked to best demonstrate how the user might influence the transition from the RAD to the UML in the construction of the UML Use Case Diagram. The Use Case Diagram for the *Follow Up Call* process is given in figure 9.1.5.1.1.

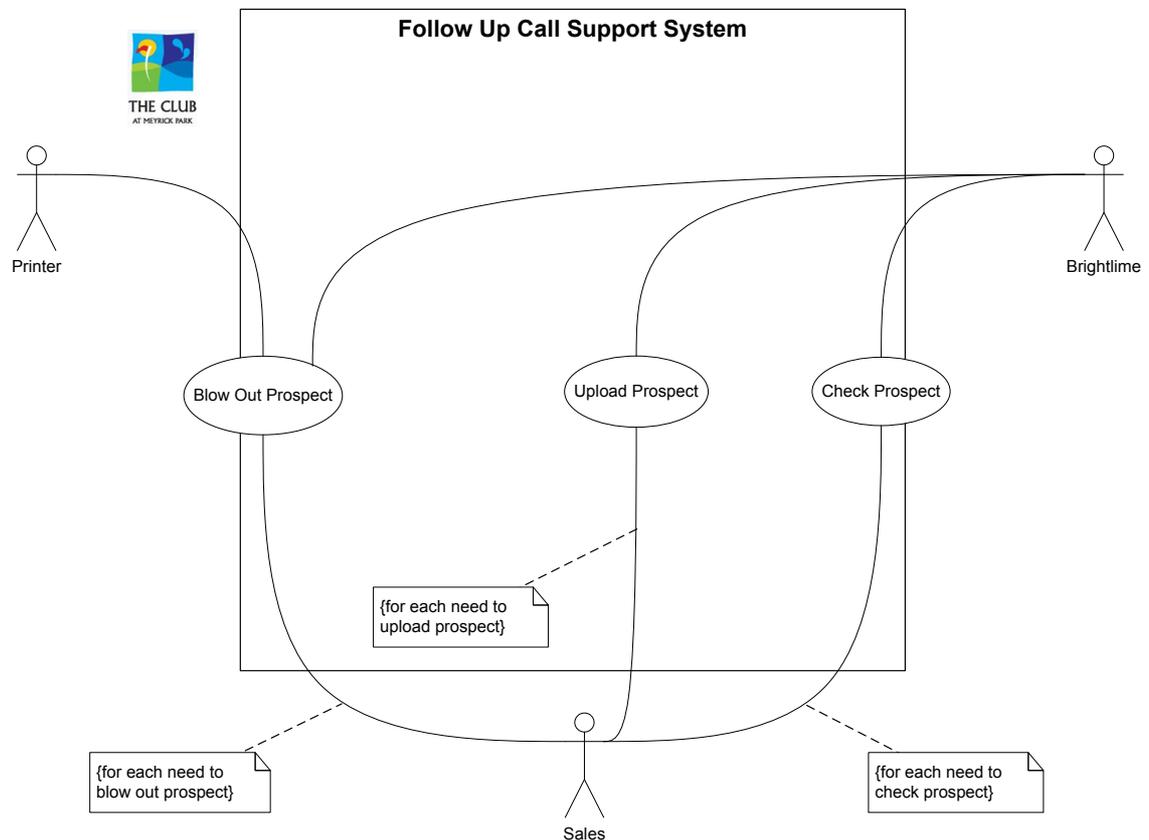


figure 9.1.5.1.1, UML Use Case Diagram for the Follow Up Call process Machine RAD.

As can be seen in figure 9.1.5.1.1, roles have translated directly into actors and relationships with Use Cases associated with those actors, triggers and replication directly into notes, with three central Use Cases being identified, that of *Blow Out Prospect*, *Upload Prospect* and *Check Prospect*. These Use Cases have been influenced directly by the rule that makes the allowance for a chunk of activity defining a single Use Case. For example, *Blow Out Prospect* could have been described by directly mapping numerous, more detailed, Use Cases including *Create Blow Out Letter*, *Send Print Request*, *Update Brightlime* involving the *include* or *extend* relationships between the Use Cases and those relating to the *Sales* and *Brightlime* actors. Relationships that are defined relate to a coupling of all RAD interactions involved in the particular Use Case.

A degree of freedom is useful in translating from the RAD to the UML Use Case. Some rules, such as recording the number of role instances (✓) within the Use Case were discarded by choice, others were not experienced (for example, those relating to refinements). Such freedom may lead to error prone specifications

and therefore it could be possible to hard-code these rules into a software application that could complete the transformation without user intervention; however, it is important that the benefit of user involvement in such model generation is not lost, since the beauty of these diagrams is in the facilitation of such interaction.

9.1.5.2 Activity Diagram

As identified in the previous section, there appears to be three central chunks of activity pertaining to the *Follow Up Call* process. When transforming into the UML Activity Diagram from the RAD, a note is put in the rule relating to the undefined element, directing the user to check the context of application, since the transformation may involve an alternate Activity Diagram. Because the RAD in question utilises such undefined elements to describe *hanging threads*, the sequential flow of activity is impossible to derive for the complete process. This is because, as discussed in Chapter 5.1.1.8, the nature of the RAD permits process definition with a closer alignment to the reality of the business context; i.e. non-sequential. The fact is, *Check Prospect* is not immediately required to be followed by *Upload Prospect*, or *Blow Out Prospect*. The three Use Cases are autonomous, and can be used alone or in combination with others, provided the identified state is reached. Therefore, three Activity Diagrams are required to capture the *Follow Up Call* process and are demonstrated in the subsequent sections.

9.1.5.2.1 Check Prospect

Figure 9.1.5.2.1.1 illustrates how the *Check Prospect* part of the *Follow Up Call* process might translate from the *Machine RAD* into the notation of the UML Activity Diagram via the application of the transformation rules provided.

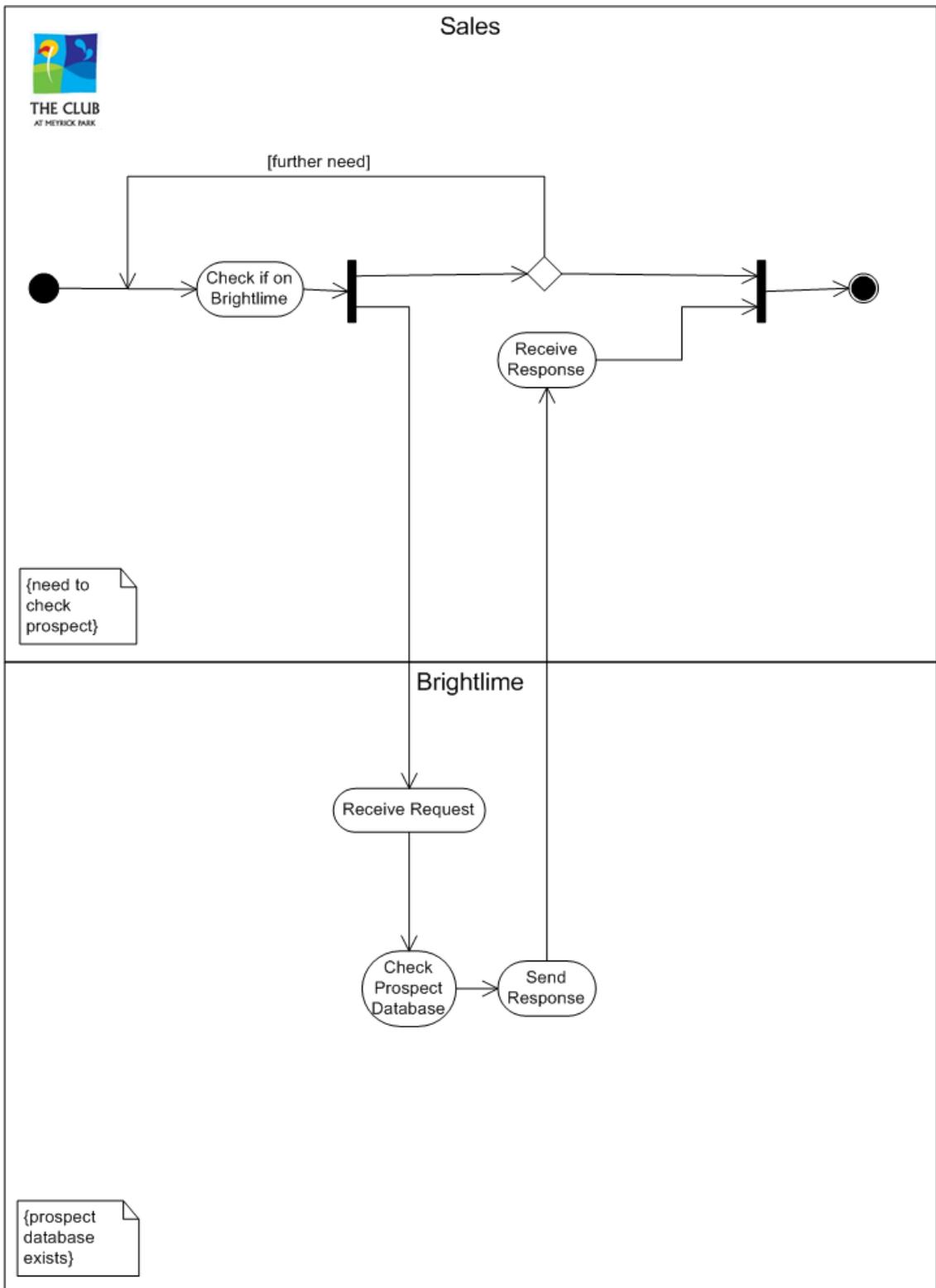


figure 9.1.5.2.1.1, UML Activity Diagram for the Check Prospect part of the Follow Up Call process Machine RAD.

As can be seen, RAD roles have directly translated into Activity Diagram partitions that contain the flow of activity relating to those roles; independent activities have formed activities; triggers and props have formed notes to guide the understanding of the process; replication has formed a decision diamond with a guard and transition loop, causing further activity to be extended via the synchronisation bar (until all activity for each loop is completed). This is important part that is captured by the xMDA because it allows the *Sales* members to continue facilitating additional requests whilst the process continues (as would occur in reality). For further discussion on this, please refer to section 9.1.5.2.3. Start and stop points are mandatory for the Activity Diagram notation and positioning ought to be defined by the user. However, they can also be defined by the Transformation and Platform Information. Here, the start point is defined by the trigger in the RAD and the stop point is defined from evaluating the undefined element at the end of the *Check Prospect* part of the *Machine RAD*.

It is important to remember that although the RAD is becoming sequenced to some degree via the transformation, it is not completed without the informed advice of the user and attempts are made to mirror the state based transitions confined within the RAD. Further elicitation may also be required to fill in some gaps should the transformation be short of information. For example, the activity *Receive Request* is not explicit in the *Machine RAD* and therefore, the involvement of *Brightlime* in the interaction involving the relationship of *Check if on Brightlime* was until now unknown.

9.1.5.2.2 Upload Prospect

The next Activity Diagram has been generated from the application of Transformation and Platform Information to the *Upload Prospect* part of the *Follow Up Call* process and is given in figure 9.1.5.2.2.1 below.

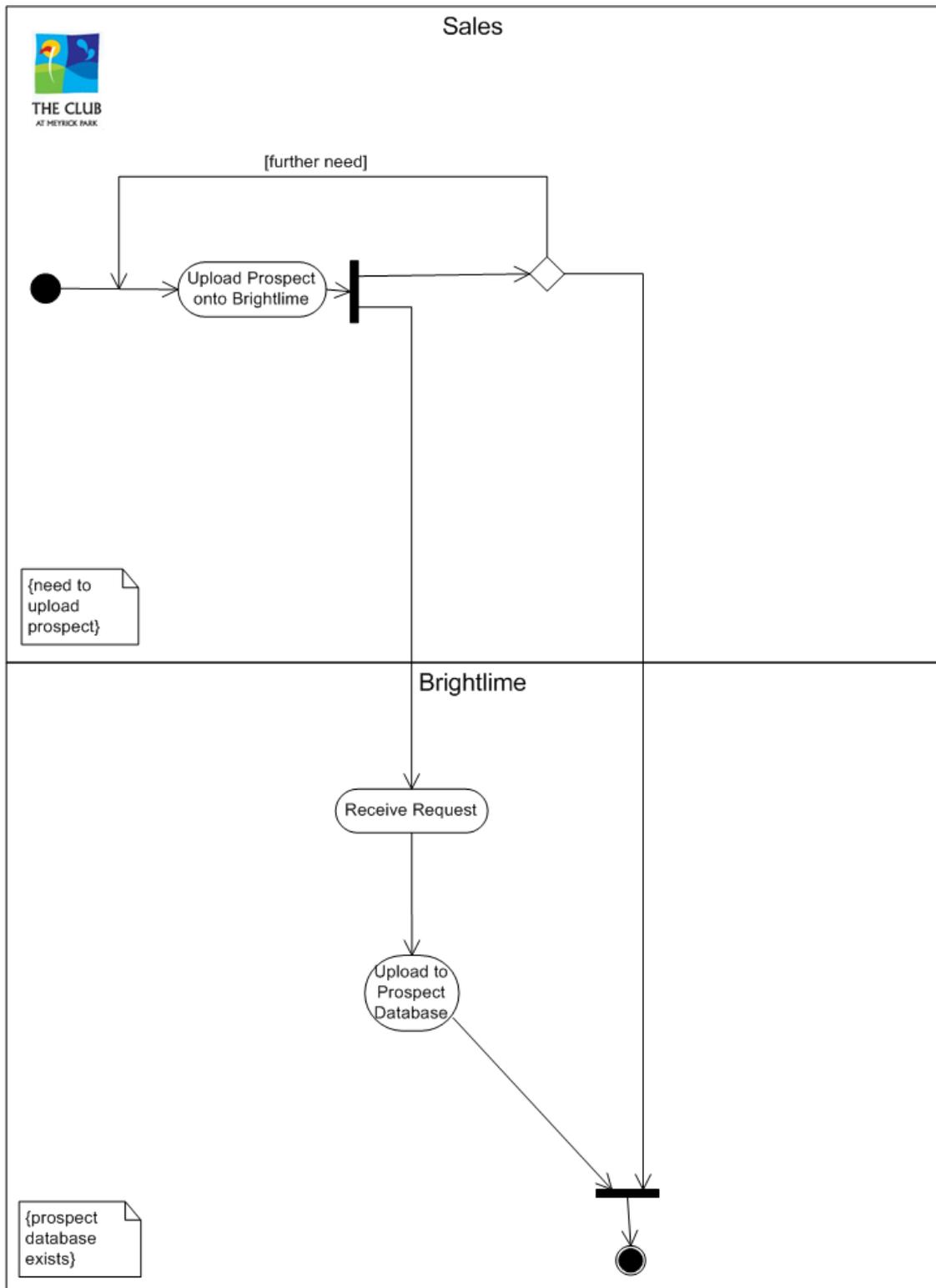


figure 9.1.5.2.2.1, UML Activity Diagram for the Upload Prospect part of the Follow Up Call process Machine RAD.

As with the previous discussion, the application applies rules that transform the *Machine RAD* into the UML Activity Diagram. Again, the start node is defined by the trigger node and the stop node by evaluating the undefined node that ends the *Upload Prospect* part of the process.

9.1.5.2.3 Blow Out Prospect

The final Activity Diagram defined to illustrate the *Follow Up Call* process is related to the *Blow Out Prospect* part of the *Machine RAD* and is given below in figure 9.1.5.2.3.1.

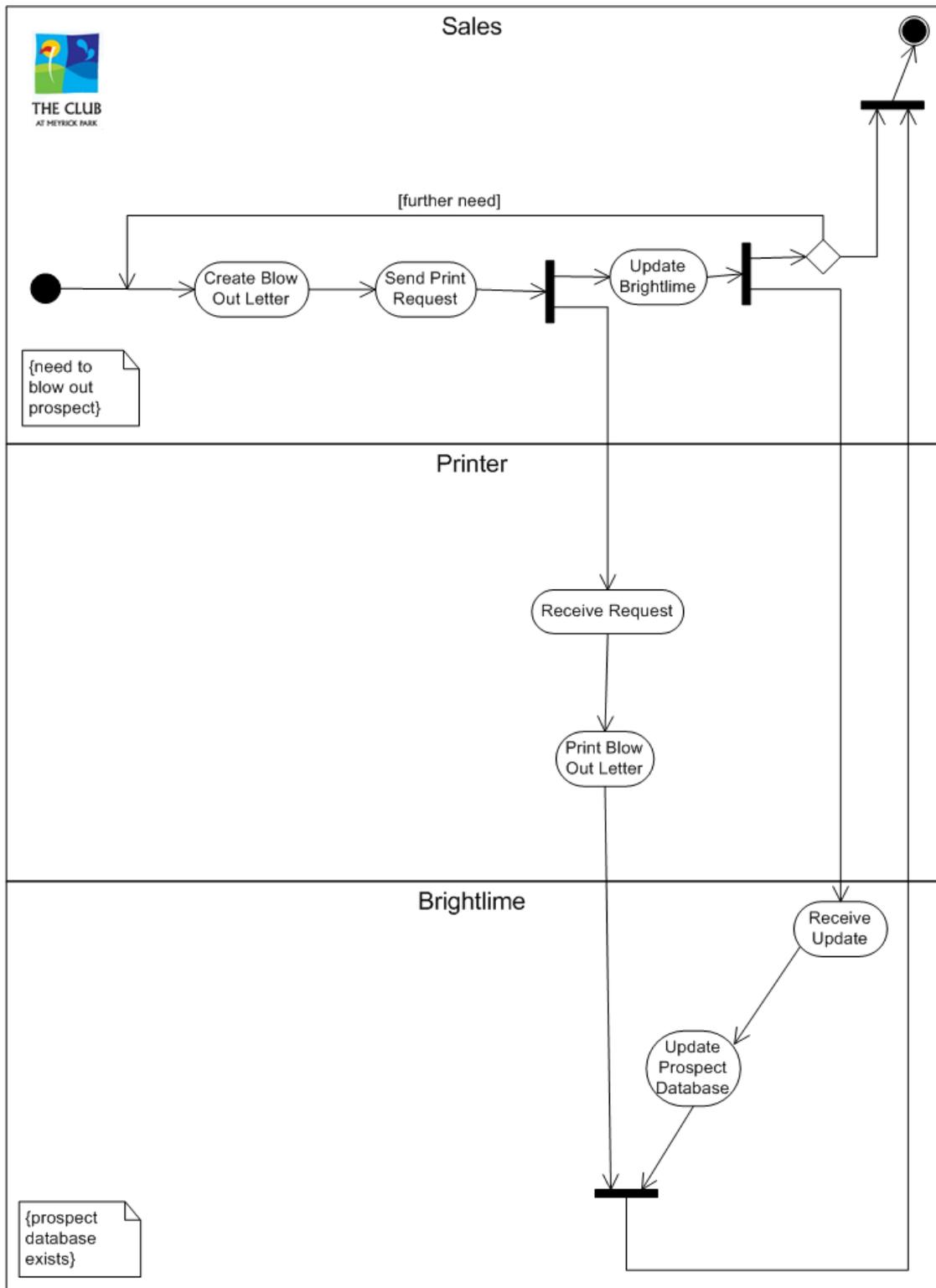


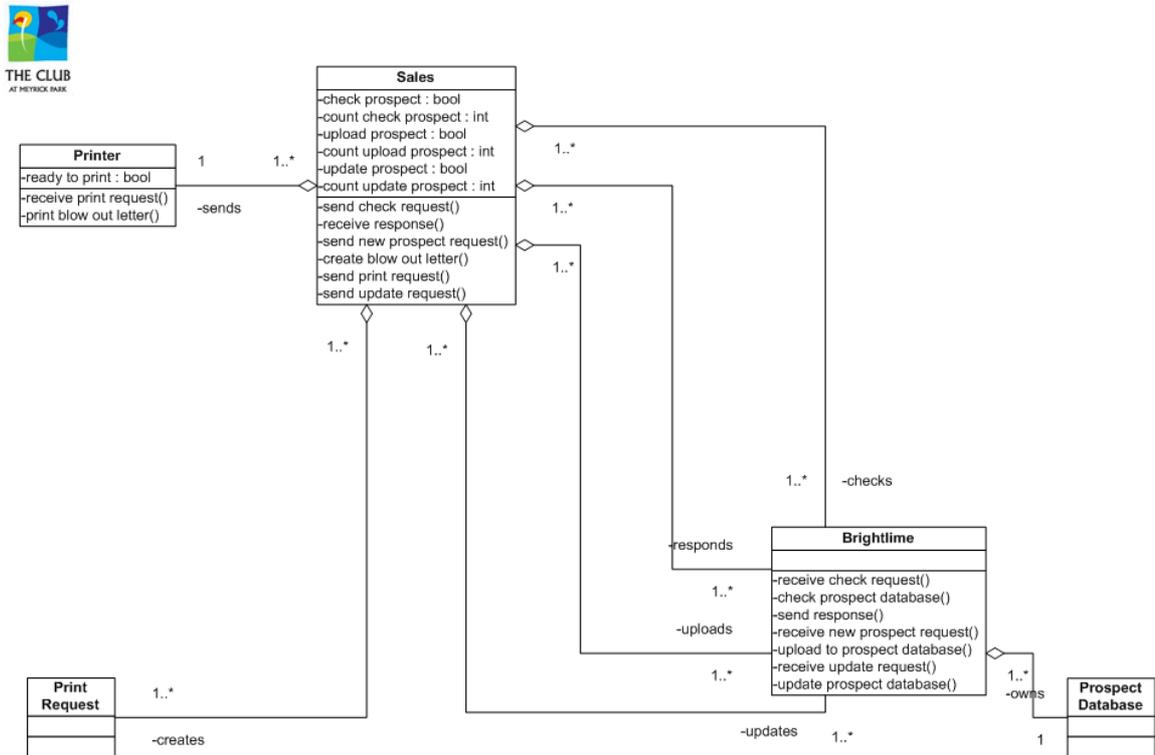
figure 9.1.5.2.3.1, UML Activity Diagram for the Blow Out Prospect part of the Follow Up Call process Machine RAD.

As can be seen in figure 9.1.5.2.3.1, this Activity Diagram is more elaborate and contains three partitions. As with the example of figure 9.1.5.2.1.1, the decision diamond is inserted here to extend the activity in the partition to allow *Sales* members to continue facilitating additional requests whilst the process continues, which mirrors the reality of the situation. Because of the nature of this process, and the relationship of the Platform Information, synchronisation bars are required to ensure the process does not terminate until both the *Print Request* is handled by the *Printer*, and the *Update* is handled by *Brightlime*. It is important to note that this application is of direct reflection of the RAD Transformation Information which describes that, with relation to looping, line and descriptor states, when there is a split of activity flow, a synchronisation bar should be added. This is due to the knowledge of the platform, i.e. the UML Activity Diagram.

The application of some rules, such as those involving refinements (although they follow a similar function as applying the decision diamond and guard, or the synchronisation bar), are not demonstrated here since they were not required by the case in question, but are detailed in Chapter 7.0. Other rules have been disregarded at will, in accordance with the transformation rules. For example, it was not deemed necessary to represent ✓ (showing the number of available instances of each activity partition via notes) on the Activity Diagrams. Such a rule could be hard-coded or represented as a user definable choice within a software implementation, if required.

9.1.5.3 Class Diagram

Perhaps the most powerful of diagrams to be generated for use within the MDA would be the UML Class Diagram representation of the *Machine RAD*. In figure 9.1.5.3.1, the Class Diagram generated for the *Follow Up Call* process is given.



Note: Enlarged version appears in Appendix IV, figure 4-4.

figure 9.1.5.3.1, UML Class Diagram for the Follow Up Call process Machine RAD.

As shown in figure 9.1.5.3.1, the application of the transformation rules has generated a Class Diagram with five individual classes, three of them relating specifically to the roles involved in the *Machine RAD*. The remaining two, *Print Request* and *Prospect Database* have been derived from examining the *Create Blow Out Letter* independent activity and *Prospect Database* prop, where data objects are considered to be created, or to previously already exist, within the process. All other independent activities and interactions are transformed into operations of the class generated from the role within which the independent activities or interactions exist. Attributes account for the descriptor states of the *Machine RAD*, such as the *ready to print: bool* attribute belonging to the *Printer*. Other attributes are derived from triggers and replicated activities that exist within the role from which a class has been generated to contain such attributes in order that these objects retain the notion of state and can reflect upon that information in enabling or disabling such operations at design-time. This notion of containment is also demonstrative of how such a method may carry ideals from the real-world and the RAD into design, in that each object retains such information which is not visible to other objects. Relationships between classes are driven mainly from interacting roles, which are embellished with names that reflect the association (for example, the *Sales* member checks *Brighttime*). Multiplicity can be derived from the ✓ that is associated with each role (for example, since only one instance of *Printer* is defined on the *Machine RAD*, only one instance may ever exist on the Class Diagram, and therefore

multiplicity is one). Aggregation is said to exist when role interactions are exclusive only to one other role. For example, since *Sales* is the only entity to interact with the *Printer*, it is said to be an aggregate of *Sales*. Further classes could be added by examining the *Tri-Step* analysis conducted in 9.1.4, but rather this is considered to be a design-time benefit.

As with previous examples, some rules were not demonstrated in the construction of the Class Diagram as they were not required for the case study (for example, role instantiation nodes would generate new classes, associations and operations in much the same way as has already described). Other rules were deemed unnecessary, such as the enforcement of an attribute to represent the RAD stop node. Of course, a software implementation would be required to account for any such rule refinement or enforcement.

9.1.6 Discussion

So far, the commercial application of the xMDA method has been described via manual application to the commercial case study. Whilst this is useful in showing that the xMDA method can accommodate business processes, it is only half of the story. The aim of the xMDA method is to both support the xMDA framework and provide a better method for defining requirements and specification within practical application to the MDA. In this section, discussion relating to the manual application of the xMDA method is given and evidence drawn upon to reveal what this method uncovered which was not already known before application and that which may not have been uncovered by other approaches.

The xMDA method suggests the starting point to be a RAD process model, output from analysis. Since this is the case, a degree of elicitation took place in creating the *Follow Up Call Process Model (05/004)*. This involved defining the RAD and iterating with the customer where necessary to ensure that the model represented the required process behaviour. Therefore, the method ensures a starting point from which many initial requirements and process flaws are already addressed. This shows that the business user can adequately define, and perhaps more importantly, verify process definitions using the method.

During the creation and evaluation of the *Environment RAD*, interactions revealed that the process involved *Customer* and *Brightlime* roles. These roles were not explicitly shown on the customer created *Follow Up Call Process Model (05/004)*. With the inclusion of those roles in the *Environment RAD*, a process level error was revealed in that one independent activity of the *Sales Advisor* actually represented an interaction with the *Brightlime* role. This demonstrates the strength of using the RAD and the purpose of the inclusion of interactions and roles. Since the xMDA method proposes the RAD with role discovery in capturing interactions, these errors and business process concerns are reduced at an early stage.

The *Shared RAD* was found to be important in establishing shared phenomena for investigation. It is here where process activities are first considered to be either related to the environment or the machine, allowing the customer to redefine the process accordingly. For example, all relationships involving the *Customer* role were found to be in the *Environment*. Therefore, it is immediately obvious that the *Customer* plays a limited role in defining the machine. However, since one interaction (*Send Blow Out Letter*) was found to be *Environment-to-Machine*, further consideration was required to discover what exactly this interaction involved. It is not so much that other methods might not make such a consideration; the xMDA method explicitly guides the user into considering relationships in terms of domain (environment, shared or machine), and making decisions regarding them - hence, the importance of specification is highlighted within the technique. The method progressively moves specification closer to the machine and in the final RAD incarnation, the *Machine RAD*, machine elements were interrogated with a view to defining only those activities required of the machine. As previously discussed, closer examination of the *Environment-to-Machine* interaction between the *Sales Advisor* and *Customer* was required. This revealed a relationship that actually represented one of *Environment-to-Environment*, where the *Sales Advisor* would post a printed letter directly to the customer; thus the discovery of a new role (*Printer*) and a *Machine-to-Machine* relationship between it and the *Sales Advisor*. Again, because the method explicitly requires the interrogation of relationships between *Environment* and *Machine* domains, and the redefinition of the specification in the *Machine RAD*, it is suggested to be a better way of defining specification and ensuring requirements are accounted for within it.

Since the *Machine RAD* represents the domain of the machine, it is suggested to be able to connect directly to design. That is, design classes can be derived (as demonstrated by the *Tri-Step* analysis conducted in Section 9.1.4) and models native to software development can be generated (as demonstrated in Section 9.1.5). This process is unique to the xMDA method and suggested to benefit the software designer, and the MDA, in that the output specification is in a language that is not only common to the designer (the UML) but also portable to software application for use in design in the case that such models could be automatically generated via a software solution. The flexibility of the xMDA framework is also demonstrated in producing models of alternative model types as output.

It is considered that, not only is the xMDA method applicable to commercial processes, it is also accessible to the business user, incorporating specification in a method designed specifically to output models useful to MDA designers, which supports the findings of Chapter 8.0, where the xMDA method was received positively in the context of other available techniques. Previous chapters have already highlighted that the CIM has no clear constitution, nor does it provide any explicit mechanism to support specification. It is therefore suggested in consideration of the findings made so far from the commercial application included within this section that the xMDA method supports a better application to embedding a requirements definition within the CIM. The method is rich enough to describe real commercial processes and forces the

discovery of potential issues for resolution, resulting in a specification output that can interface with the MDA. It is clear that, even in the examples that have been discussed, there could be real benefit in codifying these rules in a way that demonstrates how rigorous and how applicable the method is to the MDA. Therefore, the next obvious step to which the remainder of this chapter draws attention to is to discover if the xMDA method is supportive in terms of MDA tools and techniques. The Class Diagram created in figure 9.1.5.3.1 will form the basis of further analysis to look at the codification and representation of the Transformation and Platform Information within a software context to demonstrate the potential of the xMDA framework and method to the MDA.

9.2 QVT Application

In order that the xMDA method be verified beyond the manual application to the commercial case study provided in Section 9.1, this section draws upon the transformation rules defined in table 7.6.2.1 (and repeated in table 9.2.1 for reference) to define transformation relations by applying techniques of the MDA. Again, since the class diagram is considered the most useful PIM level diagram for code generation within the MDA (OMG 2003b), and to follow on from previous findings, the class diagram has been deliberately selected for the focus of the evaluation in this section.

RAD	UML Class Diagram
Role	Class
Independent Activity	Operation; Class & Association if Object results from Activity
Looping, Line and Descriptor States	Attribute (only applicable for Descriptor States)
Case Refinement (Alternatives)	Attribute
Part Refinement (Concurrency)	Attribute; Composition if refinement objects are descendent from resulting object
Interaction	Association; Operations in Role Classes; Aggregation if Role interactions are exclusive to only a single Role
Role Instantiation	Role Class; Operation in Source Role Class; and Association
Trigger	Attribute
Replication	Count attribute
Undefined	Check context; May define alternate Class Diagram
✓	Multiplicity
Prop	Class; Association with Role Class
Stop	Attribute in originating Role Class
Note	Note

table 9.2.1, initial transformation rules to map from the RAD to the UML Class Diagram.

As noted in Section 7.6.1, the standard for administering MDA transformations is the QVT (Bureck 2009; FT 2007; Giandini et al. 2009; Kusel et al. 2009). The QVT requirement for a transformation definition language

was for one that is declarative (Ignjatovic 2006). There are two declarative sub-languages defined in the QVT specification (Bureck 2009; Dan 2010; FT 2007; Giandini et al. 2009; Kusel et al. 2009; OMG 2008b). They are the *Relations* and *Core*. The Core language complements the Relations language and is used to describe “the operational semantics of the QVT Relations language” (Kusel et al. 2009) at a low level, supported by FT (2007). Focus here is therefore given to the declarative Relations language (QVT-R), which is provided in the specification defined by the OMG (2008b) and described as the “end-user view of QVT” (Ignjatovic 2006), where rules can be defined to relate metamodel elements to each other at a higher level. It is not the intention of this demonstration to deliver the extent to which QVT might be used to model these transformation rules, or the extent of the language itself; only that an implementation of these rules via QVT is possible.

In Appendix V, a complete examination of each transformation rule is provided in the context of the QVT-R language, with examples being defined in both graphical and textual QVT-R forms as described by the OMG (2008b) and discussion being presented where necessary. This chapter focuses only on those rules that are extended upon in Section 9.3, for clarity of understanding with the objective of providing a QVT definition and eventual software implementation directed at enhancing the user experience. They are: *Role2Class* (r, c); *IndependentActivity2Operation* (ia, o); *Interaction2Operation* (i, o); and *Prop2Class* (p, c).

9.2.1 Transformation Declaration

A transformation declaration in QVT-R is given in the following textual form (this example defines a transformation from a RAD model to a UML model, conforming to the *SimpleRAD* and *SimpleUML* metamodels described in Section 7.6.2).

```
transformation rad2umlcd (rad : SimpleRAD, umlcd : SimpleUML)
  {
  }
```

For the scope of this research, all transformations are required to be executed in the direction of the UML since the Transformation Information is provided to apply to the Platform Information, and not vice versa. The remainder of this chapter outlines each of the four identified rules for transforming elements of the RAD into those relating to the UML Class Diagram.

9.2.2 Role2Class

The first rule describes an unconditional mapping of a RAD *role* to a UML *class* as illustrated in figure 9.2.2.1.

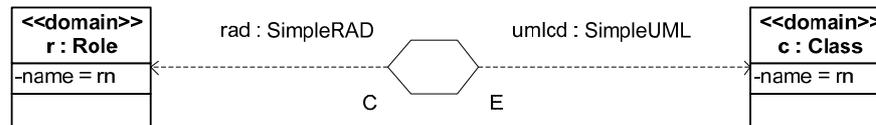


figure 9.2.2.1, a RAD role to a UML class relation.

Most cases will be required to hold true when the *Role2Class* relation holds between the role and the class containing the activity. Since this is the case, the *Role2Class* relation is defined as a top-level relation, requiring to be held true for *all* relations in a transformation. This relation is demonstrated in QVT-R textual syntax below.

```

top relation Role2Class /* map each role to a class*/
rn = 'String';
  {
    checkonly domain rad r:Role
      {
        name = rn
      }
    enforce domain umlcd c:Class
      {
        name = rn
      }
  }

```

That is, pattern *r* binds the variable *rn* to the role model element *name* and pattern *c* binds the same variable (*rn*) to the class model element *name*, resulting in both *name* model elements in the role and class elements of the *rad* and *umlcd* candidate models containing the same information, i.e. *rn*. Each domain is annotated as *Checkonly* (for verification against the rules) and *Enforce* (to create or change the target model according to the relation – that is from the RAD to the UML).

There is no complication involved in defining *Role2Class* transformation; only a single attribute (*name*) is required in the relation. *When* and *Where* clauses can also be applied to “explicitly constrain the relation” (Ignjatovic 2006) and may be formed using “arbitrary OCL expressions in addition to... relation invocation expressions” (OMG 2008b). However, no such expressions are required to further elucidate understanding of this rule.

9.2.3 IndependentActivity2Operation

Since the *Role2Class* relation has been defined as a top top-level relation, the *IndependentActivity2Operation* relation is required to hold only when the *Role2Class* relation holds between the role containing the independent activity and the class containing the operation. This relation is illustrated in figure 9.2.3.1.

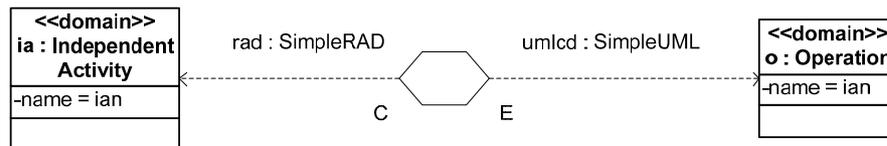


figure 9.2.3.1, a RAD independent activity to a UML operation relation.

The QVT-R that represents this relation is not defined as a top level relation, as it is not a requirement for all other relations, since the transformation is dependent on only the involved independent activity and associated operation. This is reflected in the following description.

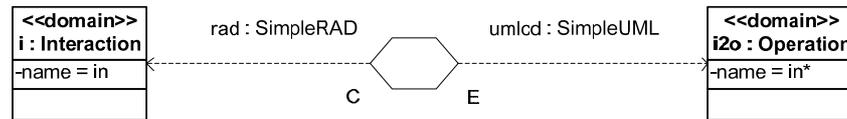
```

relation IndependentActivity2Operation /*map each independent activity to an operation*/
ian = 'String';
  {
    checkonly domain rad ia:IndependentActivity
      {
        name = ian
      }
    enforce domain umlcd o:Operation
      {
        name = ian
      }
  }

```

9.2.4 Interaction2Operation

This rule suggests that for each RAD interaction, the UML Class Diagram will exhibit operations for both driving and passive RAD interaction nodes within classes defined by the top relation (*Role2Class*). Therefore, the following relation is required, described visually in figure 9.2.4.1, with the textual counterpart after.



— where —

Interaction2Association (i, i2a)

figure 9.2.4.1, a RAD interaction to a UML operation relation.

relation Interaction2Operation /*map each interaction to an operation*/

in = 'String';

{

checkonly domain rad i:Interaction

{

name = in

}

enforce domain umlcd i2o:Operation

{

name = in* /*must be amended to reflect transitive verb and direct object (noun)*/

where

{

Interaction2Association (i, i2a);

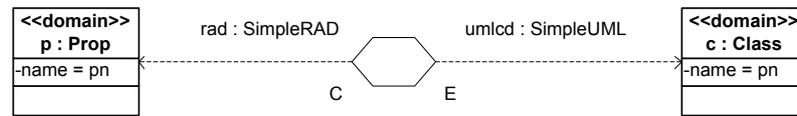
}

}

When the relation is true, the *Where* clause is used to apply further conditions. It is used in this context to describe that whenever the *Interaction2Operation* relation holds, the *Interaction2Association* relation must also hold. Further description of this relation is given in Appendix V.

9.2.5 Prop2Class

The *Prop2Class* relation is defined to show how, for each RAD prop used by a role, a new UML class is enforced to represent that prop. Typically, such a class would represent a data object that is manipulated by the class generated by the top relation *Role2Class*. The graphical syntax for this relation is given in figure 9.2.5.1, with the textual description following it.



— where —

Prop2Association (p, p2a)

figure 9.2.5.1, a RAD prop to a UML class relation.

relation Prop2Class /*map each prop to a class*/

pn = 'String';

```

{
    checkonly domain rad p:Prop
    {
        name = pn
    }
    enforce domain umlcd c:Class
    {
        name = pn
    }
    where
    {
        Prop2Association (p, p2a);
    }
}

```

Where this relation holds, a relation to create an association between the new class and the class derived from the top relation *Role2Class* must also hold. Therefore, the *Prop2Association* relation is required. See Appendix V for a description of this relation.

As can be seen in the examples given in the figures of this chapter, the graphical syntax provides an *expressive* and *easy* way to define transformations in comparison with the textual syntax (Bureck 2009). However, some difficulties are encountered in that there is no real QVT-R graphical software modelling environment available. Further to this, QVT-R Engines such as Eclipse QVT Declarative, Medini QVT, ModelMorf and MOMENT-QVT are still in the development stages (Bureck 2009; Kusel et al. 2009). The syntax itself is also not yet mature; some elements have no graphical syntax (for example, top level relations and queries), some elements are unclear in meaning, and some elements have no textual counterpart (Bureck 2009).

9.2.6 rad2umlcd QVT-Relations

A summary of the complete set of relations defined by this research in moving from a RAD in the direction of a UML Class Diagram is given in table 9.2.6.1, and detailed in Appendix V.

RAD	UML Class Diagram	Relation
Role	Class	Role2Class (r, c)
Independent Activity	Operation; Class & Association if Object results from Activity	IndependentActivity2Operation (ia, o); IndependentActivity2Class (ia, ia2c); IndependentActivity2Association (ia, ia2a)
Looping, Line and Descriptor States	Attribute (only applicable for Descriptor States)	DescriptorState2Attribute (ds, at)
Case Refinement (Alternatives)	Attribute	CaseRefinement2Attribute (cr, at); CaseRefinementElement2Attribute (cre, at)
Part Refinement (Concurrency)	Attribute; Composition if refinement objects are descendent from resulting object	PartRefinement2Attribute (pr, at); PartRefinementElement2Attribute (pre, at); PartRefinement2Composition (pr, pr2a)
Interaction	Association; Operations in Role Classes; Aggregation if Role interactions are exclusive to only a single Role	Interaction2Association (i, i2a); Interaction2Operation (i, i2o); Interaction2Aggregation (i, i2a)
Role Instantiation	Role Class; Operation in Source Role Class; and Association	RoleInstantiation2Class (ri, ri2c); RoleInstantiation2Operation (ri, ri2o); RoleInstantiation2Association (ri, ri2a)
Trigger	Attribute	Trigger2Attribute (t, at)
Replication	Count attribute	Replication2Attribute (re, at)
Undefined	Check context; May define alternate Class Diagram	N/A
✓	Multiplicity	✓2Multiplicity (ti, mu)
Prop	Class; Association with Role Class	Prop2Class (p, c); Prop2Association (p, p2a)
Stop	Attribute in originating Role Class	Stop2Attribute (s, at)
Note	Note	Note2UMLNote (n, umln)

table 9.2.6.1, complete set of QVT-Relations defined by the rad2umlcd transformation.

It is important to recognise that whilst the possibility of definition and transformation has been demonstrated, elements, such as looping and line states and the undefined element, have no Class Diagram representation. And vice versa, the Class Diagram notation also includes elements (for example, packages) that have no RAD representation, and therefore do not feature here in the description of relations. The rules defined as part of

this research to enable the transformation of models conforming to the *SimpleRAD* metamodel into UML Use Case and Activity diagrams (see Section 7.6.3) may facilitate a better understanding and treatment of objects. However, it is accepted that no matter how many representations are produced, each will represent only a particular viewpoint on that RAD. The usefulness of the xMDA method is in providing a CIM to the PIM, founded on the knowledge of requirements and specification, defined by business, and presented in a language akin to the software user. Further research may demonstrate that rules could be applicable to both directions (a promising example of this was highlighted in the *Interoperability* section of Appendix II), which would enable the PIM to be demonstrated to business users in a RAD. However, granularity levels may be a management concern for such application to be realistic.

Since the transformation rules have been verified via manual application to a case study in the previous chapter, and represented formally as QVT-R for use within the MDA in this chapter, the next obvious step is to evaluate the extent to which these transformation rules may be applied within software support, to which the final stage of this research draws attention with relation to this case study in the following chapter.

9.3 Tool Application

The benefit of employing the use of a transformation language such as the QVT is that rules can be defined in abstraction (Peltier et al. 2000), allowing for transformations to be defined at the meta-level. This can be verified with examination of the QVT transformation pattern given in figure 9.3.1 below.

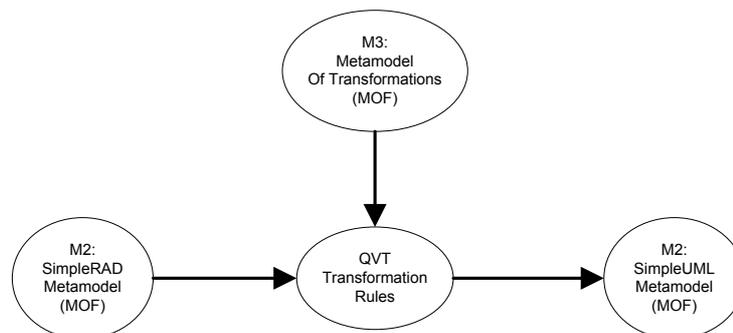


figure 9.3.1, the *rad2umlcd* QVT transformation pattern (Source: developed from FT (2007), Ignjatovic (2006), Koch (2006), Koch et al. (2006), Kusel et al. (2009), OMG (2006a, 2007c, 2008b), Peltier et al. (2000), Sheena et al. (2003)).

Figure 9.3.1 describes a pattern for meta-level transformations whereby the metamodel of transformations (MOF) informs the transformation of models conforming to the *SimpleRAD* metamodel into those conforming to the *SimpleUML* metamodel, using the abstract formalism of QVT transformation rules, which were described as relations and discussed in Section 9.2.

Since these relations have been defined at the meta-level, it is possible to extend the QVT transformation pattern and apply it within a software context to further test the application of the xMDA method (see figure 9.3.2).

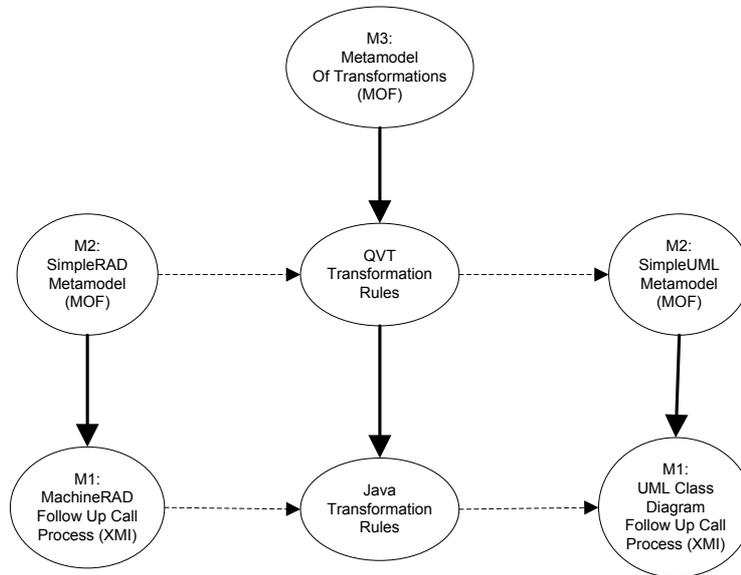


figure 9.3.2, the extended rad2umlcd QVT transformation pattern (Source: developed from FT (2007), Ignjatovic (2006), Koch (2006), Koch et al. (2006), Kusel et al. (2009), OMG (2006a, 2007c, 2008b), Peltier et al. (2000), Sheena et al. (2003)).

In figure 9.3.2, the *Follow Up Call* process *Machine RAD* (created in figure 9.1.3.1) is depicted as being informed by the *SimpleRAD* metamodel (created in figure 7.6.1.1) and the *Follow Up Call* process *Class Diagram* (created in figure 9.1.5.3.1) is depicted as being generated by applying *Java* transformation rules (founded upon the QVT transformation rules defined in Section 9.2), and informed by the *SimpleUML* metamodel (introduced in figure 7.6.2.1). Because these metamodels are defined by, and compliant with, the MOF, XMI can be used as an interchange format (Kovse and Härder 2002). Thus, the transformation pattern now includes the *M1 : Model* layer of the OMG's four-layered architecture (described in Section 7.6.1). As noted previously, transformation rules to create UML Use Case and Activity diagrams were defined in Section 7.6.3, and could be implemented instead of those defined for the generation of the Class Diagram. However, the focus here remains on the generation of the Class Diagram since QVT-R have previously been defined in Section 9.2 for the transformation rules involved and prior experience of transformations in the VIDE project (VIDE 2009) could be drawn upon.

9.3.1 M1 : Machine RAD XMI

The first step in applying the QVT-R described in Section 9.2 within a software context was to create an XMI representation of the *Follow Up Call* process *Machine RAD* at the *M1 : Model* layer. Because there is no

available solution to generate XML from the RAD notation, an alternative solution is required. The VIDE CIM Level Language (VCLL) Editor, available in VIDE (2010a), is a component of VIDE and is used specifically to generate VCLL XML from BPMN diagrams - for a detailed description of how the VIDE toolset might be used within a MDA context, see VIDE (2010b). The VCLL XML created as part of this research to represent the *Follow Up Call* process *Machine RAD* is given in figure 9.3.1.1. This VCLL XML could feasibly be auto-generated to represent the RAD concepts with the provision of a software application that recognises the RAD notation. However, since such a tool is unavailable, the VCLL here was created manually to depict aspects of the *Machine RAD* in XML, based on BPMN concepts for demonstrative purposes.

```
<?xml version="1.0" encoding="UTF-8"?>
<VcllDiagram xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI" xmlns="http://stp.eclipse.org/vcll" xmi:id="fusp" iD="fusp">
<pools xmi:type="Pool" xmi:id="p" iD="p" name="Printer">
  <vertices xmi:type="SubProcess" xmi:id="rpr" iD="rpr" outgoingEdges="1" incomingEdges="2" name="receive print request"/>
  <vertices xmi:type="SubProcess" xmi:id="pbol" iD="pbol" outgoingEdges="1" incomingEdges="2" name="print blow out letter"/>
</pools>
<pools xmi:type="Pool" xmi:id="s" iD="s" name="Sales">
  <vertices xmi:type="SubProcess" xmi:id="scr" iD="rcr" outgoingEdges="1" incomingEdges="2" name="send check request"/>
  <vertices xmi:type="SubProcess" xmi:id="rr" iD="rr" outgoingEdges="1" incomingEdges="2" name="receive response"/>
  <vertices xmi:type="SubProcess" xmi:id="snpr" iD="snpr" outgoingEdges="1" incomingEdges="2" name="send new prospect request"/>
  <vertices xmi:type="SubProcess" xmi:id="cbol" iD="cbol" outgoingEdges="1" incomingEdges="2" name="create blow out letter"/>
  <vertices xmi:type="SubProcess" xmi:id="spr" iD="spr" outgoingEdges="1" incomingEdges="2" name="send print request"/>
  <vertices xmi:type="SubProcess" xmi:id="sur" iD="sur" outgoingEdges="1" incomingEdges="2" name="send update request"/>
</pools>
<pools xmi:type="Pool" xmi:id="bl" iD="bl" name="Brightlime">
  <vertices xmi:type="SubProcess" xmi:id="rcr" iD="rcr" outgoingEdges="1" incomingEdges="2" name="receive check request"/>
  <vertices xmi:type="SubProcess" xmi:id="cpd" iD="cpd" outgoingEdges="1" incomingEdges="2" name="check prospect database"/>
  <vertices xmi:type="SubProcess" xmi:id="sr" iD="sr" outgoingEdges="1" incomingEdges="2" name="send response"/>
  <vertices xmi:type="SubProcess" xmi:id="mpr" iD="mpr" outgoingEdges="1" incomingEdges="2" name="receive new prospect request"/>
  <vertices xmi:type="SubProcess" xmi:id="utpd" iD="utpd" outgoingEdges="1" incomingEdges="2" name="upload to prospect database"/>
  <vertices xmi:type="SubProcess" xmi:id="rur" iD="rur" outgoingEdges="1" incomingEdges="2" name="receive update request"/>
  <vertices xmi:type="SubProcess" xmi:id="upd" iD="upd" outgoingEdges="1" incomingEdges="2" name="update prospect database"/>
</pools>
<pools xmi:type="Pool" xmi:id="pd" iD="pd" name="Prospect Database">
</pools>
</VcllDiagram>
```

figure 9.3.1.1, VCLL representation of the *Follow Up Call* process *Machine RAD*.

Here, *Machine RAD* roles and props are equated to BPMN pools, with operations and interactions becoming BPMN sub-processes. This is not so much to do with these concepts being similar in nature, but more with

what the Java transformation engine outputs in transformation of these concepts in the next section, and these concepts were chosen specifically for that task.

9.3.2 Java Transformation Rules

The choice of using Java for the *M1* : *Model* level mapping was because the implementation already exists as another component of the VIDE toolset, known as the PPT, also available in VIDE (2010a); other mapping languages, such as XSLT, have also been used in academia to the same effect (Dan 2010; Kovse and Härder 2002; Macek and Richta 2009; Peltier et al. 2000). The PPT currently supports VCLL as input. However, the PPT “is envisaged... [to] use different notations” (VIDE 2009), which could be important for the MDA, and the tool, should an application be created to generate XML from a RAD in terms of the xMDA method. It is also important to note that the PPT is an “early prototype” (VIDE 2010a). Several issues became known during this research and included compatibility difficulties in terms of the platform and XML language type; the inability to handle numerous classes relating to the amount of space allocated to output diagrams; ineffectual error handling in terms of reporting errors relating to the input VCLL XML; and errors during Class Diagram modification. However, despite these issues, the prototype was found to be useful in demonstrating the application of the transformations involved in part of this research.

As noted, the Java transformation engine of the PPT treats the BPMN concepts in line with the QVT-R defined in Section 9.2 for related concepts of the *Machine RAD* and, therefore, those relations were chosen specifically to account for that in this demonstration. The relations involved were *Role2Class* (*r*, *c*); *IndependentActivity2Operation* (*ia*, *o*); and *Interaction2Operation* (*i*, *o*); and *Prop2Class* (*p*, *c*). In effect, becoming BPMN *Pool2Class* (*p*, *c*); and *SubProcess2Operation* (*sp*, *o*). An extract of the Java code used within the PPT is given in figure 9.3.2.1 below; the complete Java code relating to the transformation engine of the PPT is given in Appendix VI.

```
// process all pool elements and create appropriate classes
for (int i=0; i<poollist.getLength(); i++) {
    Element element = (Element)poollist.item(i);
    Attr att = element.getAttributeNode("xmi:type");
    if (att != null){
        if (att.getNodeValue().equals("Pool")) {
            Element child = doc.createElement("packagedElement");
            child.setAttribute("xmi:type", "UML:Class");
            child.setAttribute("xmi:id", element.getAttribute("xmi:id"));
            child.setAttribute("name", element.getAttribute("name"));

            // collect lanes and append them to pool
            NodeList lanelist = element.getElementsByTagName("lanes");
            for (int j=0; j<lanelist.getLength(); j++) {
                Element element2 = (Element)lanelist.item(j);
```

```

Attr att2 = element2.getAttributeNode("xmi:type");
if (att2 != null){
    if (att2.getNodeValue().equals("Lane")) {
        Element lanechild = doc.createElement("ownedAttribute");
        lanechild.setAttribute("xmi:id", element2.getAttribute("xmi:id"));
        lanechild.setAttribute("name", String.format("Lane %d", j+1));
        lanechild.setAttribute("visibility", "private");
        child.appendChild(lanechild);
    }
}
}

// collect subprocesses and append them to pool
NodeList vertices = element.getElementsByTagName("vertices");
for (int j=0; j<vertices.getLength(); j++) {
    Element element2 = (Element)vertices.item(j);
    Attr att2 = element2.getAttributeNode("xmi:type");
    if (att2 != null) {
        if (att2.getNodeValue().equals("SubProcess")) {
            Element spchild = doc.createElement("ownedOperation");
            spchild.setAttribute("xmi:id", element2.getAttribute("xmi:id"));
            spchild.setAttribute("name", element2.getAttribute("name"));
            child.appendChild(spchild);
        }
    }
}
packagedElement.appendChild(child);
}
}
}

```

figure 9.3.2.1, Java extract showing how *Pool2Class* (*p,c*) and *SubProcess2Operation* (*sp,o*) relations are implemented in the PPT (Source: by permission from the PPT source files relating to VIDE (2010a)).

These relations have been applied by the Java transformation engine of the PPT to demonstrate how a UML Class Diagram (XMI) could be automatically derived from the VCLL XML relating to *Follow Up Call* process *Machine RAD*, with the objective being to show how realistic the application of these relations within a software solution could be.

9.3.3 M1 : Class Diagram XMI

The VCLL created previously in figure 9.3.1.1 for the *Follow Up Call* process *Machine RAD* was applied to the PPT by running it within the Java transformation engine; the resulting first-cut Class Diagram that was generated by the software, and defined on XML, is given in figure 9.3.3.1.

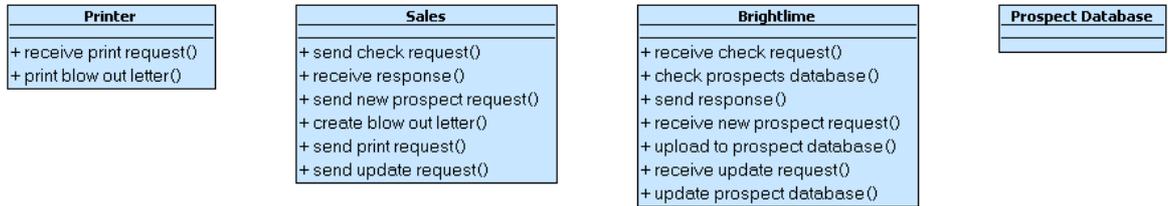
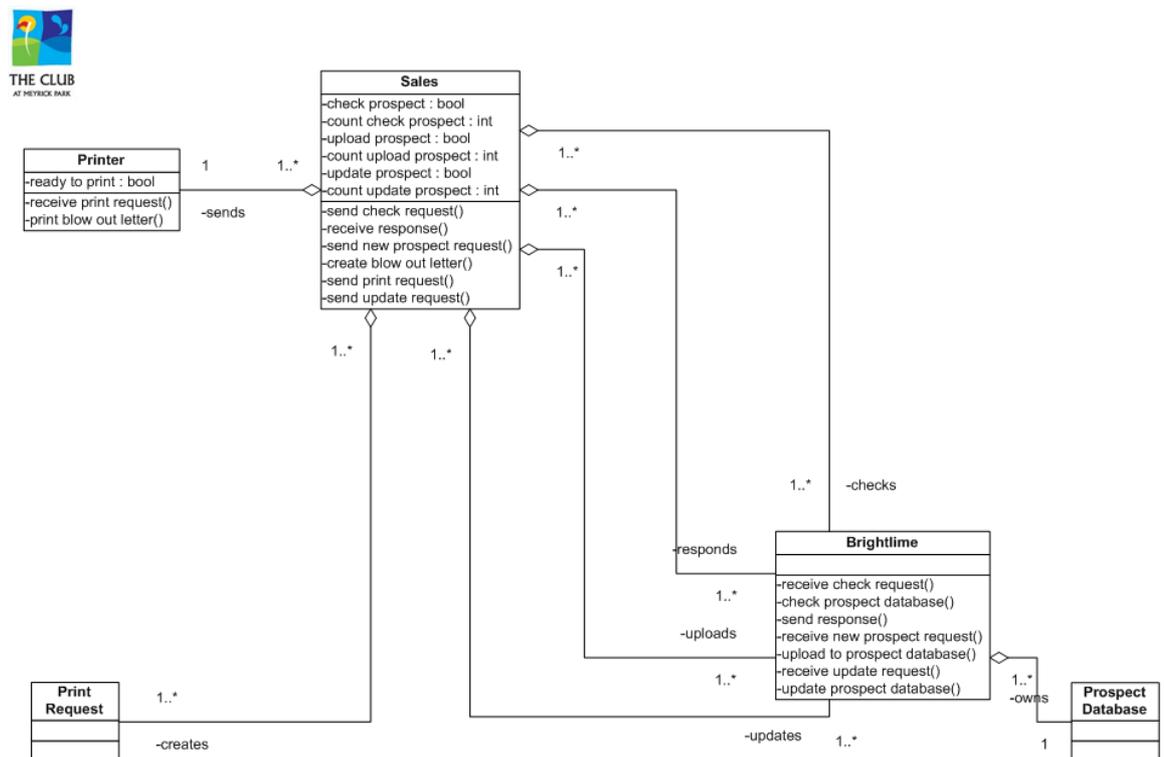


figure 9.3.3.1, UML Class Diagram for the Follow Up Call process Machine RAD created from the application of QVT transformation rules in Java (first-cut).

At this point, it may be useful to be reminded of the manually generated Class Diagram created previously in Section 9.1, to which this machine generated Class Diagram is to be compared (see figure 9.3.3.2 or refer to figure 9.1.5.3.1 for the original discussion relating to this Class Diagram).



Note: Enlarged version appears in Appendix IV.

figure 9.3.3.2, UML Class Diagram for the Follow Up Call process Machine RAD created manually from the application of transformation rules.

9.3.4 Comparative

A comparison between the automatic and manual Class Diagrams revealed a number of observations regarding the application of the transformation rules defined by the QVT-R. As mentioned previously, this transformation is limited to demonstrate the realistic application of four relations defined in Section 9.2.

The number of classes generated by the application corresponds to those created via the manual application in consideration of the *Role2Class* (r, c) and *Prop2Class* (p, c) relations. Original roles of *Sales*; *Printer*; and *Brightlime*, and the *Prospect Database* prop, have all manifested correctly.

Similarly, each class that resulted has been populated with relevant operations in comparison with the manual Class Diagram. This demonstrates the correct implementation of the *IndependentActivity2Operation* (ia, o) and *Interaction2Operation* (i, o) relations in the PPT.

This demonstrates the possibility that a software solution could support the complete set of QVT relations defined in Appendix V to support the xMDA method. However, to create such an application would require considerable extension to the transformation engine. A further auto-generated Class Diagram is provided in figure 9.3.4.2 to reflect the VCLL given below in figure 9.3.4.1, which has been designed to best represent an automatic version of the manual Class Diagram for the *Machine RAD* of the *Follow Up Call* process, by utilising the available rules encoded within the Java transformation engine of the PPT.

```
<?xml version="1.0" encoding="UTF-8"?>
<VclDiagram xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI" xmlns="http://stp.eclipse.org/vcll" xmi:id="fusp" id="fusp">
<pools xmi:type="Pool" xmi:id="s" id="s" name="Sales">
  <vertices xmi:type="SubProcess" xmi:id="scr" id="rcr" outgoingEdges="1" incomingEdges="2" name="send check request"/>
  <vertices xmi:type="SubProcess" xmi:id="rr" id="rr" outgoingEdges="1" incomingEdges="2" name="receive response"/>
  <vertices xmi:type="SubProcess" xmi:id="snpr" id="snpr" outgoingEdges="1" incomingEdges="2" name="send new prospect
  request"/>
  <vertices xmi:type="SubProcess" xmi:id="cbol" id="cbol" outgoingEdges="1" incomingEdges="2" name="create blow out letter"/>
  <vertices xmi:type="SubProcess" xmi:id="spr" id="spr" outgoingEdges="1" incomingEdges="2" name="send print request"/>
  <vertices xmi:type="SubProcess" xmi:id="sur" id="sur" outgoingEdges="1" incomingEdges="2" name="send update request"/>
  <artifacts xmi:type="Role" xmi:id="p" id="p" name="Printer">
    <associations xmi:type="Association" xmi:id="pa" target="s"/>
  </artifacts>
  <artifacts xmi:type="Role" xmi:id="pr" id="pr" name="Print Request">
    <associations xmi:type="Association" xmi:id="pra" target="s"/>
  </artifacts>
  <lanes xmi:type="Lane" xmi:id="l" id="l"/>
</pools>
<pools xmi:type="Pool" xmi:id="bl" id="bl" name="Brightlime">
  <vertices xmi:type="SubProcess" xmi:id="rcr" id="rcr" outgoingEdges="1" incomingEdges="2" name="receive check request"/>
</pools>
```

```

<vertices xmi:type="SubProcess" xmi:id="cpd" iD="cpd" outgoingEdges="1" incomingEdges="2" name="check prospect
database"/>
<vertices xmi:type="SubProcess" xmi:id="sr" iD="sr" outgoingEdges="1" incomingEdges="2" name="send response"/>
<vertices xmi:type="SubProcess" xmi:id="rnp" iD="rnp" outgoingEdges="1" incomingEdges="2" name="receive new prospect
request"/>
<vertices xmi:type="SubProcess" xmi:id="utpd" iD="utpd" outgoingEdges="1" incomingEdges="2" name="upload to prospect
database"/>
<vertices xmi:type="SubProcess" xmi:id="rur" iD="rur" outgoingEdges="1" incomingEdges="2" name="receive update request"/>
<vertices xmi:type="SubProcess" xmi:id="upd" iD="upd" outgoingEdges="1" incomingEdges="2" name="update prospect
database"/>
<artifacts xmi:type="Role" xmi:id="pd" iD="pd" name="Prospect Database">
  <associations xmi:type="Association" xmi:id="pda" target="bl"/>
</artifacts>
</pools>
</VcllDiagram>

```

figure 9.3.4.1, VCLL representation of the Follow Up Call process Machine RAD created for best representation after application of the rules encoded in the PPT.

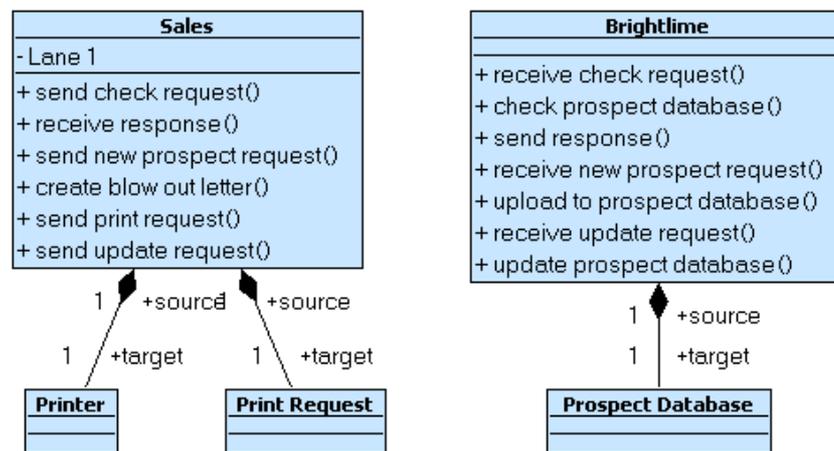


figure 9.3.4.2, UML Class Diagram for the Follow Up Call process Machine RAD created for best representation from the application of the rules encoded in the PPT.

Despite being claimed to be supported (VIDE 2008b), relationships between classes and class attributes were difficult to reproduce using the PPT. Careful examination of the Java source files revealed that this was because a great deal of the Java script that was used was constructed to account for meta-level issues peculiar to the BPMN, which left a complexity in resolving relationships from the *Machine RAD* VCLL XML since no Java code was written to complete that task. The inclusion of figures 9.3.4.1 and 9.3.4.2 exemplifies such meta-level peculiarity where the transformation enforces a relationship of composition type. However, this inclusion demonstrates that relations such as *Interaction2Association* (*i, i2a*) would be possible, given further extension to the tool. Another example is that the only support for the inclusion of attributes was to specify a BPMN lane in the VCLL (i.e. the line `<lanes xmi:type="Lane" xmi:id="1" iD="1"/>` in figure 9.3.4.1 and representation in the *Sales* class of figure 9.3.4.2), which of course is not enough to account for the numerous

relations defined in Appendix V to extract attributes from the *Machine RAD*. This difficulty is extended to account for a number of other relations involving the generation of the Class Diagram in figure 9.3.3.1, including *Interaction2Aggregation (i, ia)* and *Multiplicity (ti, mu)*, that were excluded from this transformation process. Additional Java script is required to remedy this concern.

Other relations may be difficult, if not impossible to realise with further script. For example, the *Print Request* class never materialised in the auto-generated Class Diagram of figure 9.3.3.1 because the relation *IndependentActivity2Class (ia, ia2c)* was unaccounted for. This type of relation relies on a deal of manual guidance and user intervention which would need to be accounted for in any future work that might involve enriching the implementation applied here in order that a fully developed solution is produced. Furthermore, Giandini et al. (2009) discuss that it is important “to formally verify whether [the] implementation is correct with respect to its QVT specification or not” (Giandini et al. 2009). This is specifically useful in the consistency checking of definitions in QVT where they might be described in several (or all) involved imperative/declarative languages; conformance with metamodels (such as the UML and MOF) is provided by the language itself (FT 2007).

9.3.5 Elaboration

The layout provided by the auto generation suffers from some restrictions. There is no rule to apply direction on the positioning of diagram elements. However, support is provided for user intervention in deciding where elements should be placed and for modification after the diagram is generated. Figure 9.3.5.1 is a modified version of the generated Class Diagram given in figure 9.3.3.1, and adheres to all QVT relations discussed in Appendix V.

```

// process all role elements and create appropriate classes
for (int i=0; i<roledlist.getLength(); i++) {
    Element element = (Element)roledlist.item(i);
    Attr att = element.getAttributeNode("xmi:type");
    if (att != null){
        if (att.getNodeValue().equals("Role")) {
            Element child = doc.createElement("packagedElement");
            child.setAttribute("xmi:type", "UML:Class");
            child.setAttribute("xmi:id", element.getAttribute("xmi:id"));
            child.setAttribute("name", element.getAttribute("name"));

            // collect interactions and append them to role
            NodeList vertices = element.getElementsByTagName("vertices");
            for (int j=0; j<vertices.getLength(); j++) {
                Element element2 = (Element)vertices.item(j);
                Attr att2 = element2.getAttributeNode("xmi:type");
                if (att2 != null) {
                    if (att2.getNodeValue().equals("Interaction")) {
                        Element spchild = doc.createElement("ownedOperation");
                        spchild.setAttribute("xmi:id", element2.getAttribute("xmi:id"));
                        spchild.setAttribute("name", element2.getAttribute("name"));
                        child.appendChild(spchild);
                    }
                }
            }

            // collect independent activities and append them to role
            NodeList vertices = element.getElementsByTagName("vertices");
            for (int j=0; j<vertices.getLength(); j++) {
                Element element2 = (Element)vertices.item(j);
                Attr att2 = element2.getAttributeNode("xmi:type");
                if (att2 != null) {
                    if (att2.getNodeValue().equals("IndependentActivity")) {
                        Element spchild = doc.createElement("ownedOperation");
                        spchild.setAttribute("xmi:id", element2.getAttribute("xmi:id"));
                        spchild.setAttribute("name", element2.getAttribute("name"));
                        child.appendChild(spchild);
                    }
                }
            }
            packagedElement.appendChild(child);
        }
    }
}

// process all prop elements and create appropriate classes
for (int i=0; i<propllist.getLength(); i++) {
    Element element = (Element)propllist.item(i);
    Attr att = element.getAttributeNode("xmi:type");
    if (att != null){
        if (att.getNodeValue().equals("Prop")) {
            Element child = doc.createElement("packagedElement");

```

```

        child.setAttribute("xmi:type", "UML:Class");
        child.setAttribute("xmi:id", element.getAttribute("xmi:id"));
        child.setAttribute("name", element.getAttribute("name"));
        packagedElement.appendChild(child);
    }
}

```

figure 9.3.6.1, Java extract showing how the *Role2Class* (*r*, *c*), *IndependentActivity2Operation* (*ia*, *o*), *Interaction2Operation* (*i*, *o*), and *Prop2Class* (*p*, *c*) relations could be implemented in Java (Source: developed from the PPT source files relating to VIDE (2010a)).

9.4 Summary

This chapter has taken a commercial process and applied the xMDA method for moving from analysis into design. Three types of UML model were delivered manually, based on the approach which uses alternate RAD incarnations to account for specification within the CIM phase of the MDA and applies Transformation and Platform Information to generate system models. The resulting diagrams are constructed using notations of the business domain, which have been refined to incorporate specification theory, and presented to the PIM phase of the MDA in a language that is represented by the software community and supported by the MDA (the UML). It was highlighted that the xMDA method is advantageous in terms of guiding the discovery of potential issues in deriving specification within the CIM. This makes the xMDA method both useful to the business and software user, and practical in terms of defining requirements and specification within the MDA. Both visual and textual QVT-R descriptions of rules proposed to transform a RAD to a UML Class Diagram, conforming to the *SimpleRAD* and *SimpleUML* metamodels respectively, were provided, primarily as a means to demonstrate the codification of the rules for use within an MDA implementation in the context of the xMDA. The QVT-R descriptions could also be supported with the provision of QVT-Core and QVT-Operational language definitions if required for implementation. To extend this, the practical application of QVT transformation rules in a software implementation has been demonstrated. Based upon the QVT-R defined at the meta-level, UML classes and operations have been generated from the application of rules, which reflect those described in the manual transformation process.

It is proposed that the xMDA method successfully facilitates a better connection between what is required by business and what is delivered by design, and the applicability to MDA implementations has been demonstrated. Despite the success, it is obvious that further work is required in developing a software solution to provide for a complete set of relations, with further extension being required to support the auto-generation of XML relating to the RAD notation and to provide Java coding to carefully cater for all (or as many as possible) QVT-R involved in *rad2umlcd* transformations and those involving the generation of other UML diagrams. It is also clear that some degree of user involvement will be required to ensure that important

decisions are not overlooked and, perhaps more importantly, that the software implementation has enough flexibility to cater for user-defined creations (including the adjustment of object layout, etc.). Moreover, a complete solution would be required to satisfy the xMDA method in entirety, including the definition and refinement of involved RAD descriptions.

This investigation has illustrated the advantages of introducing the xMDA method in terms of a requirements definition; the beauty being that it forces the discovery of potential issues relating to requirements and specification. Furthermore, the realistic possibility of implementing the xMDA as part of an automated software solution, in line with MDA ideals, has been demonstrated. This has shown that much (if not all) of the transformations described as part of this research could be automated. In application to the VIDE software solutions, a degree of interoperability has also been highlighted and high potential for portability and reusability demonstrated in consideration of output XML. The xMDA method therefore offers promising benefits of application.

Chapter 10

Conclusions

It's easy to rave about how wonderful life will be if you apply all of these great practices. The hard part is incorporating new techniques into the way your organisation routinely operates. The grass-will-be-greener argument motivates some people to change the way they work, but observing that the grass is on fire right behind you is even more persuasive (Wiegers 2000).

To what extent can the MDA incorporate a requirements definition created by business user involvement within the CIM phase of the MDA to be practical in the development of software systems?

Many different techniques have been employed in industry for capturing requirements, designing business processes and ultimately defining software systems. However, from reviewing the available literature, it is clear that there appears to be very limited information available on the practical use of these methods and/or the combination of methods, specifically in the context of the MDA. The MDA offers great potential as a dynamic solution to defining software systems, but somewhat neglects the phases of analysis and specification, leaving the CIM definition incomplete from the perspective of RE. This thesis defines an approach based on the literature review, and the results discovered in part of this research, addressing the connection between business and software ideals within the MDA.

Aim 1: To examine the definition of the CIM within the MDA and consider the appropriation of it as an interface with the business user for defining requirements in MDA notations.

The business user is proposed to be somewhat disconnected from the MDA in the examination of the CIM in terms of notation and method. Objective 1 was to examine the connection between the MDA and business. From reviewing MDA and related literature in Section 4.1.1, it was argued that the MDA has deficiencies and is not yet a mature technology. The CIM was found to be insufficient at connecting software development with the business user, in terms of facilitating a requirements definition, and avenues for extending the CIM were identified. Objective 2 was to determine the sufficiency of the CIM at delivering requirements to the MDA. It was suggested that what is required by the PIM is in conflict with that which is offered by requirements documentation. The case relating to the web-based cinema ticketing system adapted from Wa and Leong (2004) was examined in Section 4.1.2 and supported this argument. By forward and reverse engineering MDA documentation, software concepts supported in the notations of the UML, such as objects

and attributes, were found to be unhelpful in consideration of business users. Such users typically have no notion of software concepts and are therefore unable to utilise these notations when defining requirements. The need for and importance of specification and the system boundary was also highlighted. The MDA is dominated by notations derived from Software Engineering paradigms and would require considerable extension to adequately cater for the accessibility of the business user, specifically in terms of a requirement definition.

Aim 2: To discover how other modelling techniques which are accessible to the business user, might be integrated with the MDA in terms of method and notation, with the focus on transformation and traceability.

The suggestion that situating requirements methods within the MDA may alleviate the inaccessibility of its use by business was addressed by considering CIM-to-PIM and CIM-to-CIM transformations. Objectives 3 and 4 looked respectively at how requirements might be supported by the PIM and how useful CIM definitions are at interfacing with the business user. In Section 5.1.1, a simple order processing system was used to demonstrate how it could be possible to transform from a behavioural model of business (RAD) to a software counterpart (Activity Diagram). A way to produce a foundation static Class Diagram (RUD) derived from the RAD was also found by looking at a sample case relating to a traditional musical jukebox system. However, Section 4.1.2 raised the importance of distinguishing system elements in defining specification. Although the RAD was found to be rich enough to describe necessary detail, the notation has no mechanism to account for the system boundary. Therefore, it was suggested that such transformations may prove unhelpful for software development. In Section 5.1.2 the importance of the system boundary was again highlighted in the analysis of a supported software notation for modelling the CIM (BPMN) and a business technique for specification (Use Case) in terms of the simplified travel reservation system case study adapted from Silver (2008d). The Use Case can represent the system boundary, but it is an extremely simplistic notation and suggested to be incapable of retaining the richness required of the CIM. Both Sections 5.1.1 and 5.1.2 suggest that notations available to business, such as the RAD and the Use Case, can closely represent the business process and be utilised by the business user, yet form no part of CIM definition or artefact to the MDA. Models supported by the MDA, such as those defined in the UML and BPMN, serve better the Software Engineer than the business community. Evidently, transformations between business and software models appear to result in a loss of overall richness, with some requirements being distorted or, at worse, lost to translation. Therefore, a way to enhance the MDA was required by defining an extension which could both retain such richness and appropriately account for requirements and specification in the process.

Aim 3: To extend the framework of the MDA to account for specification within the CIM.

The fusion of the business process and IT is proposed to be an important step for the MDA and one that will need to be made before the consolidation of the MDA infrastructure is made. There is a need for the CIM

definition to adequately account for requirements, without any constraint on modelling method or tool, in order that such fusion between business and software be realistic (Hansz and Fado 2003). This is supported by the findings made in achievement of aims 1 and 2. The argument was to define a mechanism to account for specification within the CIM, which objective 5 was defined to achieve. In Chapters 6.0 and 7.0, the xMDA was proposed to extend the MDA framework, and was illustrated via the application of the xMDA method to a sample order processing case study. The xMDA is defined on the MDA ideals of interoperability, portability and reusability, with specification at the heart of the CIM, thereby embedding RE within the MDA. This is suggested to ensure that systems are developed complete and correct from the requirements stage into the MDA. The use of models, UML-based or otherwise, is encouraged so that they might help “build the system” (Mellor and Balcer 2002). The keyword here is *help*. It does not matter that an *object* is conceptually a software term, as long as real semantics from RE and BPM paradigms are ideally kept. The xMDA draws on this notion by defining abstract mechanisms to move the CIM from analysis to design via specification in the form of three phases (*Environment, Shared* and *Machine*) and by applying Transformation and Platform Information to derive models natural to software design. This was illustrated in Section 6.4 and Chapter 7.0 via the xMDA method in the form of the extended RAD notation which accounts for those phases and meta-level transformation rules to apply the *Machine RAD* to the UML. However, it is the intention that the use of any notation or technique may be applied to the framework, so long as the central notions are adhered to, thus signifying the extensibility of the framework.

Aim 4: To determine the academic and commercial value of extended mechanisms.

The final objective was defined to verify the extended mechanisms of the xMDA to show how the xMDA and associated method could be viable and practical. In Chapter 8.0, the xMDA was evaluated in academia using student subjects. The xMDA method of moving from analysis to design via specification was presented along with other methods to members of the BPR unit on the Software Systems framework at Bournemouth University, with written feedback being drawn upon via thematic analysis. Difficulties in moving from analysis to design were highlighted, with the xMDA method being reflected on positively in terms of measures included to alleviate concerns, such as the consideration of the system boundary, the thoroughness of the technique, and the simplicity of application whilst remaining rich in description. The xMDA method measured successfully in comparison with other techniques, which demonstrated that the xMDA was both viable and accessible to the students. This addressed key factors in moving from analysis to design in the development process, however, practical experience and application remained a student concern. Hence, Chapter 9.0 was directed at verifying the factual application of the xMDA in terms of tools and techniques via a commercial case study based upon The Club at Meyrick Park, Bournemouth. In Section 9.1, the xMDA and associated method were found to be manually applicable to the commercial case study in generating models sufficient for connecting CIM level representations to the PIM via the application of Transformation and Platform Information. Further to this, the xMDA method was found to provide mechanisms that force the

discovery of potential issues for resolution during the specification process and facilitates the discovery of candidate design classes. To demonstrate the applicability of the xMDA to the techniques of the MDA, Section 9.2 (and Appendix V) provided the description of transformation rules in the form of QVT-R definitions. These were then verified in Section 9.3 with application to an MDA tool, producing a software generated model comparable to that which was created from manual application, thus laying foundation to potential semi-automated software support.

“There is no single correct way to analyse system requirements” (Sommerville 2004). However, interaction and collaboration between non-technical stakeholders and technical developers in generating requirements for the system to be delivered in the PIM, rather than transferring elements from the PD into software systems design models (Génova et al. 2005; Nuseibeh and Easterbrook 2000), is considered to be the real benefit of this research. It is likely to remain difficult to make a pronounced connection to business with the current deficiency of specification within the MDA. By giving consideration to the tools and techniques available to RE and the MDA, it is possible to make the connection between domain and software models. This research has exposed the inadequacy of the CIM at delivering a true model of requirements, and provided a solution that includes transformation and traceability mechanisms in terms of the xMDA framework, and verified this via method application.

10.1 Contributions

In conclusion, the contributions of this thesis are viewed to be as follows:

- The justification for and description of an extended framework into which different notations and tools can be placed to facilitate the accessibility of the MDA to business users;
- A unique method, including a mechanism for evolving an analysis RAD into a RAD suitable for specification within the MDA by extending the RAD notation, and rules to transform RAD elements into the UML (e.g. the RUD) derived from the *SimpleRAD* and *SimpleUML* metamodels (developed by this research), conforming to the MOF; and
- The verification of the extended framework and associated method, demonstrating the viability and accessibility of the xMDA to academia as learnable, and to industry as applicable, highlighting analysis problems often overlooked by the MDA when applied to commercial processes, and practical in terms of MDA tools and techniques.

10.2 Related Work

Half of art is knowing when to stop (Arthur W. Radford).

The chairs of the OMG's Object and Reference Model Subcommittee (ORMSC) and the OMG's MDA Users' Group have been approached via the Vice President and Technical Director of the OMG regarding the findings of this research, in the hope that the xMDA might affect the MDA standard to account for discoveries made by this research. The investigation into the applicability of the xMDA is expected to continue, specifically focusing on extending the verification of commercial application, and further submissions to the OMG regarding the direction and future of the MDA Guide are likely to be made.

Further research is required in examining the xMDA, utilising alternate and combined techniques, with the view of discovering more about how versatile the proposal really is. Analysis could be conducted in terms of the RAD replacing current modelling conventions, such as the Use Case and Activity diagrams. Therefore, further theoretical examination ought to be conducted relating to aim 2 on how other modelling techniques which are accessible to the business user might be integrated with the MDA, and how such techniques might complement one another. Specifically, this could be directed at discovering whether or not there is correlation between techniques and how the early derivation and/or automation of test cases might support the software process. Suggestion is given that the interface between analysis and design can be achieved within the MDA through the application of appropriate techniques which are not founded on Software Engineering concepts, so continued focus should be given to such techniques and applying those that hold the most promise to the xMDA. The xMDA method could be applied with greater rigor in a multitude of situations and combinations, or even in an enterprise test environment, so as to ascertain whether or not the Transformation and Platform Information could really be bi-directional and applicable across multiple languages. For example, supporting the generation of a RAD from the UML, and supporting the generation of models to and from alternate languages. This would be ideal since it would enable models of design to be reversed engineered into a RAD (or other suitable language) that facilitates the understanding of business and software, such as the BPMN. This type of investigation could be extended to examine the validity of the xMDA in support of procedural and combined solutions, away from object orientation and the support of other software processes such as the SOA. Moreover, analysis is required in determining how sufficient the *Machine RAD* generated Class Diagram is to design. It is possible that since the Class Diagram is generated out of specification in a software language, designers may *presume* the design task is complete (or even started). However, this should not be the case since xMDA output is pure specification, and not design at all. The use of this output therefore amounts to nothing more than specification, irrespective of language, and ought not to be considered otherwise without sufficient attention being placed on design.

Regarding tool support for the xMDA method, there are many avenues of research worthy of consideration. An enactor engine for the RAD that incorporates the xMDA would be useful in part to develop a tool that entirely supports BPM and the xMDA, since none is currently available. Similarly, an enhancement to enable XML to be generated from the RAD notation would also assist in moving the RAD into mainstream process modelling within BPM and software development. This investigation could be extended to further account for more, if not all, of the QVT-R described in Appendix V, by producing the necessary Java code. With a degree of vision, it is possible to imagine an xMDA tool that could support many alternate transformation definitions, in many different languages, for many alternate notations. This would require a powerful transformation engine that should be orchestrated to account for user-defined creations in the refinement of analysis models through specification and ultimately, into design via Transformation and Platform Information. Moreover, the increase in complexity and cost for users to learn associated tools and methods also requires consideration.

From the wider perspective of the MDA, other avenues of research could be taken to investigate the application further. The MDA comes with much promise, however, little is known about the real cost associated with MDA projects and modelling training. Many tools claim to support MDA ideals, but just as many are found to be in breach of the MDA specification and cause users to become locked into using specific tools, mutate available conventions, or to neglect phases (such as the CIM) because tool support is questionable or unavailable. There are also other avenues to model driven development (and development in general) which could be investigated comparatively with the MDA (such as the SOA) to see if the promise the MDA offers is realistic in terms of application and implementation in the wider software process field.

The outcome of software system operation in the real-world is inherently uncertain with the precise area of uncertainty also not knowable (Lehman 1989).

In summary, the key to successfully migrating from an outlined description of user needs to a concise set of specifications, and indeed, eventually to a fully functional and successful system, is to understand the customer requirements, the software application, the available resources and the surrounding environment (Sommerville 2004). Moreover, Al-Neimat (2005), Bilodeau (2010), Gonzales (2009b), Ward-Dutton (2011) raise the important issue of managerial support for such initiatives. Rothman (Rothman 2007) specifically highlights key management ideals, whilst not being exhaustive, it is thought that the presence of such support systems would help to ensure that the right management culture is in place for the successful implementation of the xMDA. The Greek philosopher Plato once proclaimed that “no law or ordinance is mightier than understanding” and in the context of successful MDA implementation, the understanding between business and software is vital.

References

- Abeysinghe, G., and Phalp, K., 1997. Combining process modelling methods. *Information and Software Technology*, 39 (2), 107 - 124.
- Adler, A. D. 2001. New frontiers in sketch understanding (pp. 9): MIT MEngTP.
- Al-Neimat, T., 2005. Why it projects fail. *The Project Perfect White Paper Collection*. Available from: http://www.projectperfect.com.au/downloads/Info/info_it_projects_fail.pdf [Accessed: 22nd June 2009].
- Ambler, S. W., 2007. A roadmap for agile MDA. Available from: <http://www.agilemodeling.com/essays/agileMDA.htm> [Accessed: 26th October 2007].
- Ample. 2007. Ample project. Available from: <http://www.ample-project.net/> [Accessed: 31st October 2007].
- Appukkuttan, B., Clark, T., Reddy, S., Tratt, L., and Venkatesh, R., 2003a, Tuesday, October 21st, 2003. *A model driven approach to building implementable model transformations*. Paper presented at the Workshop in Software Model Engineering, 6th International Conference on the Unified Modelling Language, San Francisco, USA.
- Appukkuttan, B., Clark, T., Reddy, S., Tratt, L., and Venkatesh, R., 2003b. A model driven approach to model transformations. *Model Driven Architecture: Foundations and Applications*. University of Twente, 7 - 18.
- Ashok, V., Ramanathan, J., Sarker, S., and Venugopal, V., 1988, 11 - 13th May 1988. *Process modelling in software environments*. Paper presented at the 4th International Software Process Workshop, Moretonhampstead, Devon, UK.
- Atwood, D., 2006. BPM process patterns: Repeatable design for BPM process models. *BPTrends*. Available from: <http://www.businessprocesstrends.com/publicationfiles/05%2D06%2DWDP%2DBPMProcessPatterns%2DAtwood1%2Epdf> [Accessed: 20th February 2008].
- Badica, C., Teodorescu, M., Spahiu, C., Badica, A., and Fox, C., 2005. *Integrating Role Activity Diagrams and hybrid IDEF for business process modelling using MDA*. Paper presented at the 7th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing.
- Baskerville, R. L., 1997. Distinguishing action research from participative case studies. *Journal of Systems and Information Technology*, 1 (1), 25 - 45.
- Baskerville, R. L., 1999. Investigating information systems with action research. *Communications of the Association for Information Systems*, 2 (19), 32.
- Basson, G., 2009a. Integrated business management in the process age: Creating an agile business management paradigm. *BPMInstitute.org*. Available from: <http://www.bpminstitute.org/articles/article/article/integrated-business-management-in-the-process-age-creating-an-agile-business-management-paradigm.html> [Accessed: 3rd June 2009].
- Basson, G., 2009b. Process-oriented systems paradigm for the process age. *BPMInstitute.org*. Available from: <http://www.bpminstitute.org/articles/article/article/process-oriented-systems-paradigm-for-the-process-age.html> [Accessed: 2nd July 2009].
- Beeson, I., Green, S., Sa, J., and Sully, A., 2002. Linking business processes and information systems provision in a dynamic environment. *Information Systems Frontiers*, 4 (3), 317-329.
- Berrisford, G., 2004, September 2004. *Why IT veterans are sceptical about MDA*. Paper presented at the 2nd European Workshop On Model Driven Architecture, Canterbury, UK.
-

-
- Bilodeau, N., 2010. Process ownership and governance; paradigm shift. *BPMInstitute.org*. Available from: <http://www.bpminstitute.org/articles/article/article/process-ownership-and-governance-paradigm-shift.html> [Accessed: 28th March 2011].
- Blanc, X., 2009, 2nd April 2009. *Model Driven Software Engineering Environment*. Paper presented at the 4th edition of Kermeta Workshop, IRISA Rennes, France.
- Bleistein, S. J., Cox, K., Verner, J., and Phalp, K. T., 2006. B-SCP: A requirements analysis framework for validating strategic alignment of organisational it based on strategy, context and process. *Information and Software Technology*, 48 (9), 846 - 868.
- Boyko, S., Dvorak, R., and Igdalov, A., 2009, 23rd - 26th March 2009. *The art of model transformation with operational QVT*. Paper presented at the EclipseCon, Santa Clara, California.
- Braa, K., and Vidgen, R., 1999. Interpretation, intervention, and reduction in the organisational laboratory: A framework for in-context information system research. *Accounting, Management and Information Technologies*, 9 (1), 25 - 47.
- Brahe, S., and Bordbar, B., 2006, December 4 - 7th, 2006. *A pattern-based approach to business process modelling and implementation in web services*. Paper presented at the 4th International Conference on Service-Oriented Computing (ICSOC), Chicago, IL, USA.
- Brannen, J. 2005. Mixed methods research: A discussion paper, *NCRM Methods Review Papers*: Institute of Education, University of London.
- Bray, I., 2002. *An introduction to requirements engineering*. England: Addison Wesley.
- Bray, I., 2004. Lecture notes on analysis. *Bournemouth University*. Available from: [http://dec.bournemouth.ac.uk/staff/ibray/RElecture03\(analysis&SA\).ppt](http://dec.bournemouth.ac.uk/staff/ibray/RElecture03(analysis&SA).ppt) [Accessed: 16th March 2011].
- Brooks, F., 1975. *The mythical man-month: Essays on software engineering*. Boston, USA: Addison-Wesley Publishing.
- Brown, A. W., 2004a. An introduction to Model Driven Architecture. Available from: <http://www-128.ibm.com/developerworks/rational/library/3100.html> [Accessed: 20th November 2007].
- Brown, A. W., 2004b. Model Driven Architecture: Principles and practice. *Software Systems Modelling*, 3, 314 - 327.
- Brown, A. W., 2008, 25th - 29th February. *MDA redux: Practical realization of Model Driven Architecture*. Paper presented at the 7th International Conference on Composition-Based Software Systems (ICBSS 2008), Madrid, Spain.
- Brown, A. W., Iyengar, S., and Johnston, S., 2006. A rational approach to Model-Driven Development. *IBM Syst. J.*, 45 (3), 463-480.
- Bureck, M., 2009, 9th June 2009. *Visual QVT/R: A concrete graphical syntax for QVT/R*. Paper presented at the Eclipse DemoCamp, Berlin.
- Bushell, S., 2005. Is there a method to the BPM madness? A review of two books describing a new foundation for business process management. *A BPT Book Review*. Available from: <http://www.bptrends.com/publicationfiles/10-05%20BR%20BPM%20and%20Human%20Interactions%20-%20%20Sue%20Bushell.pdf> [Accessed: 19th April 2010].
- Cachia, E., 2005. DFD from textual description (an example). *The University of Malta*. Available from: http://staff.um.edu.mt/ecac1//files/DFD_from_description.pdf [Accessed: 6th July 2009].
- Card, D. N., Church, V. E., and Agresti, W. W., 1986. An empirical study of software design practices. *IEEE Trans. Softw. Eng.*, 12 (2), 264-271.
-

-
- Casallas, R., Acero, C., and López, N., 2005. *From high level business rules to an implementation on an event-based platform to integrate applications*. Paper presented at the EDOC Workshop, VORTE, Enshede, The Netherlands.
- Castro, J., Kolp, M., and Mylopoulos, J., 2002. Towards requirements-driven information systems engineering: The tropos project. *Inf. Syst.*, 27 (6), 365-389.
- Celms, E., Kalnins, A., and Lace, L., 2003, October 2003. *Diagram definition facilities based on metamodel mappings*. Paper presented at the 18th International Conference, OOPSLA'2003 (Workshop on Domain-Specific Modelling) Anaheim, California, USA.
- Checkland, P., 1981. *Systems thinking, systems practice*. John Wiley & Sons, Chichester.
- Checkland, P., 2000. Soft systems methodology: A thirty year retrospective. *Systems Research and Behavioural Science*, 17 (S1), S11 - S58.
- Checkland, P., and Scholes. 1990. *Soft systems methodology in action*. John Wiley & Sons, Chichester.
- Chen, P. P.-S., 1976. The Entity-Relationship model - toward a unified view of data. 1 (1), 9 - 36.
- Chen, P. P., 1983. English sentence structure and Entity-Relationship diagrams. *Information Sciences*, 1 (1), 127 - 149.
- Cobern, W. W., 1993. *World view, metaphysics, and epistemology*. Paper presented at the Annual Meeting of the National Association for Research in Science Teaching, Atlanta, Georgia, U.S.A.
- Cockburn, A., 2007. *Agile software development : The cooperative game*. Second ed. Upper Saddle River, NJ: Addison-Wesley.
- Conradi, R., Fernström, C., Fuggetta, A., and Snowdon, R. A., 1992. *Towards a reference framework for process concepts*. Paper presented at the 2nd European Workshop on Software Process Technology.
- Conradi, R., Høydalsvik, G. M., and Sindre, G., 1994, February 1994. *A comparison of modelling frameworks for software processes and information systems*. Paper presented at the 3rd European Workshop on Software Process Technology (EWSPT), Villard de Lans, France.
- Cook, S., 2004a. MDA journal: Domain specific modelling and Model Driven Architecture. *A BPT Column*. Available from:
http://www.bptrends.com/deliver_file.cfm?fileType=publication&fileName=01%2D04%20COL%20Dom%20Spec%20Modeling%20Frankel%2DCook%2Epdf [Accessed: 6th January 2004].
- Cook, S., 2004b, October 11-14, 2004. *Software factories*. Paper presented at the Strategic Architect Forum, Redmond, Washington.
- Coughlan, J., and Macredie, R., 2002. Effective communication in requirements elicitation: A comparison of methodologies. *Requirements Engineering*, 7 (2), 47 - 60.
- Cox, K., Dubois, E., Pigneur, Y., Bleistein, S. J., Verner, J., Davis, A. M., and Wieringa, R., 2005a, 29 - 30th August 2005. *Introductory notes*. Paper presented at the 1st International Workshop on Requirements Engineering for Business Need and IT Alignment (REBNITA), part of the 13th IEEE International Requirements Engineering Conference (RE), Sorbonne, Paris, France.
- Cox, K., and Phalp, K., 2003, 16 - 17 June 2003. *From process model to problem frame - a position paper*. Paper presented at the 9th International Workshop on Requirements Engineering: Foundation For Software Quality (REFSQ), Essener Informatik Beitrage, Velden, Austria.
- Cox, K., Phalp, K., and Shepperd, M., 2001, June 2001. *Comparing use case writing guidelines*. Paper presented at the 7th international workshop on requirements engineering: Foundation for software quality (REFSQ'01), Interlaken, Switzerland.
- Cox, K., and Phalp, K. T., 2007. Practical experience of eliciting classes from use case descriptions. *J. Syst. Softw.*, 80 (8), 1286-1304.
-

-
- Cox, K., Phalp, K. T., Bleistein, S. J., and Verner, J. M., 2005b. Deriving requirements from process models via the problem frames approach. *Information and Software Technology*, 47 (5), 319-337.
- Dan, L., 2010, 22nd - 26th March 2010 *QVT based model transformation from sequence diagram to CSP*. Paper presented at the 5th IEEE International workshop UML and AADL, held in conjunction with the 15th International Conference on Engineering of Complex Computer Systems (ICECCS), University of Oxford, UK.
- Dawkins, S., 1998. *Role Activity Diagrams for safety process definition*. Paper presented at the 16th International System Safety Conference, Seattle, WA, USA.
- Debevoise, T., and Smith, M., 2009. Round table: A truly business-friendly approach to BPM and BRM. *BPMInstitute.org*. Available from: <http://www.bpminstitute.org/roundtables/upcoming-round-table/article/a-truly-business-friendly-approach-to-BPM-and-brm.html> [Accessed: 18th March 2009].
- Deiters, W., Gruhn, V., and W.Schafer. 1989, 10 - 13th October 1989. *Process programming: A structured multi-paradigm approach could be achieved*. Paper presented at the 5th International Software Process Workshop, Kennebunkport, Maine, USA.
- Demarco, T., 1979. *Structured analysis and system specification*. Prentice Hall PTR.
- Deveaux, R. D., Velleman, P. F., and Bock, D. E., 2005. *Stats: Data & models*. International ed. London, UK: Addison-Wesley.
- Dijkstra, E. W., 1968. Letters to the editor: Go To statement considered harmful. *Communications of the ACM*, 11 (3), 147 - 148.
- Dowdle, P., and Stevens, J., 2009. Starting the journey towards process based management. *BPMInstitute.org*. Available from: <http://www.bpminstitute.org/articles/article/article/starting-the-journey-towards-process-based-management.html> [Accessed: 4th June 2009].
- Dowson, M., 1987a. IStar - an integrated project support environment. *ACM SIGPLAN Notices*, 22 (1), 27 - 33.
- Dowson, M., 1987b, 30th March - 2nd April. *IStar and the contractual approach*. Paper presented at the 9th international conference on Software Engineering, Monterey, California, USA.
- Dowson, M., 1987c, 30th March - 2nd April. *Iteration in the software process; review of the 3rd international software process workshop*. Paper presented at the 9th international conference on Software Engineering, Monterey, California, USA.
- Dubielewicz, I., Hnatkowska, B., Huzar, Z., and Tuzinkiewicz, L., 2006. An approach to evaluation of PSM/MDA database models in the context of transaction performance. *International Journal Of Computer Science And Network Security (IJCSNS)*, 6 (10).
- Dwyer, T., 2010. Is the Business Process Modelling Notation (BPMN) suitable for use by business people? Available from: http://brainstorm.leveragesoftware.com/group_discussion.aspx?DiscussionID=1d0a140989b04ac191f87fb87c3e8707 [Accessed: 3rd March 2011].
- Easterbrook, S., 2003. Lecture 6: Requirements modelling ii. *Lecture notes on Requirements Engineering, University of Toronto*. Available from: <http://www.cs.toronto.edu/~sme/CSC2106S/slides/06-modeling-info-behav.pdf> [Accessed: 31st July 2009].
- Edlich, S., Paterson, J., and Hörning, H., 2006. *The definitive guide to DB4O*. Apress.
- Elliott, J., and Raynor-Smith, P., 2000, February 2000. *Achieving customer satisfaction through requirements understanding*. Paper presented at the 7th European Workshop on Software Process Technology (EWSPT) Kaprun, Austria.
-

-
- Fingar, P., 2007. Extreme competition - edp audit and control redux. *BPTrends*. Available from: <http://www.bptrends.com/publicationfiles/11%2D07%2DCOL%2DEDP%20Audit%2DFingar%2DFinal%2Epdf> [Accessed: 27th March 2008].
- Finkelstein, A., 1987, 17 - 19th November 1986. *Making formal specifications dynamic objects*. Paper presented at the 3rd International Software Process Workshop, Colorado, USA.
- Ford, N., 2009. UML failed so here we have AML (Arbitrary Modelling Language). *Architects Zone*. Available from: <http://architects.dzone.com/news/UML-failed-so-here-we-have-aml> [Accessed: 26th June 2009].
- Fouad, A., Phalp, K., Jeary, S., and Kanyaru, J. M., 2009, 6 - 8th April 2009. *The consideration of a requirements phase in the Model Driven Architecture*. Paper presented at the Software Quality in the 21st Century, Software Quality XVII, 17th International Software Quality Management Conference (SQM), Southampton, UK.
- Fouad, A., Phalp, K., Kanyaru, J. M., and Jeary, S., 2011. Embedding requirements within the Model Driven Architecture. *Software Quality Journal*, 19 (2), 411 - 430.
- Frank, U., 2002. *Multi-perspective Enterprise Modelling (MEMO) - conceptual framework and modelling languages*. Paper presented at the 35th Annual Hawaii International Conference on System Sciences (HICSS).
- Frankel, D. S., 2005. MDA journal: Eclipse and the MDA. *A BPT Column*. Available from: <http://www.bptrends.com/publicationfiles/03%2D05%20COL%20Eclipse%20and%20MDA%20%20%2D%20Frankel1%2Epdf> [Accessed: 24th September 2008].
- Frankel, D. S., 2006. Model-driven business process platforms. Available from: <http://www.omg.org/MDA/MDA-webcasts.htm> [Accessed: 2nd November 2007].
- FT. 2007. Smartqvt documentation, *France Telecom*.
- Gane, C., and Sarson, T., 1977. *Structured systems analysis: Tools and techniques*. McDonnell Douglas Systems Integration Company.
- Gao, Y., 2006. BPMN - BPEL transformation and round trip engineering. Available from: http://www.eclarus.com/pdf/BPMN_BPEL_Mapping.pdf [Accessed: 18th August 2008].
- Garrido, J. L., Noguera, M., Gonzalez, M., Hurtado, M. V., and Rodríguez, M. L., 2007. Definition and use of Computation Independent Models in an MDA-based groupware development process. *Science of Computer Programming*, 66 (1), 25 - 43.
- Génova, G., Valiente, M. C., and Nubiola, J., 2005, 13 - 14 June 2005. *A semiotic approach to UML models*. Paper presented at the 1st International Workshop on Philosophical Foundations of Information Systems Engineering (PHISE) - In the proceedings of the 17th Conference on Advanced Information Systems Engineering (CAiSE) workshops, Porto, Portugal.
- Giandini, R., Pons, C., and Pérez, G., 2009, 13 - 17th April 2009. *A two-level formal semantics for the QVT language*. Paper presented at the 12th Ibero-American conference on software engineering (CIBSE), Medellín, Colombia.
- Gill, J., and Johnson, P., 1997. *Research methods for managers*. Second ed. Hants, UK: Paul Chapman Publishing Limited.
- Glaser, B. G., and Strauss, A., 1967. *The discovery of grounded theory: Strategies for qualitative research*. Chicago: Aldine Transaction.
- Golbaz, M., Hasheminasab, A., and Daneshpour, N., 2008, 19 - 21 March 2008. *An XML definition language to support use case-based requirements engineering*. Paper presented at the International MultiConference of Engineers and Computer Scientists (IMECS), Regal Kowloon Hotel, Tsimshatsui, Kowloon, Hong Kong.
-

-
- Gonzales, R. C., 2009a. Formal definition of the process to be automated (second pillar). *BPMInstitute.org*. Available from: <http://www.bpminstitute.org/articles/article/article/formal-definition-of-the-process-to-be-automated-second-pillar.html> [Accessed: 23rd October 2009].
- Gonzales, R. C., 2009b. Four pillars for business process automation. *BPMInstitute.org*. Available from: <http://www.bpminstitute.org/articles/article/article/four-pillars-for-business-process-automation/news-browse/1.html> [Accessed: 5th June 2009].
- Gotel, O. C. Z., and Finkelstein, A. C. W., 1994, 18th - 22nd April 1994. *An analysis of the requirements traceability problem*. Paper presented at the 1st International Conference on Requirements Engineering (ICRE'94), Colorado Springs, USA.
- Greenspan, S., Mylopoulos, J., and Borgida, A., 1994. *On formal requirements modelling languages: RML revisited*. Paper presented at the 16th international conference on Software engineering, Sorrento, Italy.
- Greenspan, S. J., Mylopoulos, J., and Borgida, A., 1982. *Capturing more world knowledge in the requirements specification*. Paper presented at the 6th international conference on Software engineering, Tokyo, Japan.
- Grimm, F., Beier, G., Thalp, K., and Vincent, J., 2007, 20th February 2007. *Towards semiautomatic, mental-map-preserving visual merging of UML class models*. Paper presented at the IADIS International Conference Applied Computing, Salamanca, Spain.
- Grützner, I., Münch, J., Fernandez, A., and Garzaldeen, B., 2004. Guided support for collaborative modelling, enactment and simulation of software development processes. *Software Process: Improvement and Practice - Special Issue on ProSim 2003, The 4th International Workshop on Software Process Simulation and Modelling, Portland, OR, May 2003*, 9 (2), 95 - 106.
- Gunter, C. A., Gunter, E. L., Jackson, M., and Zave, P., 2000. A reference model for requirements and specifications. *IEEE Softw.*, 17 (3), 37-43.
- Gupta, N., 2007a. Descriptive specifications of software systems. *The University of Arizona*. Available from: <http://www.cs.arizona.edu/classes/cs436/spring07/Lectures/LogicSpec1.PDF> [Accessed: 27th July 2009].
- Gupta, N., 2007b. Finite state machines. *The University of Arizona*. Available from: <http://www.cs.arizona.edu/classes/cs436/spring07/Lectures/FSM.pdf> [Accessed: 22nd July 2009].
- Gupta, N., 2007c. Introduction to software specifications and Data Flow Diagrams. *The University of Arizona*. Available from: <http://www.cs.arizona.edu/classes/cs436/spring07/Lectures/IntroDFD.pdf> [Accessed: 21st July 2009].
- Gustavson, P., 2004, September 11-15th. *Fitting the UML into your development process*. Paper presented at the Borland Conference, San Jose, California.
- Haan, J. D., 2009. 8 reasons why model-driven development is dangerous. *The Enterprise Architect: Building and Agile Enterprise*. Available from: <http://www.theenterpriseearchitect.eu/archive/2009/06/25/8-reasons-why-model-driven-development-is-dangerous> [Accessed: 1st July 2009].
- Haan, J. D., 2011. Why aren't we all doing model driven development yet? Available from: <http://www.theenterpriseearchitect.eu/archive/2011/04/16/why-arent-we-all-doing-model-driven-development-yet> [Accessed: 20th April 2011].
- Hammond, T. A. 2001. Natural sketch recognition in UML class diagrams, *MIT Student Oxygen Workshop*. USA: MIT.
- Hampton, C. R., 2004. Epistemology to Ontology. *Journal of the ACMS*.
- Hansz, D., and Fado, D., 2003, December 2-5. *Unambiguous, non-binding requirements for MDA*. Paper presented at the MDA™ Implementers' Workshop Succeeding with Model Driven Systems, Burlingame, CA, USA
-

-
- Harel, D., 1988. On visual formalisms. *Communications of the ACM*, 31 (5), 514 - 530.
- Harmon, P., 2005. Email advisor: Standardising business process notation. *Business Process Trends*, 3 (19).
- Harmon, P., 2006. Review of "Learning to see" By Mike Rother and John Shook. The Lean enterprise, ver. 1.3, 2003. A *BPT Book Review*. Available from: <http://www.bptrends.com/publicationfiles/01%2D06%20BR%20Learning%20to%20See%20%2D%20Rother%2DShook%20ph1%2Epdf> [Accessed: 3rd September 2008].
- Harrington, J. L., 2000. *Object-oriented database design clearly explained*. Morgan Kaufmann Publishers Inc.
- Harrison-Broninski, K., 2005a. Modelling human interactions: Part 2. *BPTrends*. Available from: <http://www.businessprocesstrends.com/publicationfiles/07%2D05%20WP%20Modeling%20Human%20Interactions%20%2D%20Pt%202%20%2D%20Harrison%2DBro%2E%80%A6%2Epdf> [Accessed: 29th August 2008].
- Harrison-Broninski, K., 2005b. RADs and the UML. 1.0. Available from: http://human-interaction-management.info/RADs_and_the_UML_1_0.pdf [Accessed: 2nd September 2008].
- Harrison-Broninski, K., 2005c. The technology of human interaction management. *BPM.COM*. Available from: <http://www.BPM.com/FeatureRO.asp?FeatureId=168> [Accessed: 26th March 2008].
- Harrison-Broninski, K., 2006a. BPM, anyone? *BPTrends*. Available from: http://www.businessprocesstrends.com/deliver_file.cfm?fileType=publication&fileName=02-06-DIS-Re-ReStandardizingBPNotation-Harrison-Broninski.pdf [Accessed: 6th February 2008].
- Harrison-Broninski, K., 2006b. Business system support case study. *BPTrends*. Available from: <http://www.businessprocesstrends.com/publicationfiles/01%2D06%2DCS%2DBusiness%5FSystem%5FSupport%5FCase%5FStudy%2DHarrison%2DBroninski%2Epdf> [Accessed: 27th February 2008].
- Harrison-Broninski, K., 2006c. The future of BPM (parts 1 to 6). *BPTrends*. Available from: http://www.businessprocesstrends.com/resources_publications.cfm?publicationtypeID=DFFB9D1C-1031-D522-3AAF1211DDD4AD95 [Accessed: 18th November 2008].
- Harrison-Broninski, K., and Hayden, F., 2004. Role-based transaction management in collaborative systems. Available from: <http://66.102.1.104/scholar?hl=en&lr=&q=cache:BFLxbIi-S5AJ:www.human-interaction-management.info/A%2520Role-Based%2520Approach%2520To%2520Business%2520Process%2520Management.pdf+> [Accessed: 14th November 2008].
- Havey, M., 2007. The flesh and bone of soa. *SOA World Magazine*. Available from: <http://soa.sys-con.com/node/380265/print> [Accessed: 13th November 2008].
- Henderson, P., and Pratten, G. D., 1995, November 6 - 10th. *POSD - a notation for presenting complex systems of processes*. Paper presented at the 1st IEEE International Conference on Engineering of Complex Computer Systems (ICECCS), Florida, USA.
- Hendryx, S., Vincent, P., and Cribbs, J., 2002, October 21st - 24th, 2002. *Business rules with MDA*. Paper presented at the Third Workshop on UML® for Enterprise Applications: Model Driven Solutions for the Enterprise, San Francisco, CA, USA.
- Hoare, C. A. R., 1978. Communicating Sequential Processes. *Communications of the ACM*, 21 (8), 666 - 677.
- Hofstader, J., 2006. Building distributed applications - model-driven development. Available from: <http://msdn2.microsoft.com/en-us/library/aa964145.aspx> [Accessed: 10th October 2007].
- Hogg, L., 2009. Case management strategies and best practices: Taking traditional BPM way beyond conventional workflow, routing and collaboration. *BPMInstitute.org*. Available from: <http://www.bpminstitute.org/roundtables/upcoming-round-table/article/taking-traditional-BPM-way-beyond-conventional-workflow-routing-and-collaboration.html> [Accessed: 1st September 2009].
-

-
- Horan, P., 2000. *Using rich pictures in information systems teaching*. Paper presented at the 1st International Conference on Systems Thinking in Management (ICSTM), Geelong, Australia.
- Huseth, S., and Vines, D., 1987, 17 - 19th November 1986. *Describing the software process*. Paper presented at the 3rd International Software Process Workshop, Colorado, USA.
- Husserl, E., 2001. *Logical investigations*. London: Routledge.
- Huxley, A., 1971. *The doors of perception & heaven and hell*. Middlesex, UK: Penguin Books Limited.
- Ignjatovic, M., 2006. OMG: Query/View/Transformation. *ObjektSpektrum*. Available from: http://www.sofismo.ch/links/OMG-QVT_ObjektSpektrum_2006_E.pdf [Accessed: 6th April 2010].
- Ikeda, K., 2007. A critical examination of the interview as a research method for qualitative language-based studies. *Language and Culture, Nagoya University Graduate School of Languages and Cultures*, 29 (1), 63 - 73.
- Ince, D. C., Sharp, H., and Woodman, M., 1993. *Introduction to software project management and quality assurance*. Maidenhead, Berkshire, England: McGraw-Hill, Inc.
- Issa, A., Odeh, M., and Coward, D., 2005. *Using use case models to generate object points*. Paper presented at the IASTED International Conference on Software Engineering, Austria.
- Jackson, M., 1995. *Software requirements & specifications: A lexicon of practice, principles and prejudices*. Harlow, England: ACM Press Books, Addison-Wesley.
- Jackson, M., 2000. *The real world*. Paper presented at the Millennial Perspectives in Computer Science, 1999 Oxford-Microsoft symposium in honour of Sir Antony Hoare, St. Catherine's College, Oxford, UK.
- Jackson, M., and Zave, P., 1993. *Domain descriptions*. Paper presented at the 2nd IEEE International Symposium on Requirements Engineering, Los Alamitos, CA, USA.
- Jackson, M., and Zave, P., 1995. *Deriving specifications from requirements: An example*. Paper presented at the 17th international conference on Software engineering, Seattle, Washington, United States.
- Jackson, M. A., 1982. A system development method. In: *Tools and notions for program construction: An advanced course*, Nice: Cambridge University Press, 1982.
- Jeary, S., Fouad, A., and Phalp, K., 2008, 30th June - 4th July 2008. *Extending the Model Driven Architecture with a pre-CIM level*. Paper presented at the 1st International Workshop on Business Support for MDA (MDABIZ), co-located with Tools Europe, Zurich, Switzerland.
- Johnson, W. L., 1987, 17 - 19th November 1986. *Specification via scenarios and views*. Paper presented at the 3rd International Software Process Workshop, Colorado, USA.
- Johnson, W. L., 1988. *Deriving specifications from requirements*. Paper presented at the 10th international conference on Software engineering, Singapore.
- Jos, W., and Anneke, K., 2003. *The Object Constraint Language: Getting your models ready for MDA*. Addison-Wesley Longman Publishing Co., Inc.
- Jouault, F., and Kurtev, I., 2006. *On the architectural alignment of ATL and QVT*. Paper presented at the 2006 ACM symposium on Applied computing, Dijon, France.
- Kabanda, S., and Adigun, M., 2006. *Extending Model Driven Architecture benefits to requirements engineering*. Paper presented at the 2006 annual research conference of the South African institute of computer scientists and information technologists on IT research in developing countries, Somerset West, South Africa.
- Kaiser, G. E., 1988, 11 - 13th May 1988. *Rule-based modelling of the software development process*. Paper presented at the 4th International Software Process Workshop, Moretonhampstead, Devon, UK.
- Kanyaru, J. M., 2006. *Requirements validation with enactable descriptions of use cases*. Thesis (Ph.D.). Bournemouth University, Bournemouth.
-

-
- Kanyaru, J. M., Coles, M., Jeary, S., and Phalp, K., 2008a, 3rd July 2008. *Using visualisation to elicit domain information as part of the Model Driven Architecture approach*. Paper presented at the 1st International Workshop on Business Support for MDA (MDABIZ), co-located with Tools Europe, Zurich, Switzerland.
- Kanyaru, J. M., Jeary, S., Coles, M., Phalp, K., and Vincent, J., 2008b. *Assessing graphical user interfaces in modelling tools for MDA using cognitive dimensions*. Paper presented at the Software Quality Management 2008, University of Ulster, Newtownabbey, Northern Ireland.
- Kanyaru, J. M., and Phalp, K., 2005, 29th August - 2nd September, 2005. *Aligning business process models with specifications using enactable use case tools*. Paper presented at the 1st International Workshop on Requirements Engineering for Business Need and IT Alignment (REBNITA), part of the 13th IEEE International Requirements Engineering Conference (RE), Sorbonne, Paris, France.
- Kanyaru, J. M., and Phalp, K., 2009. Validating software requirements with enactable use case descriptions. 14 (1), 1-14.
- Kappelman, L. A., Mckeeman, R., and Zhang, L., 2006. Early warning signs of it project failure: The dominant dozen. *Information Systems Management: IT Project Management*, 23 (4), 31 - 36.
- Karow, M., and Gehlert, A., 2006, 4th - 6th August 2006. *On the transition from Computation Independent to Platform Independent Models*. Paper presented at the 12th Americas Conference on Information Systems (AMCIS), Acapulco, Mexico.
- Katayama, T., 1988, 11 - 13th May 1988. *A hierarchical and functional approach to software process description*. Paper presented at the 4th International Software Process Workshop, Moretonhampstead, Devon, UK.
- Kavakli, E., 2004. *Modelling organizational goals: Analysis of current methods*. Paper presented at the 2004 ACM symposium on Applied computing, Nicosia, Cyprus.
- Kavis, M., 2008. CERN leverages BPMS tools to become more efficient. *BPMInstitute.org*. Available from: <http://www.bpminstitute.org/articles/article/article/cern-leverages-bpms-tools-to-become-more-efficient.html> [Accessed: 9th June 2008].
- Kellner, M. I., 1989, 10 - 13th October 1989. *Software process modelling example*. Paper presented at the 5th International Software Process Workshop, Kennebunkport, Maine, USA.
- Kellner, M. I., 1991, 21st - 22nd October 1991. *Software process modelling support for management planning and control*. Paper presented at the 1st International Conference on the Software Process: Manufacturing Complex Systems, Redondo Beach, California, USA.
- Kemsley, S., 2006. A short history of BPM, part 1 - column 2. *ebizQ: The Insider's Guide to Business and IT Agility*. Available from: http://www.ebizq.net/blogs/column2/2006/05/a_short_history.php [Accessed: 15th February 2008].
- Kherraf, S., Lefebvre, E., and Suryn, W., 2008, 25 - 28 March 2008. *Transformation from CIM to PIM using patterns and archetypes*. Paper presented at the 19th Australian Conference on Software Engineering (ASWEC), Perth, Western Australia.
- Kim, H.-K., 2008. Modelling of distributed systems with SOA & MDA. *IAENG International Journal of Computer Science*, 35 (4).
- King, N., 2006. Using templates in the thematic analysis of text. In: Cassell, C., and Symon, G. eds. *Essential guide to qualitative methods in organisational research*. London, England: SAGE Publications Ltd., 256 - 270.
- Kleppe, A. G., Warmer, J., and Bast, W., 2003. *MDA explained: The Model Driven Architecture: Practice and promise*. Addison-Wesley Longman Publishing Co., Inc.
- Koch, N., 2006. *Transformation techniques in the model-driven development process of UWE*. Paper presented at the 6th international conference on Web engineering, Palo Alto, California.
-

-
- Koch, N., Zhang, G., and Escalona, M. J., 2006. *Model transformations from requirements to web system design*. Paper presented at the 6th international conference on Web engineering, Palo Alto, California, USA.
- Koehler, J., Gschwind, T., Küster, J., Pautasso, C., Ryndina, K., Vanhatalo, J., and Völzer, H., 2007. *Combining quality assurance and model transformations in business-driven development*. Paper presented at the 3rd International Workshop and Symposium on Applications of Graph Transformation with Industrial Relevance (AGTIVE), Kassel, Germany.
- Koehler, J., Tirenni, G., and Kumaran, S., 2002. *From business process model to consistent implementation: A case for formal verification methods*. Paper presented at the 6th International Enterprise Distributed Object Computing Conference.
- Kokune, A., Mizuno, M., Kadoya, K., and Yamamoto, S., 2005, 29th August - 2nd September, 2005. *A fact based collaboration modelling and its application*. Paper presented at the 1st International Workshop on Requirements Engineering for Business Need and IT Alignment (REBNITA), part of the 13th IEEE International Requirements Engineering Conference 2005 (RE), Sorbonne, Paris, France.
- Kokune, A., Mizuno, M., Kadoya, K., and Yamamoto, S., 2007. FBCM: Strategy modelling method for the validation of software requirements. *Journal of Systems and Software*, 80 (3), 314 - 327.
- Kolb, D. A., Rubin, I. M., and McIntyre, J. M., 1979. *Organizational psychology: An experiential approach*. London: Prentice Hall.
- Korhonen, J. J., 2008. Business process management. *SoberIT - Software Business and Engineering Institute, Helsinki University of Technology*. Available from: http://jannekorhonen.fi/Business_Process_Management.pdf [Accessed: 16th March 2011].
- Kovse, J., and Härder, T., 2002. *Generic XMI-based UML model transformations*. Paper presented at the 8th International Conference on Object-Oriented. Information Systems.
- Kusel, A., Schwinger, W., Wimmer, M., and Retschitzegger, W., 2009. *Common pitfalls of using QVT relations - graphical debugging as remedy*. Paper presented at the 14th IEEE International Conference on Engineering of Complex Computer Systems.
- Lahote, D., 2008. Q & A on an introduction to the key concepts of Lean. *Lean Enterprise Institute*. Available from: <http://www.lean.org/Community/Registered/ArticleDocuments/Key%20Concepts%20of%20Lean%20QA%20LaHote%20DRAFT2.pdf> [Accessed: 12th September 2008].
- Lautenbacher, F., Sieber, T., Cabral, A., and Bauer, B., 2007, October 21 - 25. *Linguistic modelling methods and ontologies in requirements engineering*. Paper presented at the The International Workshop on Semantic-Based Software Development at OOPSLA, Montreal, Canada.
- Lavagno, L., and Mueller, W. 2006. UML as a next generation language for SoC design: IMMOS - Integrated Method for the Model-based Development of Automotive Control Units.
- Ledru, Y., Laleau, R., Lemoine, M., Vignes, S., Bert, D., Donzeau-Gouge, V., Dubois, C., and Peureux, F., 2006, June 2006. *An attempt to combine UML and formal methods to model airport security*. Paper presented at the CAISE Forum 2006 - Proceedings of the Forum of the 18th International Conference on Advanced Information Systems Engineering, Presses universitaires de Namur, Luxembourg.
- Lehman, M. M., 1988, April 1988. *Some reservations on software process programming*. Paper presented at the 4th international software process workshop on Representing and enacting the software process, Devon, United Kingdom.
- Lehman, M. M., 1989, 10 - 13th October 1989. *The role of process models in software and systems development and evolution*. Paper presented at the 5th International Software Process Workshop, Kennebunkport, Maine, USA.
-

-
- Leonardi, M. C., and Mauco, M. V., 2004, 9 - 10th December 2004. *Integrating natural language oriented requirements models into MDA*. Paper presented at the Anais do WER04 - Workshop em Engenharia de Requisitos, Tandil, Argentina.
- Lester, S., 1999. An introduction to phenomenological research. *Stan Lester Developments*. Available from: <http://www.sld.demon.co.uk/resmethy.pdf> [Accessed: 21st August 2010].
- Lombardi. 2008. Lombardi white paper: How to structure your first BPM project to avoid disaster. Available from: http://www.lombardisoftware.com/downloads/WP_Structuring_Your_First_BPM_Project.pdf [Accessed: 15th June 2009].
- Lubbe, S., 2003. The development of a case study methodology in the Information Technology (IT) field: A step by step approach. 2003 (September), 2 - 2.
- Macek, O., and Richta, K., 2009. *The BPM to UML activity diagram transformation using XSLT*. Paper presented at the Annual International Workshop on Databases, Texts, Specifications, Objects (DATESO), Spindleruv Mlyn, Czech Republic.
- Maguire, M., and Bevan, N., 2002, 25 - 29th August, 2002. *User requirements analysis: A review of supporting methods*. Paper presented at the IFIP 17th World Computer Congress - TC13 Stream on Usability: Gaining a Competitive Edge, Montréal, Québec, Canada.
- Mahmudi, J., and Tavakkoli, V., 2005. *Simulation: The best solution for BPR*. Paper presented at the MSSANZ International Congress on Modelling and Simulation (MODSIM), Melbourne University, Victoria, Australia.
- Marsland, N., Wilson, I., Abeyasekera, S., and Kleih, U. 1998. A methodological framework for combining quantitative and qualitative survey methods, *An output from the DFID-funded Natural Resources Systems Programme, project R7033*: Social and Economic Development Department, Natural Resources Institute and the Statistical Services Centre, The University of Reading.
- Martin, A., and Loos, P., 2008, 30th June - 4th July 2008. *Software support for the computation independent modelling in the MDA context*. Paper presented at the 1st International Workshop on Business Support for MDA (MDABIZ), co-located with Tools Europe, Zurich, Switzerland.
- Massoni, T., Gheyi, R., and Borba, P., 2004. *A UML class diagram analyser*. Paper presented at the 3rd International Workshop on Critical Systems Development with UML, Lisbon, Portugal.
- Mattsson, A., Lundell, B., Lings, B., and Fitzgerald, B., 2009. Linking model-driven development and software architecture: A case study. *IEEE Transactions on Software Engineering*, 35 (1), 83 - 93.
- May, L. J., 1998. Major causes of software project failures. *Crosstalk - The Journal Of Defence Software Engineering*, 9 - 12.
- McNeile, A., 2003. MDA: The vision with the hole? Available from: <http://www.metamaxim.com/download/documents/MDAv1.pdf> [Accessed: 25th October 2007].
- Mellor, S. J. 2004. Agile MDA (pp. 1-9): Project Technology, Inc.
- Mellor, S. J., and Balcer, M., 2002. *Executable UML: A foundation for model-driven architectures*. Addison-Wesley Longman Publishing Co., Inc.
- Menzel, C., and Mayer, R. J., 1998. *The IDEF family of languages*. Paper presented at the Handbook on Architectures for Information Systems.
- Meservy, T. O., and Fenstermacher, K. D., 2005. Transforming software development: An MDA road map. *Computer*, 38, 52 - 58.
- Murdoch, J., and McDermid, J. A., 2000. Modelling engineering design processes with Role Activity Diagrams. *J. Integr. Des. Process Sci.*, 4 (2), 45-65.
- Musschoot, T., 2009. Enterprise BPM. *BPMInstitute.org White Paper*. Available from: <http://www.bpmstitute.org/whitepapers/whitepaper/article/enterprise-BPM.html> [Accessed: 18th June 2009].
-

-
- Musschoot, T., 2010. Business / IT collaboration model: A practical approach. *BPMInstitute.org*. Available from: <http://www.bpminstitute.org/articles/article/article/business-it-collaboration-model-a-practical-approach.html> [Accessed: 8th April 2010].
- Mylopoulos, J., 2004a. III. Structured analysis and design technique. *Lecture Notes on Conceptual Modelling*. Available from: <http://www.cs.toronto.edu/~jm/2507S/Notes04/SADT.pdf> [Accessed: 20th July 2009].
- Mylopoulos, J., 2004b. Lecture notes on conceptual modelling - formal requirements modelling languages. *Department of Computer Science, the University of Toronto* Available from: <http://www.cs.toronto.edu/~jm/2507S/Notes04/FormalRML.pdf> [Accessed: 18th March 2011].
- Nakagawa, A. T., and Futatsugi, K., 1989, 10 - 13th October 1989. *Product-based process models*. Paper presented at the 5th International Software Process Workshop, Kennebunkport, Maine, USA.
- Niehaves, B., and Stirna, J., 2006. *Participative enterprise modelling for balanced scorecard implementation*. Paper presented at the 14th European Conference on Information Systems (ECIS), Göteborg, Sweden.
- Nuseibeh, B., and Easterbrook, S., 2000. *Requirements engineering: A roadmap*. Paper presented at the Conference on The Future of Software Engineering, Limerick, Ireland.
- Nuseibeh, B., Finkelstein, A., and Kramer, J., 1993, 6 - 7th December 1993. *Fine-grain process modelling*. Paper presented at the 7th international workshop on Software specification and design, Redondo Beach, California.
- O'brien, R., 2001. An overview of the methodological approach of action research. In: Richardson, R. ed. *Theory and Practice of Action Research*, João Pessoa, Brazil: Universidade Federal da Paraíba.
- Odeh, M., Beeson, I., Green, S., and Sa, J., 2002, 16th - 19th December, 2002. *Modelling processes using rad and UML activity diagrams : An exploratory study*. Paper presented at the 3rd International Arab Conference on Information Technology (ACIT), University of Qatar, Qatar.
- Ohata, A., and Butts, K. R., 2005. *Towards a concurrent engine system design methodology*. Paper presented at the American Control Conference, Portland, Oregon, USA.
- Oliveira, T. C., Filho, I. M., Lucena, C. J. P. D., Alencar, P., and Cowan, D. D., 2004, September 7th-8th 2004. *Enabling model driven product line architectures*. Paper presented at the 2nd European Workshop on Model Driven Architecture (EWMDA), Canterbury, UK.
- OMG. 2003a. Common Warehouse Metamodel™ (CWM™) specification, v1.1: Object Management Group.
- OMG. 2003b. MDA guide version 1.0.1: Object Management Group.
- OMG. 2005. BPMI.Org and OMG announce strategic merger of business process management activities. Available from: <http://www.omg.org/news/releases/pr2005/06-29-05.htm> [Accessed: 21st January 2009].
- OMG. 2006a. Meta Object Facility core specification version 2.0: Object Management Group.
- OMG. 2006b. Object Constraint Language specification, version 2.0: Object Management Group.
- OMG. 2007a. Committed companies and their products. Available from: <http://www.omg.org/MDA/committed-products.htm> [Accessed: 25th October 2007].
- OMG. 2007b. MOF 2.0 / XMI mapping specification, v2.1.1: Object Management Group.
- OMG. 2007c. Unified Modelling Language (UML), version 2.1.2.
- OMG. 2008a. Business Process Modelling Notation, v1.1: Object Management Group.
- OMG. 2008b. Meta Object Facility (MOF) 2.0 Query/View/Transformation specification, version 1.0: Object Management Group.
-

-
- OMG. 2010. Success stories. Available from: http://www.omg.org/mda/products_success.htm [Accessed: 26th August 2011].
- Osis, J., Asnina, E., and Grave, A., 2007, 23-25 April, 2007. *Formal Computation Independent Model of the problem domain within the MDA*. Paper presented at the 10th International Conference on Information System Implementation and Modelling (ISIM), Hradec nad Moravicí, Czech Republic.
- Osterweil, L., 1987. *Software processes are software too*. Paper presented at the 9th international conference on Software Engineering, Monterey, California, United States.
- Ould, M., 2004a. RIVA: A rigorous approach for business process management. Available from: <http://www.veniceconsulting.co.uk/riva.pdf> [Accessed: 15th November 2009].
- Ould, M. A., 2003, January 2003. *Incremental process deployment*. Paper presented at the Process Modelling Workshop, Bristol.
- Ould, M. A., 2004b. All the world's a stage. *BPTrends*. Available from: http://www.bptrends.com/deliver_file.cfm?fileType=publication&fileName=11%2D04%20ART%20Worlds%20a%20Stage%20%2D%20Ould1%2Epdf [Accessed: 6th December 2009].
- Ould, M. A., 2004c. *Business process management: A rigorous approach*. British Computer Society.
- Ould, M. A., 2004d. Getting your head round mozzarella. *BPTrends*. Available from: http://www.bptrends.com/deliver_file.cfm?fileType=publication&fileName=09%2D04%20ART%20Mozzarella%20Ould%2Epdf [Accessed: 6th December 2009].
- Ould, M. A., 2004e. Getting your head round spaghetti. *BPTrends*. Available from: http://www.bptrends.com/deliver_file.cfm?fileType=publication&fileName=10%2D04%20ART%20Spaghetti%20%2D%20Ould%2Epdf [Accessed: 6th December 2009].
- Ould, M. A., 2005. Business process management: A rigorous approach. *BPTrends*. Available from: <http://www.businessprocesstrends.com/publicationfiles/05%2D05%20ART%20BPM%20A%20Rigorous%20App%20%2D%20Ould%2Epdf> [Accessed: 3rd March 2008].
- Ould, M. A., 2006. *Getting your head around your business processes*. Paper presented at the Royal Holloway, IEEE & UCL Joint Event, London, England.
- Ould, M. A., and Roberts, C., 1987, 17 - 19th November 1986. *Modelling iteration in the software process*. Paper presented at the 3rd International Software Process Workshop, Colorado, USA.
- Owen, C. L., 2007. Covering user needs. *BPMInstitute.org*. Available from: <http://www.bpminstitute.org/articles/article/article/covering-user-needs.html> [Accessed: 24th June 2009].
- Owen, C. L., 2009a. The power of abstraction. *BPMInstitute.org*. Available from: <http://www.bpminstitute.org/articles/article/article/the-power-of-abstraction.html> [Accessed: 13th November 2009].
- Owen, C. L., 2009b. The systems viewpoint. *BPMInstitute.org*. Available from: <http://www.bpminstitute.org/articles/article/article/the-systems-viewpoint.html> [Accessed: 2nd June 2009].
- Peixoto, D. C. C., Batista, V. A., Atayde, A. P., Borges, E. P., Resende, R. S. F., and Pádua, C. I. P. S., 2008, 2nd to 6th June, 2008. *A comparison of BPMN and UML 2.0 activity diagrams*. Paper presented at the 7th Simpósio Brasileiro de Qualidade de Software, Florianópolis, Brazil.
- Peltier, M., Ziserman, F., and Bézin, J., 2000. *On levels of model transformation*. Paper presented at the XML Europe, Paris, France.
- Perry, D. E., Sim, S. E., and Easterbrook, S. M., 2004, 23 - 28 May 2004. *Case studies for software engineers*. Paper presented at the 26th International Conference on Software Engineering (ICSE'04), Edinburgh, Scotland.
-

-
- Perry, S., 2006, 7th March 2006. *When is a process model not a process model - a comparison between UML and BPMN*. Paper presented at the IEE Seminar on Process Modelling Using UML, London.
- Phalp, K., 2002, September. *Surrounding the requirements process*. Paper presented at the ESERG Workshop on Software Engineering, Bournemouth University.
- Phalp, K., Adlem, A., Jeary, S., Vincent, J., and Kanyaru, J., 2011. The role of comprehension in requirements and implications for use case descriptions. *Software Quality Journal*, 19 (2), 461 - 486.
- Phalp, K., and Cox, K., 2001, August 27th-29th, 2001. *Guiding use case driven requirements elicitation and analysis*. Paper presented at the 7th International Conference on Object-Oriented Information Systems (OOIS), Calgary, Canada.
- Phalp, K., and Shepperd, M., 1994, February 1994. *A pragmatic approach to process modelling*. Paper presented at the 3rd European Workshop on Software Process Technology (EWSPT'94) Villard de Lans, France.
- Phalp, K. T., 1998. The CAP framework for business process modelling. *Information and Software Technology*, 40 (13), 731 - 744.
- Phalp, K. T., Henderson, P., Walters, R. J., and Abeysinghe, G., 1998. Rolenact: Role-based enactable models of business processes. *Information and Software Technology*, 40 (3), 123 - 133.
- Phalp, K. T., and Jeary, S., 2010, 29 - 31 March 2010. *An empirical investigation of the utility of 'pre-CIM' models*. Paper presented at the Software Quality XVIII, 18th International Software Quality Management Conference (SQM), London, England.
- Phalp, K. T., Jeary, S., Vincent, J., Kanyaru, J. M., and Crowle, S., 2007, 1 - 2 August 2007. *Supporting stakeholders in the MDA process*. Paper presented at the Software Quality Management / INSPIRE 2007, University of Tampere, Finland.
- Poernomo, I., Tsaramirsis, G., and Zuna, V., 2008, 30th June - 4th July 2008. *A methodology for requirements analysis at CIM level*. Paper presented at the 1st International Workshop on Business Support for MDA (MDABIZ), co-located with Tools Europe, Zurich, Switzerland.
- Purper, C. B., 2000, February 2000. *Transcribing process model standards into meta-processes*. Paper presented at the 7th European Workshop on Software Process Technology (EWSPT) Kaprun, Austria.
- Pyke, J., 2006. Why workflow sucks. *ebizQ*. Available from: http://www.ebizq.net/hot_topics/BPM/features/7462.html [Accessed: 15th September 2008].
- Raffo, D., Kaltio, T., Partridge, D., Phalp, K., and Ramil, J. F., 1999. Empirical studies applied to software process models. *Working Group Report: ICSE Workshop on Empirical Studies of Software Development and Evolution, Empirical Software Engineering Journal*, 4 (4), 353 - 369.
- Rech, J., and Schmitt, M., 2008, 30th June - 4th July 2008. *Embedding defect and traceability information in CIM- and PIM-level software models*. Paper presented at the 1st International Workshop on Business Support for MDA (MDABIZ), co-located with Tools Europe, Zurich, Switzerland.
- Recker, J., 2006. Process modelling in the 21st century. *BPTrends*. Available from: <http://www.bptrends.com/publicationfiles/05-06-ART-ProcessModeling21stCent-Recker1.pdf> [Accessed: 3rd February 2011].
- Rivkin, W., 2008. Closing the business-it gap once and for all. *BPMInstitute.ORG*. Available from: <http://www.bpm institute.org/articles/article/article/closing-the-business-it-gap-once-and-for-all.html> [Accessed: 28th January 2009].
- Roberts, C., 1988, 11 - 13th May 1988. *Describing and acting process models with PML*. Paper presented at the 4th International Software Process Workshop, Moretonhampstead, Devon, UK.
- Robinson, W. N., 2007. *Extended OCL for goal monitoring*. Paper presented at the Ocl4All: Modelling Systems with OCL at MoDELS, Nashville, TN, USA.
-

-
- Rodriguez, A., Fernandez-Medina, E., and Piattini, M., 2007a, 24-28 September 2007. *Towards CIM to PIM transformation: From secure business processes defined by BPMN to use cases*. Paper presented at the 5th International Conference On Business Process Management, Brisbane, Australia.
- Rodriguez, A., Fernández-Medina, E., and Piattini, M., 2007b, 11 - 15th June 2007. *Using QVT to obtain use cases from secure business processes modelled with BPMN*. Paper presented at the 8th Workshop on Business Process Modelling, Development, and Support (BPMS), In conjunction with CAiSE, Trondheim, Norway.
- Rombach, H. D., 1988, 11 - 13th May 1988. *A specification framework for sw processes: Formal specification & derivation of information base requirements*. Paper presented at the 4th International Software Process Workshop, Moretonhampstead, Devon, UK.
- Rothman, J., 2007, 16th January 2007. *Behind closed doors: Secrets of great management*. Paper presented at the Boston SPIN meeting (Software Process Improvement Network).
- Russell, N. C., 2007. *Foundations of process-aware information systems*. Thesis (PhD). Queensland University of Technology, Brisbane, Australia.
- Sa, J., and Warboys, B. C., 1994, February 1994. *Modelling processes using a stepwise refinement technique*. Paper presented at the 3rd European Workshop on Software Process Technology (EWSPT) Villard de Lans, France.
- Saeki, M., Kaneko, T., and Sakamoto, M., 1991, 21st - 22nd October 1991. *A method for software process modelling and description using LOTOS*. Paper presented at the 1st International Conference on the Software Process: Manufacturing Complex Systems, Redondo Beach, California, USA.
- Saunders, M., Thornhill, A., and Lewis, P., 2003. *Research methods for business students*. Third ed. Harlow, UK: Prentice Hall.
- Scacchi, W., 2002. *Process models in software engineering*. Paper presented at the Encyclopaedia of Software Engineering, New York, NY, USA.
- Sendall, S., and Kozaczynski, W., 2003. Model transformation: The heart and soul of model-driven software development. *IEEE Software*, 20 (5), 42 - 45.
- Shannak, R. O., 2009. Grounded theory as a methodology for theory generation in information systems research. *European Journal of Economics, Finance and Administrative Sciences* (15), 32 - 50.
- Sheena, R. J., Doris, L. C., and Robert, B. F., 2003. *A metamodelling approach to model transformation*. Paper presented at the 18th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications, Anaheim, CA, USA.
- Shelmerdine, S., 2010. Qualitative research methods. *Lecture Notes*. Available from: <http://web.uct.ac.za/depts/psychology/webie/courses/psy206fo/Lecture%201.ppt> [Accessed: 21st August 2010].
- Shneiderman, B., 2002. *Leonardo's laptop: Human needs and the new computing technologies*. London, England: MIT Press.
- Silver, B., 2008a. BPMS watch: BPMN's three levels, reconsidered. *BPMInstitute.org*. Available from: <http://www.bpmoinstitute.org/articles/article/article/bpms-watch-BPMN-s-three-levels-reconsidered.html> [Accessed: 30th January 2009].
- Silver, B., 2008b. BPMS watch: Bringing method to the madness. *BPMInstitute.org*. Available from: <http://www.bpmoinstitute.org/articles/article/article/bpms-watch-bringing-method-to-the-madness.html> [Accessed: 10 November 2008].
- Silver, B., 2008c. What's wrong with this picture, redux. *Bruce Silver's blog on business process management*. Available from: <http://www.brsilver.com/wordpress/2006/11/29/whats-wrong-with-this-picture-redux/> [Accessed: 23rd January 2009].
-

-
- Silver, B., 2008d. What's wrong with this picture? *Bruce Silver's blog on business process management*. Available from: <http://www.brsilver.com/wordpress/2006/09/06/whats-wrong-with-this-picture/> [Accessed: 25th January 2009].
- Silver, B., 2008e. What's wrong with this picture? Part 2. *Bruce Silver's blog on business process management*. Available from: <http://www.brsilver.com/wordpress/2006/09/14/whats-wrong-with-this-picture-part-2/> [Accessed: 22nd January 2009].
- Silver, B., 2008f. What's wrong with this picture? Part 3. *Bruce Silver's blog on business process management*. Available from: <http://www.brsilver.com/wordpress/2006/09/19/whats-wrong-with-this-picture-part-3/> [Accessed: 22nd January 2009].
- Silver, B., 2009a. Bpms watch: Five things they left out of BPMN 2.0. *BPMInstitute.org*. Available from: <http://www.bpmoinstitute.org/articles/article/article/bpms-watch-five-things-they-left-out-of-BPMN-2-0.html> [Accessed: 11th June 2009].
- Silver, B., 2009b. Bpms watch: Five things to love about BPMN 2.0. *BPMInstitute.org*. Available from: <http://www.bpmoinstitute.org/articles/article/article/bpms-watch-five-things-to-love-about-BPMN-2-0.html> [Accessed: 8th June 2009].
- Sinnott, R. O., and Turner, K. J., 1994, October 1994. *Modelling ODP viewpoints*. Paper presented at the International Conference on Object Oriented Programming, Systems, Languages and Applications Workshop on Precise Behavioural Specifications in Object-Oriented Information Modelling (OOPSLA), Portland, Oregon, USA.
- Skalle, H., 2009. Round table: How to combine Lean Six Sigma, SOA & BPM to deliver real business results. *BPMInstitute.org*. Available from: <http://www.bpmoinstitute.org/roundtables/upcoming-round-table/article/how-to-combine-lean-six-sigma-soa-BPM-to-deliver-real-business-results.html> [Accessed: 27th January 2009].
- Slack, S. E., 2008. The business analyst in model-driven architecture: Separating design from architecture. 1 - 8. Available from: <http://download.boulder.ibm.com/ibmdl/pub/software/dw/architecture/ar-bamda/ar-bamda-pdf.pdf> [Accessed: 29th October 2009].
- Smith, H., and Fingar, P., 2003. *Business process management: The third wave*. First ed.: Meghan-Kiffer Press.
- Soley, R. M., 2006. Modelling all the way up... Modelling all the way down. Available from: <http://www.omg.org/MDA/MDA-webcasts.htm> [Accessed: 21st November 2007].
- Sommerville, I., 2004. Requirements engineering processes. In: *Software engineering* Seventh ed. London, England: Addison Wesley.
- Spinellis, D., 2010. UML everywhere. *IEEE Software*, 27 (5), 90 - 91.
- Starke, G., 1994, February 1994. *Why is process modelling so difficult?* Paper presented at the 3rd European Workshop on Software Process Technology (EWSPT) Villard de Lans, France.
- Staron, M., and Wohlin, C., 2006. *An industrial case study on the choice between language customization mechanisms*. Vol. 4034/2006. Berlin, ALLEMAGNE: Springer.
- Stevens, P., and Pooley, R., 2000. *Using UML: Software engineering with objects and components*. Updated ed. Harlow, Essex, England: Pearson Education Limited.
- Stevens, W., Myers, G., and Constantine, L., 1974. Structured design. *IBM Systems Journal*, 13 (2), 115 - 139.
- STSC. 2003. Guidelines for successful acquisition and management of software-intensive systems: Weapon systems, command and control systems, management information systems. *Chapter 4: Requirements Management*, Condensed Version 4.0, 3 - 10. Available from: http://www.stsc.hill.af.mil/resources/tech_docs/gsam4.html [Accessed: 29th June 2009].
-

-
- Subramaniam, K., Far, B. H., and Eberlein, A., 2004, 2 - 5 May 2004. *Automating the transition from stakeholders' requests to use cases in ooad*. Paper presented at the Canadian Conference on Electrical and Computer Engineering (CCECE) Niagara Falls, Canada.
- Sutton, S. M., Ziv, H., Heimbigner, D., Yessayan, H. E., Maybee, M., Osterweil, L. J., and Song, X., 1991, 21st - 22nd October 1991. *Programming a software requirements-specification process*. Paper presented at the 1st International Conference on the Software Process: Manufacturing Complex Systems, Redondo Beach, California, USA.
- Taylor, R. N., 1987, 17 - 19th November 1986. *Concurrency and software process models*. Paper presented at the 3rd International Software Process Workshop, Colorado, USA.
- Thangaraj, S., 2004. Introduction to Model Driven Architecture, ncicb software development processes, facilitating systems interoperability. *Cancer Biomedical Informatics Grid, National Cancer Institute*. Available from: http://cabig.nci.nih.gov/workspaces/ICR/Meetings/ICR_Workspace/August_Face-to-Face_Meeting/F2F_MDA_Intro_Presentation.pdf [Accessed: 2nd October 2007].
- Thiemann, P., 2009. Software engineering Model Driven Architecture applications of metamodelling. *University of Freiburg*. Available from: <http://proglang.informatik.uni-freiburg.de/teaching/swt/2009/v14-meta-app.en.pdf> [Accessed: 6th December 2009].
- Thom, L. H., Iochpe, C., and Reichert, M., 2007, June 2007 *Workflow patterns for business process modelling*. Paper presented at the 8th International Workshop on Business Process Modelling, Development, and Support (BPMDS), Trondheim, Norway.
- Thomas, D., 2004. MDA: Revenge of the modellers or UML utopia? *IEEE Software*, 21 (3), 15 -17.
- Thomas, I., 1989, 10 - 13th October 1989. *The software process as a goal-directed activity*. Paper presented at the 5th International Software Process Workshop, Kennebunkport, Maine, USA.
- Tratt, L., 2005. Model transformations and tool integration. *Journal of Software and Systems Modelling*, 4 (2), 112 - 122.
- Tully, C. J., 1987, 17 - 19th November 1986. *Software process models and iteration*. Paper presented at the 3rd International Software Process Workshop, Colorado, USA.
- Turgeon, J., and Madhavji, N. H., 2000a, 10 - 12 July. *A model for process congruence*. Paper presented at the International Workshop on Feedback and Evolution in Software and Business Processes (FEAST 2000), Imperial College, London.
- Turgeon, J., and Madhavji, N. H., 2000b, February 2000. *View-based vs. Traditional modelling approaches: Which is better?* Paper presented at the 7th European Workshop on Software Process Technology (EWSPT 2000) Kaprun, Austria.
- UC4. 2008. Beyond job scheduling: The road to enterprise process automation. Available from: <http://erp.ittoolbox.com/research/beyond-job-scheduling-the-road-to-enterprise-process-automation-6204> [Accessed: 4th June 2009].
- Uhl, A., and Ambler, S. W., 2003. Point/counterpoint. *IEEE Software*, 20 (5), 70-73.
- VIDE. 2007. Visualise all model driven programming (VIDE) - the visual user interface (pp. 206): Deliverable 5.1 of the VIDE Project.
- VIDE. 2008a. Visualise all model driven programming (VIDE) - industrial use cases (pp. 78): Deliverable D11.1 of the VIDE Project.
- VIDE. 2008b. Visualise all model driven programming (VIDE) - vide cookbook (pp. 118): Deliverable D10.a (internal deliverable) of the VIDE Project.
- VIDE. 2009. Visualise all model driven programming (VIDE) - vide final report (pp. 63).
- VIDE. 2010a. Distribution. *Visualise all model driven programming (VIDE)*. Available from: <http://www.vide-ist.eu/reflib/dist.html> [Accessed: 26th July 2010].
-

-
- VIDE. 2010b. E-learning: 1.4 VIDE tool usage scenarios. *Visualise all model driven programming (VIDE)*. Available from: <http://www.vide-ist.eu/reflib/elearning/vide/a/a0005.html> [Accessed: 26th July 2010].
- Wa, L. C., and Leong, C. W. 2004. Object-oriented analysis & design with Unified Modelling Language: E-ticket system, *COMP 2221 SOFTWARE ENGINEERING, Prof. Jiming Liu* (pp. 13). Hong Kong: Department of Computer Science, Hong Kong Baptist University.
- Ward-Dutton, N., 2011. Are you set up to manage process change? *BPMInstitute.org*. Available from: <http://www.bpm-institute.org/articles/article/article/are-you-set-up-to-manage-process-change.html> [Accessed: 28th March 2011].
- Ward-Dutton, N., and Baxter, B., 2009. Beyond model-driven development: Delivering on the promise of BPM. Available from: <http://w.on24.com/r.htm?e=145185&s=1&k=6B4FB27399B8EACD2B59EC16DB26BCEC> [Accessed: 20th May 2009].
- Wegmann, A., Regev, G., and Loison, B., 2005, 29th August - 2nd September, 2005. *Business and IT alignment with SEAM*. Paper presented at the 1st International Workshop on Requirements Engineering for Business Need and IT Alignment (REBNITA), part of the 13th IEEE International Requirements Engineering Conference (RE), Sorbonne, Paris, France.
- Wegmann, A., Regev, G., Rychkova, I., Le, L.-S., and Julia, P., 2007, 15 - 19th October 2007. *Business and IT alignment with SEAM for enterprise architecture*. Paper presented at the 11th IEEE International Enterprise Distributed Object Computing Conference (EDOC), Annapolis, Maryland, USA.
- White, S. A., 2004. Process modelling notations and workflow patterns. *BPTrends*. Available from: http://www.businessprocesstrends.com/deliver_file.cfm?fileType=publication&fileName=03-04%20WP%20Notations%20and%20Workflow%20Patterns%20-%20White.pdf [Accessed: 27th August 2008].
- Wieggers, K. E., 2000. When telepathy won't do: Requirements engineering key practices. *Cutter IT Journal*.
- Williams, L. G., 1988, 11 - 13th May 1988. *A behavioural approach to software process modelling*. Paper presented at the 4th International Software Process Workshop, Moretonhampstead, Devon, UK.
- Willis, P., and Trondman, M., 2002. Manifesto for ethnography. *Cultural Studies, Critical Methodologies*, 2 (3), 394 - 402.
- Wohed, P., Aalst, W. M. P. V. D., Dumas, M., Hofstede, A. H. M. T., and Russell, N., 2005. *Pattern-based analysis of the control-flow perspective of UML activity diagrams*. Paper presented at the ER 2005.
- Wohed, P., Van Der Aalst, W. M. P., Dumas, M., and Ter Hofstede, A. H. M., 2002. Pattern based analysis of BPEL4WS. *QUT Technical report, FIT-TR-2002-04, Queensland University of Technology, Brisbane*. Available from: http://www.workflowpatterns.com/documentation/documents/qut_bpel_rep.pdf [Accessed: 22nd April 2010].
- Wohed, P., Van Der Aalst, W. M. P., Dumas, M., Ter Hofstede, A. H. M., and Russell, N., 2006. Pattern-based analysis of BPMN. *BPM Center Report BPM-06-17, BPMcenter.org, Queensland University of Technology, Brisbane*. Available from: <http://eprints.qut.edu.au/2977/> [Accessed: 21st April 2010].
- Wong, P. Y. H., and Gibbons, J., 2008, 27 - 31 October 2008. *A process semantics for BPMN*. Paper presented at the 10th International Conference on Formal Engineering Methods., Kitakyushu International Conference Centre, Kitakyushu-City, Japan.
- Wood, B., 2002, 27th November 2002. *UML for ODP viewpoint specifications*. Paper presented at the ITU-T/SG17 meeting, Geneva, Switzerland.
- Wood, B., 2005, April 2005. *The use of ODP in MDA system specifications*. Paper presented at the OMG MDA users SIG, Open-IT.
-

-
- Yin, R. K., 1994. *Case study research: Design and methods*. Second ed. Beverly Hills, CA, U.S.A.: Sage Publishing.
- Yin, R. K., 2008. *Case study research: Design and methods*. Fourth ed. Beverly Hills, CA, U.S.A.: Sage Publishing.
- Yoda, T., 2001, 6th December 2001. *Creating applications using parameterized frameworks: Quickly developed and highly customised*. Paper presented at the OMG's 2nd Workshop: UML for Enterprise Applications: Model Driven Solutions for the Enterprise, Burlingame, CA, USA.
- Yourdon, E., 1989. *Modern structured analysis*. Yourdon Press.
- Zave, P., 1989, 10 - 13th October 1989. *Domain understanding and the software process*. Paper presented at the 5th International Software Process Workshop, Kennebunkport, Maine, USA.
- Zhang, J., Feng, P., Wu, Z., Yu, D., and Chen, K. 2008. Activity based CIM modelling and transformation for business process systems: Department of Precision Instruments and Mechanology, Tsinghua University, Beijing, China.
- Zhu, C., Lee, Y., Zhao, W., and Zhang, J. 2006. A feature oriented approach to mapping from domain requirements to product line architecture: Software Engineering Lab, Computer Science and Technology Department, Fudan University, Shanghai, China.

Appendix I

xMDA and the Order Processing Illustration

- Environment, Shared and Machine RAD
- Class Diagram

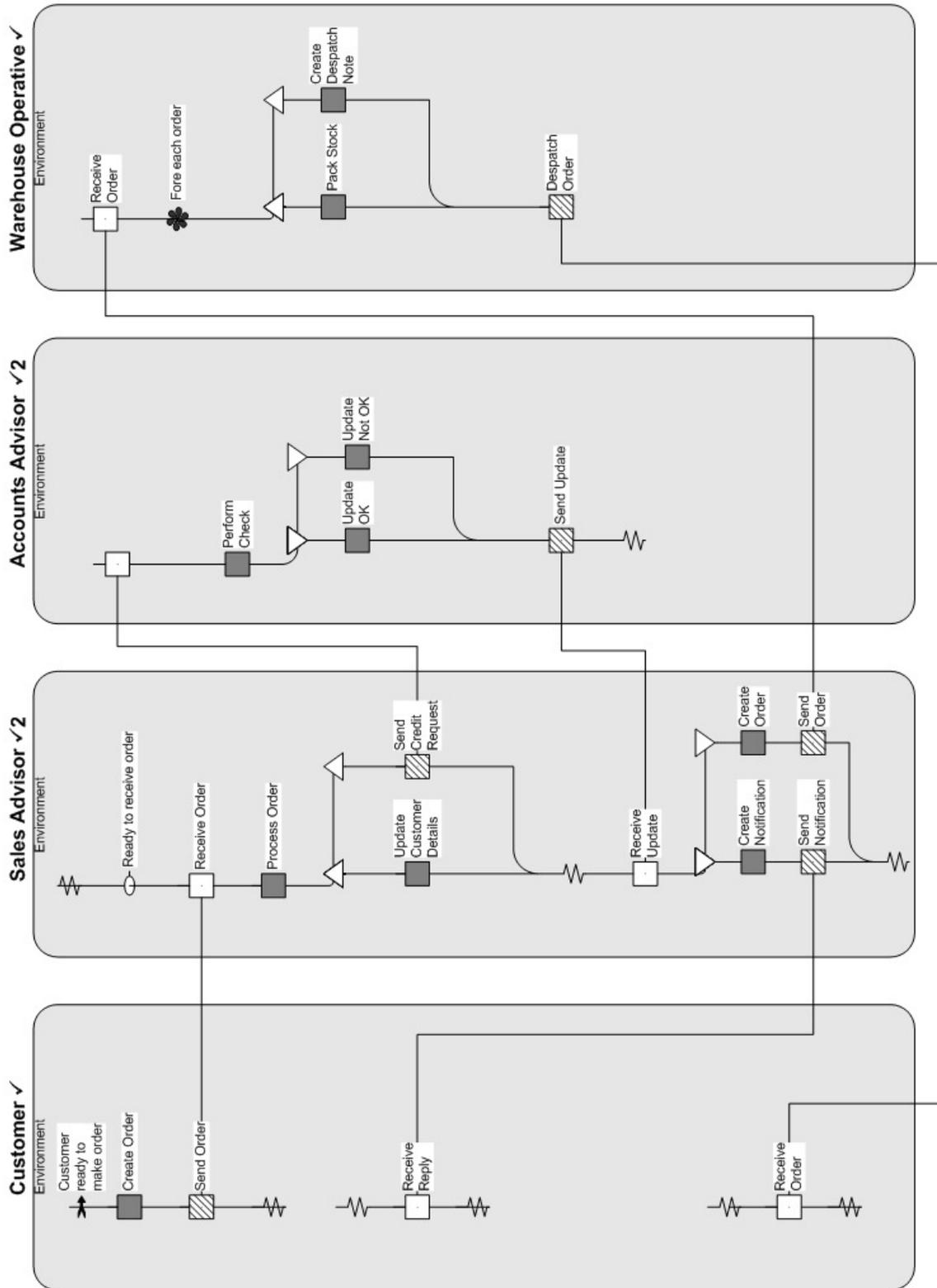


figure 1-1, Environment RAD for the Order Processing example (enlarged).

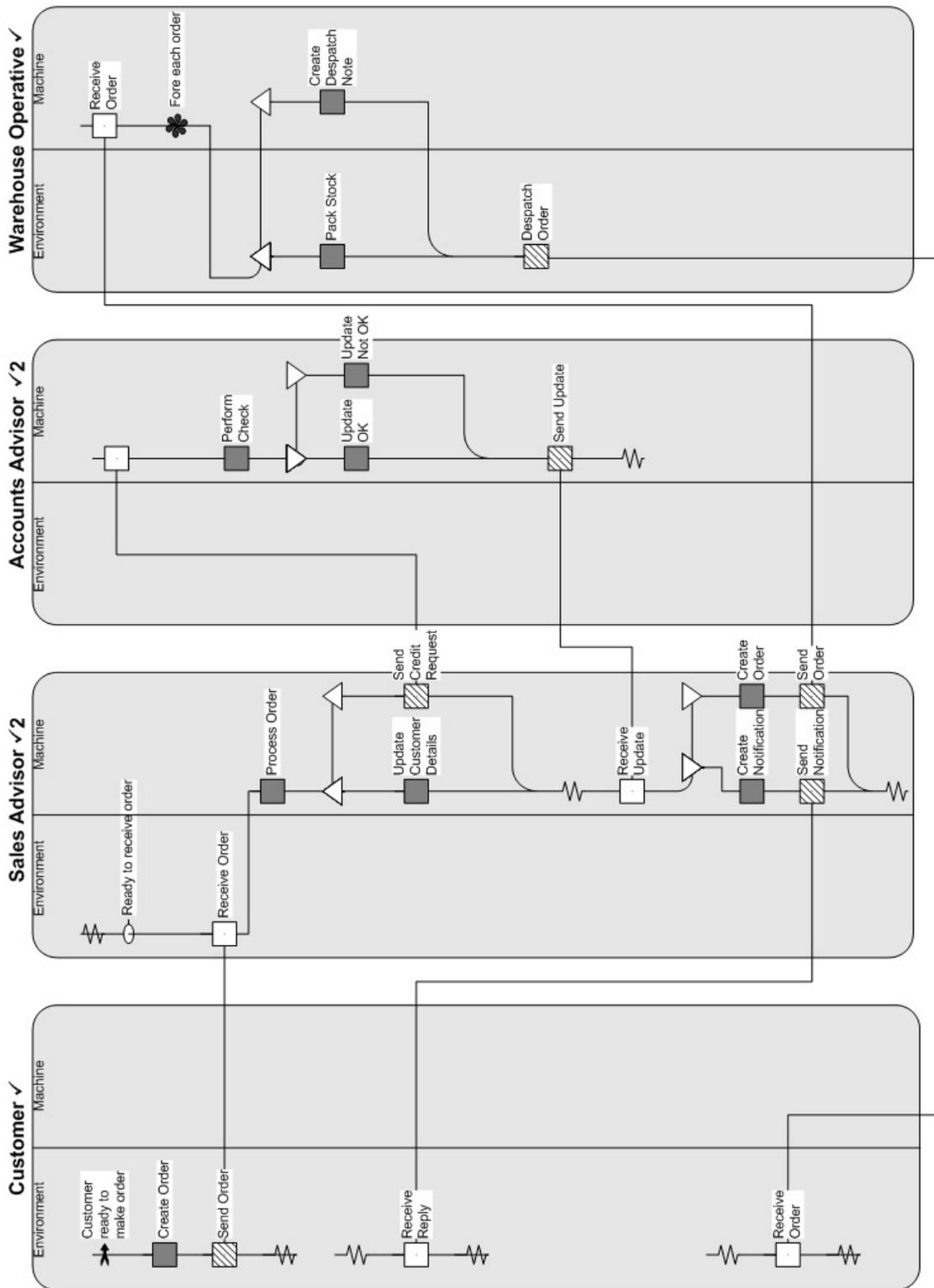


figure 1-2, Shared RAD for the Order Processing example.

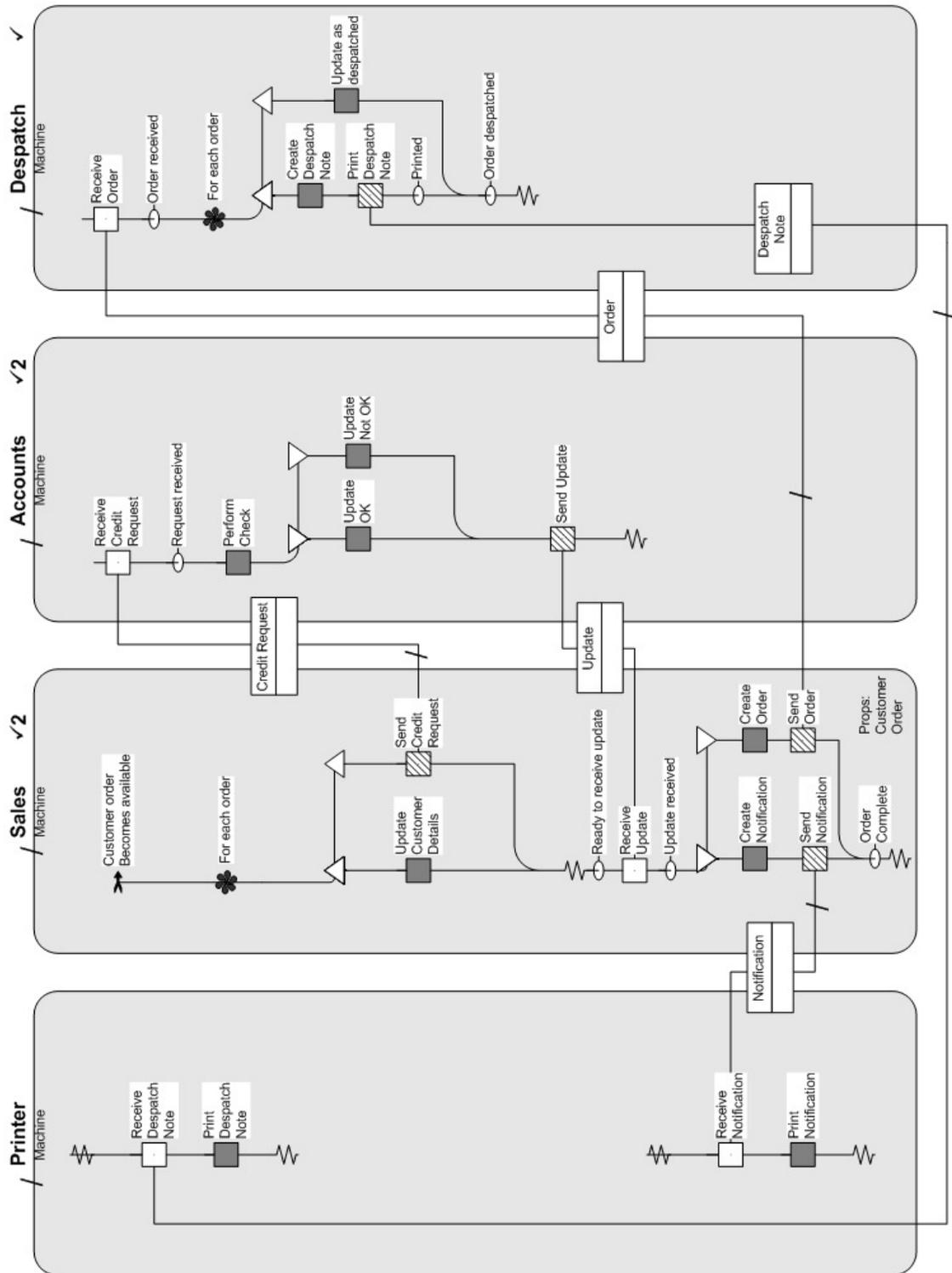


figure 1-3, Machine RAD for the Order Processing example.

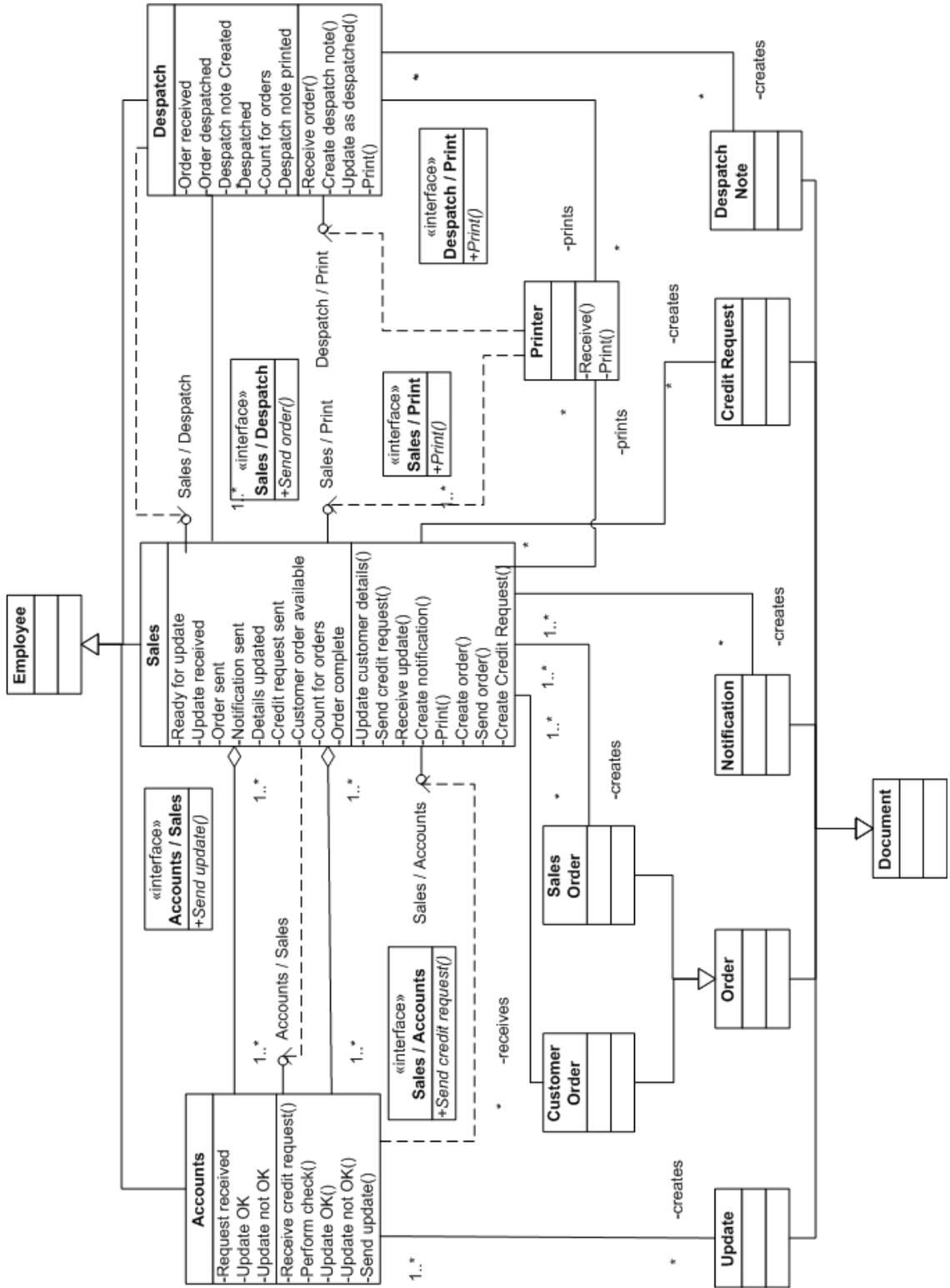


figure 1-4, UML Class Diagram for the Order Processing example.

Appendix II

Template Analysis: Students of the Business

Processes and Requirements Unit, Bournemouth

University

- Assignment Brief
- Sample Student Feedback
- Results
- Observations

Assignment Brief**SCHOOL OF DESIGN, ENGINEERING & COMPUTING****ASSIGNMENT – 2009/10**

Course:	Software Systems Framework
Year:	Final Year (Level H)
Unit:	Business Processes and Requirements
Assignment Number:	1 (Part Two)
Unit Leaders:	Cornelius Ncube & Keith Phalp
Issue Date:	19/11/2009
Due Date:	18/12/2009

This is an individual assignment

This assignment forms the second part of the coursework for this unit, and covers (or partially fulfils) learning outcomes, 2, 3 and 5.

1. Appraise critically approaches to the principal requirements engineering tasks; elicitation, analysis, specification and validation.
2. Demonstrate a comprehensive understanding of relationships among client business processes, requirements and software systems.
3. Evaluate, select, and produce appropriate models of business process scenarios or problem domains, and matching requirements and specifications.
4. Evaluate critically requirements methods and research.
5. Understand the impact of professionalism upon the requirements phase.

Marks for this second part are given (out of 100).

However, parts one and two of the coursework will be weighted 70:30 to form your overall coursework mark.

Deliverables and Assessment Criteria / Marking Scheme

You have been given a brief description of a particular application domain, and have been asked to produce a number of deliverables, for the final requirements document.

Models of Processes

Produce Role Activity Diagrams for the given scenario(s). Marks will be awarded for:

- Appropriate separation of problem into constituent parts.
- Sensible choices for logic of process & appropriate level of abstraction.
- Appropriate (and correct) use of notation, e.g., state, actions, interactions & control constructs.

(50 marks)

Analysis of Processes

- Analysis of process. Describe any ambiguities that you have discovered from your analysis, relating this to the models you have produced.
- Describe, as a process modelling professional, any changes you might suggest to the process scenario, and any benefits and potential risks of such changes.

(20 marks)

Reflections on Method

Discuss the issues and solutions encountered in moving from analysis (the process models) to specification and design, and mechanisms that you would use to ensure alignment of the business process model (and business needs) and the IT system.

(Maximum 1000 words)

(30 marks)

Signature of Assignment Setter

Signature of QA

Poole House Holdings

Poole House Holdings is a typical small to medium sized enterprise (SME) that is looking to develop a software system to support a small scale mail order business.

Because of logistics of the enterprise system that is already in place, the new support system is to be interoperable with other systems and have a portable / reusable architecture. The following describes the activities of the mail order process.

The marketing department is responsible for the creation and despatch of all marketing materials.

Once a customer is ready to make an order, they fill out the order form and mail it into the company. Once received, a sales advisor processes the order. The customer details must be kept on file for future reference and credit checks are made to ensure the customer account is in credit before any order can be authorised.

The accounts advisors are therefore required to respond to any sales order request by providing them with an update on client credit worthiness. Sales advisors who are suitably trained may, if the accounts advisors are very busy, choose to do this themselves.

If a client has poor credit, the sales advisor notifies them of the situation, otherwise an order is created and sent to the despatch team; who pack and despatch the order, along with the required despatch note.

A copy of the despatch note is forwarded to the accounts advisors for billing purposes, which is already supported by the accounting system. There are usually two sales advisors and two accounts advisors available to the mail order enterprise at any given time.

The warehouse team number varies with seasonal demand and the company believe that they are unlikely to have any impact in the design of the new support system.

*Sample Student Feedback (1)***ISSUES IN MOVING FROM ANALYSIS TO SPECIFICATION AND DESIGN**

From analysing the documentation provided about how Poole House Holdings operates I have been able to create a model which shows a fair representation of the problem domain.

A major issue which would have to be resolved before moving further in the development life cycle would be to arrange discussion with the customer to ensure that ambiguities can be cleared up and that any incorrect assumption made in the original RAD are altered and a final version is agreed with by the customer.

Journal article (1) describes a useful way of doing this though the reasoning in the paper is slightly different the method can be used to, "Step 1. Describe role activity diagram" talks about creating new process descriptions even though the context has already be described.

By describing the system process based on the RAD and comparing it to the description the RAD was based on will allow links to be created between ambiguities and the part of the system which has been assumed. From this analysis customer input can be used to ensure that the RAD is in alignment with what the customer wants to be created.

The model I have produced in the form of a role activity diagram (RAD) shows how different components within the whole system interact with each other. These components include people, departments and the system itself.

From the model certain parts of the specification and design can be created such as ensuring secure login is used when people interact with the system, however exactly how this is done is not modelled by a RAD since there are several different security measures that can be put in place.

Without further information and clarification a system with biometric security may be designed when a simple password login would suffice leading to software which far exceeds the business needs being created.

The reason why situation such as the one I have just described can occur from creating a specification just from a RAD is that only interactions are modelled and not what is actually taking place such what is inputted or outputted.

To resolve these issues clear definitions of what data is inputted and outputted and how the interaction takes place needs to be described. Use case descriptions could provide some of this essential information as interaction and activities which up the RAD, can be broken down to steps and from these steps allowing a more accurate specification possibly being produced.

figure 2-1, sample student feedback received with respect to the 'Reflections on Method' part of the 2009/10 Business Processes and Requirements assignment.

*Sample Student Feedback (2)***Reflections on method**

To understand the issues and the solutions related to moving through 3 phases of modelling it is important to understand the purpose of each stage.

The initial stage is the analysis where the problem domain and the problems are defined in order to realise the requirements. This stage can often be misunderstood and can jeopardise the following stages due to misunderstanding the situation. The stages are iterative and rely on the output of the previous for their production; having a greater understanding of the original problem should respectively facilitate a better solution system. The next phase is the specification which is where the interaction between the problem domain and the solution system is defined. The third phase is the design which defines the internal operations of the potential system to be built which will achieve a required business goal.

There are a variety of problems in moving through these stages because of their separation and differences in purpose. The notations of an analysis diagram (e.g RAD), specification diagram/document (e.g use case) and design (e.g. class diagram) are orthogonal and each shows a different view. A RAD diagram (analysis) contains a number of specific roles which encompass activities, where the activities can cross between roles; a use case (specification) almost reverses this where the actors (similar to roles) are assigned to the activities; the design then assigns activities to the objects.

An issue in moving through the phases is that they are different in their levels of abstraction and therefore create the problem of establishing a level of detail. A RAD will often have a granular interaction level which makes it difficult to translate into a use case specification which has a higher abstraction. When translating a RAD into a use case the level of detail essentially decreases, however the use case will need to portray the same essential information at a higher level without losing vital specifics.

A possible solution of transitioning from the analysis stage to the specification stage is through gathering the analysis interactions into specification events and bundling the related actions in the analysis phase. The process of grouping lower level actions and interactions that are related from the analysis stage not only helps to create the specification but also helps to ensure traceability between the two phases.

The method of grouping activities can best be performed through the use of certain mechanisms. POSD is an intermediary notation that can be used to connect between a RAD and a use case. A perspective of the process is provided through a simplistic view of the roles, which are deduced from highly coupled behaviours. The behaviours on a RAD can often be spread over multiple roles whereas a POSD will group these behaviours to a higher level and generalise the behaviours to make a POSD easier to understand. The behaviour can sometimes be too complex to group into a higher level and could need to remain separate.

The POSD itself can be seen at different levels of detail. The connections can be shown at a low level such as 'locate details' and 'transfer detail to database' and then be grouped into a higher level such as 'import details'. These higher level groupings can then be given sequence numbers to translate

the sequence of interactions of the driver part interaction and the part interaction from a RAD diagram. This quality of POSD is another effective way of facilitating the transition from analysis to specification. The behaviours of a POSD can be translated to an actor in a use case and the connection between the behaviours in a POSD can be translated into the use cases, successfully translating the analysis document into a specification.

The process of moving from a RAD to a design drawing can be achieved through the separation of the RAD into the three different types. These are the environment RAD (analysis), the shared RAD (specification) and the machine RAD (design). The environment RAD defines the activities, interactions, states and roles; at this point the RAD looks similar to a regular RAD. The shared RAD then splits the roles into what is performed by the environment (e.g a person within a role) and the machine. The shared RAD lacks the use of a 'System' role but this is accommodated by the use of the machine within the role. The machine RAD then removes the activities performed by the environment to leave a clear view of what activities are performed by the machine. This approach then has the benefit of being able to create another design diagram such as a class diagram through transformation. The 3 RAD approach helps to maintain alignment of the business process model and the IT system because the machine RAD is derived from the shared RAD which is derived from the environment RAD.

From the research it is clear that the process of moving from the analysis stage through to design can be difficult while being exposed to risk at each transformation step. A key issue in reducing risk is to identify the system boundary which is required to determine the scope of those who interact with the process. The risk is not helped by the case that the transformation is mainly a manual process for example moving from the environment RAD, to shared RAD and then the machine RAD. I believe that the 3 RAD approach described is a logical and relatively simple approach to moving from the analysis stage all the way to design; however I think that automating the process would not only reduce risk in the process, but also the time taken to develop each RAD with clear alignment to the business needs.

figure 2-2, sample student feedback received with respect to the 'Reflections on Method' part of the 2009/10 Business Processes and Requirements assignment.

Sample Student Feedback (3)**3. Reflections on Method**

Software engineering is an engineering discipline whose focus is the cost-effective development of high-quality software systems (Somerville, 2007). There are various definitions of 'quality' in the context of software; IEEE defines it as "the degree to which a system, component, or process meets customer or user needs or expectations". This is the basis behind the need for the controlled analysis, specification and design of software systems.

While in principle this seems like a common sense approach to software development, the alignment of business needs and goals from the Customer standpoint and the rich process description in Analysis is often lost in the transition through to specification and design. To enhance coherent understanding of the problem domain between parties, the results of phases are visually represented through models; e.g. Analysis by Role Activity Diagrams (RADs), Specification by UML Use Case models and Design through UML Class and State diagrams. Given the different notations, difficulty can arise in maintaining the mapping between phases.

While use case modelling as a sole form of specification has its merits, the traditional approach is flawed in that it does not capture the rigor of business process modelling like RADs. UML use case modelling (in the traditional form) also has no provisions for describing interdependencies amongst use case events (Kanyaru et al, 2005).

Phalp (2009) proposes that one method of aligning business needs to specification is by using POSD (Process Oriented System Design) notation. POSD notation is designed to model the business systems of large end-user organisations (Henderson & Pratton, 1995) and by using the notation combined with a process notation such as the RAD we can begin to move to UML Use Case specification by mapping roles to large-scale activities in groups of interactions.

Using POSD notation, non-mechanistic roles are connected by actions derived from the RAD. One or more actions are then grouped into use cases defined by the sequence of interactions. We can then use various methods to move from specification to design; such as manually discovering objects from use case descriptions.

Fouad (2009) takes a different approach to moving from Analysis to Design. Fouad shows it is possible to move from Analysis to Design by incorporating RADs into the Model Driven Architecture (MDA) framework. The MDA framework considers software development from three sub-models. The first is the Computation Independent Model (CIM). This model focuses on the problem domain and requirements – the 'what' of software development – independent of any form of design. The second model is the Platform Independent Model (PIM). PIM looks at the design of the system based on CIM, but independent of any particular development platform. This can be modelled in UML Class and Activity diagrams. The final model is the Platform Specific Model (PSM) which defines the code and is therefore platform dependent.

Mapping is maintained by transforming an initial RAD through three phases. Firstly, an environment RAD is developed showing the roles independent of mechanistic distinction. A shared RAD is then produced showing for each role, the environment actions, the machine actions (where possible) and the interactions between the two. Finally, a machine RAD is developed which illustrates purely the roles of the machine rather than the environment – deriving the design. At this stage, it becomes possible to create UML use case (specification), class and activity diagrams (design). While one of the unique selling points of Model Driven Development (MDD) is the auto-generation of code, this method of transformation does not support it.

Another approach to moving from Analysis to Design is discussed in (Cox and Phalp, 2009). This approach looks at moving from RADs to Problem Frames. Problem Frames describe the generalisation of a class of a problem (Jackson, 1995) as a means of applying the same methods that worked with similar problems. The paper illustrates the move from RADs to Context Diagrams through a methodical approach which identifies roles, interactions between them, and maps the result to a context diagram. While it is identified that this method provides a viable and methodical transformation, it has issues relating to the potential loss of important design domains. It is postulated that a remedy here is to explore interactions between roles for outcomes for potential domains. Interactions should also be explored for rules for requirements or constraints - governing use or control of domains.

The POSD method is flexible with the types of project it can be tailored to, has the advantage of being reasonably simplistic and offers a good level of maturity. It would seem though, that while POSD offers an effective method of moving from Analysis to Specification, current POSD methods do not offer an explicit method of moving from specification to design - relying on practitioner's ability to derive classes from use case descriptions.

Fouad's notion of using the MDA framework as a basis for moving from Analysis to Design provides a structured, relevant approach for ensuring business needs are maintained through to development - carrying the advantages of being simple to understand and providing concise representations of complex systems. Commonly, businesses have too many project constraints to perform the level of analysis required to completely and correctly use this technique. However, the time taken to get to this stage can be the strength of this technique because it forces the analysis team to put considerable thought into the problem. Of course, this method also provides the merits of the MDA framework through abstraction of the problem allowing analysts to work on the problem domain, and developers to work on the real solution through the PSM.

While the problem frames approach has been proven with some empirical credibility to be effective, the method is still in its early stages of development, with further guidelines planned for future research.

From a review of three different methods of moving from Analysis to Design, it is your author's opinion that the method you choose should be dependent on the type of project. It is proposed that Fouad's concept seems slightly stronger, but application is limited based on project methodology.

Word Count: 997

Very Nice :)

4. References

1. Phalp, K., 2009. *Aligning Process Models (domain) and specification*. Bournemouth University Software Systems Research Centre. Available from: <http://dec.bournemouth.ac.uk/ESERG/kphalp/re/radtouc.pdf> [Accessed 12 December 2009].
2. Henderson, P and Pattern, G.D, 1995. *POSD – a notation for presenting complex systems of processes*.
3. Kanyaru, J.M. and Phalp, K. T., 2005. *Supporting the Consideration of Dependencies in Use Case Specifications*. In: 11th International Workshop on Requirements Engineering: Foundation For Software Quality (REFSQ'05), 13-14 June 2005, Porto, Portugal. (Unpublished)
4. Fouad, A., 2009. *Using RADs to move from Analysis to Design*. Bournemouth University Software Systems Modelling Research Group. Available from: <http://dec.bournemouth.ac.uk/ESERG/kphalp/alislides.pdf> [Accessed 12 December 2009].
5. K. Cox and K. Phalp. *From process model to problem frame—a position paper*. 9th International Workshop on Requirements Engineering—Foundations for Software Quality (REFSQ'03), pages 93–96.
6. Jackson, M., 1995. *Software requirements & specifications: a lexicon of practice, principles and prejudices*. ACM Press/Addison-Wesley Publishing Co. New York, NY, USA
7. *IEEE Standard Glossary of Software Engineering Terminology*, IEEE Std 610.12-1990, 1990.

The image shows a piece of handwritten feedback in cursive script. The text is written on a light-colored background and appears to be a student's response to a 'Reflections on Method' assignment. The handwriting is somewhat slanted and includes several lines of text, though the specific words are difficult to decipher due to the cursive style. It seems to contain a few lines of text, possibly starting with 'good' and ending with 'at all'.

figure 2-3, sample student feedback received with respect to the 'Reflections on Method' part of the 2009/10 Business Processes and Requirements assignment.

Results

This section of the appendix is directed at providing a summary of central themes found from examining student manuscripts. From the preliminary examination of the texts under scrutiny, an analytical template was created. Further analysis of texts revealed revisions to the template. As per recommendations in King (2006), the data was reviewed twice before the final template (see table 2-1) was agreed upon.

Theme	Sub-Theme
1. Simplicity	<i>1.1 Application</i>
	<i>1.2 Communication</i>
	<i>1.3 Notation</i>
2. Richness	<i>2.1 Notation</i>
	<i>2.2 Requirements & Specification</i>
	<i>2.3 System Boundary</i>
3. Transformation, Traceability & Interoperability	<i>3.1 Transformation & Mapping</i>
	<i>3.2 Traceability</i>
	<i>3.3 Interoperability</i>
4. Approach	<i>4.1 Software Process</i>
	<i>4.2 Alignment</i>
	<i>4.3 Tooling</i>
	<i>4.4 Maturity</i>
5. Solution	<i>5.1 POSD</i>
	<i>5.2 SystemRAD</i>
	<i>5.3 xMDA Method</i>
	<i>5.4 Other</i>
	<i>5.5 Combination</i>

table 2-1, thematic analysis template relating to the study of student manuscripts.

As table 2-1 shows, five themes were found to be central to student arguments and each theme was broken down into further sub-themes. Because the arguments were directed at the identified sub-themes and not the individual methods *per se*, it was not possible to completely organise the analysis by method. However, it was found that in discussing an overall solution (theme five), individual methods were discussed and therefore this theme focuses on those methods. It is considered important to fully “justify... each code, and... define how it should be used” (King 2006) and, therefore, this section of the appendix looks at each theme individually and discusses the extent of inclusion together with a summary of the total number of students found to be addressing each theme and sub-theme.

Simplicity

The inclusion of *simplicity* as a theme relates to the fact that many students were found to include this concept in relation to aspects in discussing the process of moving from analysis to design via specification. Therefore, it was decided to include this theme as a basis for analysis to examine the extent to which simplicity is a requirement and product of presented solution mechanisms. Table 2-2 identifies the *simplicity* theme.

	Total Number of Students (n of 47)	Percent of Students (%)
1. Simplicity		
<i>1.1 Application</i>		
1.1.1 Simple	19	40.43
1.1.2 Difficult	10	21.28
<i>1.2 Communication</i>		
1.2.1 Stakeholder involvement	25	53.19
1.2.2 IT involvement	4	8.51
<i>1.3 Notation</i>		
1.3.1 Identification of Actors / Roles	4	8.51
1.3.2 Steps to increase accuracy	10	21.28
1.3.3 Colour Coding; numbering or meaningful names	17	36.17
1.3.4 Visual	4	8.51

table 2-2, thematic analysis template for the 'Simplicity' theme.

As seen in table 2-2, the theme has been divided into three alternate sub-themes of *application*; *communication*; and *notation*. *Application* applies to comments relating to the perceived ease or difficulty in the use of methods applied to move from analysis through to specification and design. *Communication* relates primarily to how stakeholders might be involved in the process. *Notation* focuses on specific accountability of notations to facilitate *simplicity* in the process.

Richness

The next identified theme was *richness*. Many students made reference to the importance of methods being able to capture certain concepts integral in moving from analysis to specification and design. The analysis template for this theme is included in table 2-3.

	Total Number of Students (n of 47)	Percent of Students (%)
2. Richness		
<i>2.1 Notation</i>		
2.1.1 Data description	7	14.89
2.1.2 Role importance	11	23.40
2.1.3 Behavioural dependency capture	14	29.79
2.1.4 Identification of Process flaws	9	19.15
<i>2.2 Requirements & Specification</i>		
2.2.1 Analysis & Specification don't occur; or occur inadequately	19	40.43
<i>2.3 System Boundary & Abstraction</i>		
2.3.1 Description of System Boundary	27	57.45
2.3.2 High-level of abstraction	27	57.45

table 2-3, thematic analysis template for the 'Richness' theme.

Table 2-3 breaks down the *richness* theme into three alternate sub-themes of *notation*; *requirements & specification*; and *system boundary & abstraction*. Primarily, *notation* was discussed in terms of how mechanisms may or may not support enough richness in transferring information discovered during analysis through to specification and design. *Requirements & specification* was found to be important in understanding and supporting this richness and therefore included as part of the analysis. From reviewing student arguments, the concept of the *system boundary & abstraction* appeared significant enough to be included separate for discussion of the *richness* theme.

Transformation, Traceability and Interoperability

Table 2-4 identifies the thematic analysis template for what is the largest theme in terms of response associated with the process of moving to specification and design and is concerned with the notions of transformation, traceability and interoperability, and how they relate.

	Total Number of Students (n of 47)	Percent of Students (%)
3. Transformation, Traceability & Interoperability		
<i>3.1 Transformation & Mapping</i>		
3.1.1 Information is lost in the process	22	46.81
3.1.2 Accounting for information loss	27	57.45
3.1.3 Mapping is a difficulty	27	57.45
3.1.4 Addressing the difficulty of mapping	22	46.81
3.1.5 Errors being transferred in the process	5	10.64
<i>3.2 Traceability</i>		
3.2.1 Traceability as an issue	8	17.02
3.2.2 Accounting for Traceability	9	19.15
<i>3.3 Interoperability</i>		
3.3.1 Moving a RAD to a Use Case	29	61.70
3.3.2 Moving a RAD to System Models (UML)	24	51.06
3.3.3 Use with SEAM	1	2.13
3.3.4 Use Cases not required for Specification	11	23.40

table 2-4, thematic analysis template for the 'Transformation, traceability & interoperability' theme.

It is seen in table 2-4 that each area relating to the theme of *transformation, traceability & interoperability* is subdivided into three sub-themes. *Transformation & mapping* addresses issues related directly to loss or retention of information during the transformation and mapping process and the issues and solutions that are associated. *Traceability* is concerned with student observations of issues linked to how specification and design models might be traced back to original requirements and analysis. *Interoperability* is associated specifically to the ability of methods to be interoperable between other modelling conventions, and the requirement of such interoperability.

Approach

This theme addresses general notions relating to available approaches in moving towards design. It is concerned more with the high-level view of approaches and the issues surrounding them. The thematic analysis template for the *approach* theme is given in table 2-5.

	Total Number of Students (n of 47)	Percent of Students (%)
4. Approach		
<i>4.1 Software Process</i>		
4.1.1 Software Processes and interpretations	3	6.38
<i>4.2 Alignment</i>		
4.2.1 Alignment as a concern	21	44.68
4.2.2 Alignment is time consuming	11	23.40
4.2.3 OO or MDA Approach to address alignment	4	8.51
<i>4.3 Tooling</i>		
4.3.1 Tool Support	6	12.77
4.3.2 Code generation	3	6.38
4.3.3 Manual	8	17.02
4.3.4 Semi-Automatic	5	10.64
4.3.5 Automatic	4	8.51
<i>4.4 Maturity</i>		
4.4.1 Further academic work required	4	8.51
4.4.2 Enterprise Systems a concern	7	14.89

table 2-5, thematic analysis template for the 'Approach' theme.

Four sub-themes are identified in table 2-5 relating to the *software process*; *alignment*; *tooling*; and *maturity*. The *software process* sub-theme demonstrates the impact the selection of process might have on an approach, and how some approaches might be designed for a particular approach (thereby presuming approach selection). *Alignment* is described specifically as an overall ideal with relation to chosen solutions and therefore considered independently in this context with relation to the time taken to achieve such alignment. *Tooling* is mentioned to aid solution approaches; the extent being evaluated by the inclusion of this sub-theme. Lastly, whilst much is said for particular approaches and the idea of moving from analysis to design via specification, some students discussed the validity of offered approaches in terms of maturity and therefore *maturity* is included as a sub-theme.

Solution

The final thematic inclusion is associated with the student identification of solutions relating to the successful transition from analysis to specification and design. From discussed approaches, the students spoke in detail about which (if any) would be considered an appropriate remedy. Table 2-6 outlines the analysis template for this theme.

5. Solution	Total Number of Students (n of 47)	Percent of Students (%)
<i>5.1 POSD</i>	6	12.77
<i>5.2 SystemRAD</i>	2	4.26
<i>5.3 xMDA Method</i>	15	31.91
<i>5.4 Other</i>	3	6.38
<i>5.5 Combination</i>	13	27.66

table 2-6, thematic analysis template for the 'Solution' theme.

This theme focuses on five sub-themes relating to whether the student selected one of the three discussed solution mechanisms of either POSD, SystemRAD, or the xMDA method, whether they related better to an alternate method (with the inclusion of the sub-theme *other*), or whether a suggestion for a *combination* of methods might be considered a better suited solution.

Observations

In this section of the appendix, results highlighted in the previous section are elaborated on with reference to the issues related to the application and use of methods described in Section 8.3 and the student experience in moving from analysis to design via specification. In presenting the findings, it is imperative to produce a “coherent ‘story’” (King 2006), giving particular focus to the experiences of the individual, with direct quotes being “essential” (King 2006). Therefore, the final template (see table 2-1) is used to lead the interpretation of findings and where student experience is drawn upon, the student is referenced by number in brackets for identity protection. This does not mean the themes are related in such a hierarchical way as the template suggests, or by no means implies there is no relationship between attributes that appear disassociated.

Each of the identified themes are discussed in turn, drawing on individual case experience to gain clarity and understanding of the student perception of the process in moving from analysis to design, experience of and attitude toward given solutions and the overall reception of methods. As recommended by King (2006), discoveries are based upon the careful examination of the spread of codes, data entries that occur singularly, and where there is no occurrence. Whilst remaining selective in choices made regarding interpretation and remaining open to the consideration of inter-theme relationships, it is important to stress that it is not the intention of this research to derive factual evidence from these observations, rather to gain insight into the personal experiences of the individual in consideration of available techniques and specifically to learn whether or not the xMDA method would be received as a viable.

Simplicity

Simplicity is an area that appeared to be important in the consideration of moving from analysis to design and the notations used to complete the task. Over 40% commented on the simple application of methods, 53% identified the involvement of various stakeholders for consideration and 36% discussed ways to simplify methods through varying means.

Application

When moving from analysis to design there seems to be some attraction to the Use Case as a notation for specifying software systems. “Use cases are often perceived as an integral part of an object-oriented approach to software development” (Phalp and Cox 2001). In Kanyaru and Phalp (2005), this popularity is associated with being able to represent specification via actors and natural English; this in turn enables specification to be understood by stakeholders. Student 10 concurs by saying that “they rely on prose and can be seen as fairly basic representations of complex problems, they are supremely easy to understand” (10) and Student 20 identifies that because the Use Case “does not use symbols but words to describe the different processes” (20) they facilitate understanding. However, Use Cases can lead to ambiguities (1 and 44). Therefore, some

authors attempt to facilitate the move from analysis to design, focussing on intermediate techniques that are used in moving from a richer process model to a Use Case. POSD is one such method that simplifies this process by accommodating the coupling of behaviours, abstracting away from the detail of the business process (4 and 23). This makes “it is very easy to understand in terms of what parts of the system interact with each other” (24) and “easy to follow” (32). Other students argued however that the POSD technique is “not as straight forward” (36) as other techniques and not a “complete solution” (38). This is perhaps because the application of POSD demonstrated to the students uses other techniques to complete the move which complicated understanding for those students.

RADs are identified as being a “fairly easy, simple notation” (37) for modelling the business process and providing a communication conduit between stakeholders and requirements specialists; supported by Murdoch and McDermid (2000). This is because a single-page RAD is able to simplify the description of complex processes (38) with minimal training (37). A RAD offers a straightforward description of “the process to the client and captures the business needs” (29). RADs have “the advantage of describing the timeline of the system or process” (20), making the process easy to follow; an ideal represented by all three of the described methods. In Brown et al. (2006), model transformations are described to include *refactoring* (where one notation is used during transformation); *model-to-model* (where different notations are mapped); and *model-to-code* (where code is generated directly from the model). One student argued that refactoring transformations are of greater simplicity because “fewer notations have to be learnt and understood, thus conveying and communicating meaning much more effectively” (9). The xMDA method is based fully on the modified RAD where “every step of the transformation allows you to see what has changed making it relatively easy to follow” (4) and transitions to design are described as “much easier” (6). The xMDA method carries “the advantages of being simple to understand” (33) and is less complicated in comparison with POSD (31). However, others argue that the MDA is “a very complex approach to creating a system” (3) and the xMDA method has a “complex nature of splitting RADs” (34) with the RAD notation itself being complex (44 and 45).

Other mechanisms were mentioned, such as B-SCP and Problem Frames, with the central notion of simplicity in application being highlighted (7, 44, 45 and 46). Student 24 writes that the objective of this research area is to “simplify the process involved” (24) by proposing methods that help stakeholders address the issues involved in moving from analysis to design. Student 47 goes a step further by suggesting that the initial decision of choosing an appropriate method is essentially difficult because it is compounded by the consideration of the repercussions involved when choosing an inappropriate method (47).

Communication

Stakeholders are identified as “anyone that could be materially affected by the implementation or outcome of a new system” (Phalp et al. 2007), whose objectives and goals vary in as much as the number of mechanisms available to move from analysis to design (Kavakli 2004), supported by Coughlan and Macredie (2002), Kavakli (2004), Nuseibeh and Easterbrook (2000), Peixoto et al. (2008), Phalp and Jeary (2010), Phalp et al. (2007), Sommerville (2004). The communication of information between stakeholders in moving from analysis to design was highlighted as a key student concern. This is perhaps because specification provides mechanism between Business and Systems Analysts, and inadequate systems may result if there is a mismatch of understanding. “If business models aren’t understood and consequently not used to their full potential they could be seen as a waste of time and money” (16) and, conversely, by “totally focussing on the customer there is the tendency to ignore the problem context” (17), i.e. the derivation of systems design, which highlights the importance of the connection between Business and Systems Analysts.

Solutions discussed include applying notations that are simple for both domains to understand, such as RADs, as they are an “extremely beneficial method of ensuring a thorough understanding of the process is shared between all stakeholders” (6), or Use Cases, because they are “more easy to understand” (20), “which should allow the client to discuss, validate and experiment with process changes” (29). A RAD is “useful for developers who want to perform analysis; it does not include a foundation to initialise specification and design” (14). POSD “has the advantage of being easily understandable for a less technical user” (8) and can transform a RAD to a Use Case model. The xMDA method can be used to “gain a better understanding of the scenario presented and how to turn the analysis... onto the next level of requirements and specifications” (32), and “provides a structured, relevant approach for ensuring business needs are maintained through to development” (33). It is important that such notations must include the Business Analyst since often system models are commonly software-based and “exclude anyone from a non-programming background” (8). Indeed, “this could be done through ‘user-facing’ (audience) models aimed towards stakeholders” (9). With that in mind, the xMDA method is an example of a technique that aims to include both the Business and Systems Analyst by providing mechanisms that support process and design modelling through the CIM; “the advantage of using CIM is that it has better connection to business users” (19), whilst remaining attractive to IT because of this connection (3).

It is suggested that it is sometimes difficult to achieve understanding when many different notations are used because “people cannot be expected to be familiar with each notation” (16); this would result in exhaustive training (16 and 18). In the context of the SystemRAD (and the xMDA method) users must “be trained to deal with system adaptation” as well as having a clear understanding of the application of the RADs (16). This issue could be compounded when models are placed in enterprise systems where multiple teams work on individual projects. “When models are moved from team to team, they do not necessarily carry the same

experience or knowledge as others and it can become difficult to translate what one person knows and creates to what someone else can understand" (18).

Means to increase stakeholder understanding in the context of transformation (such as colour coding, numbering and the application of meaningful naming conventions) can be used in strengthen such techniques (2). Particular style guidelines, like the CP Style Rules for Use Cases (Cox et al. 2001; Phalp 2002), can be used to facilitate understanding "because there is much less ambiguity from the part of the reader" (21). Tool support, such as the enactable Use Case tool described in Kanyaru (2006), Kanyaru and Phalp (2005, 2009), can allow "stakeholders to be fully involved" (7), with descriptions containing "both pre and post states to ensure that before and after each process, the same detail and business need is understood" (18).

Others argue that it is the onus of analysis to ensure "all stakeholders understand the scope of the system and that everything in the scope is documented consistently" (8). Without this, a complete specification would not be able to be prepared and "would require further interaction with the customer... to find out what was needed" (27) to complete it. Student 41 recommends a combination of techniques should be used to maximise efficiency, ensuring that "no processes are missed out" and the "potential for risk" is accounted for (41). The important thing being that the stakeholder is involved from the start of the project.

Notation

The concern for simplicity was identified further, regarding how notations can specifically simplify and be simplified to identify and demonstrate processes involved in moving from specification to design. One way to look at this was to examine the ability of techniques to model the concepts such as *role* or *actor*. RADs are described as "an excellent way to initially project how a system works looking at specific roles and how they interact with the system as well as other roles" (36). POSD can "identify roles and actors within the system and contributes to identifying connections between these roles" (6). A POSD will take interactions and, by grouping, can simplify a RAD to highlight the essential roles and interactions (31). Surprisingly, the ease of *role* or *actor* discovery was not really mentioned in the context of the SystemRAD and the xMDA method.

Some notations demonstrate that, by having alternate phases or steps, the issue of moving from analysis to design can become simplified. The xMDA method notably moves the RAD through a three-phased approach in moving to design (1, 14 and 41) making it "very powerful" (4). This gives it an advantage simply because of the conversion in "small stages" (20), which allows for the identification of the system boundary (24). The B-SCP framework also contains three stages of strategy, context and process for an "integrated model" (1). The SystemRAD approach moves from an analysis to system RAD, then Use Case (6), much like the POSD approach moves from analysis RAD to POSD, then Use Case (14). One student suggested that such abstraction can help in consideration of the PD, i.e. problem complexity can be removed by sub-dividing the

process (43). Other students viewed this to be a software process issue, as much as notational. "One of the main issues in moving from analysis to specification and design is not separating these tasks properly. Each stage is a separate task in its own right and should be focussed on individually to ensure accuracy as a whole. Blurring the division between these can result in an inaccurate, out of scope view of what the system should do" (15). Student 34 concurs where it is said that "the main thing to be noted here is that each area should be kept completely separate and not combined. Only once the issues within the PD have been identified and new requirements drawn up for the new system should the specification section come into play" (34).

As previously mentioned, notations can be enhanced with the inclusion of colour coding, numbering and the application of meaningful naming conventions in the context of transformation as a means to increase stakeholder understanding and this was recognised by a numerous students, most likely due to this information being made directly available in the course material. Colour coding and numbering can be used to identify which parts of a source model relate to which parts of a target model after a mapping process (2, 5, 10, 16, 18, 19, 21, 23, 27 and 45). POSD can be enhanced by the use of meaningful naming conventions for the bundled interactions (3, 7, 15, 27 and 35), indeed using logical naming conventions no matter what the method can result in a preservation of alignment (8).

Models are of course visual representations that help simplify the understanding of a process. The ideas discussed on colour coding, etc. are further visual representations used to help facilitate understanding of the mapping process. This is an important point to make since much appears to indirectly rely on these visual representations. RADs are visual representation of the business process (1). POSD can form a visual abstraction the RAD (1 and 3). Use Cases are used to give a visual overview of the "functionality provided by the system... in terms of actors, their goals which are represented as Use Cases, and any dependencies between those Use Cases" (14). It is suggested that this should not be neglected since, to "enhance coherent understanding of the problem domain between parties, the results of phases are visually represented through models" (33).

Richness

The second theme that was discovered was the level of richness incorporated in notations used in moving from analysis to design. Over 40% of participants identified the significance of a rich analysis and specification process in order that a useful design is achieved. 57% of the students considered a part of this achievement relates to the discovery of the system boundary, with the same number recognising that a high-level of abstraction could help the process.

Notation

This support of richness was first and foremost found to be the onus of the notations used in moving to design models. A difficulty found was that process models simply do not provide an adequate data description and, therefore, it is next to impossible to transform these candidate models into useful designs. This is because subsequent modelling activity is concerned with modelling system activity (2). The RAD does not describe “what is actually taking place” in terms of system inputs and outputs (11 and 27), such technical detail is “omitted” (37). The POSD technique does not “offer an explicit method of moving from specification to design – relying on practitioner’s ability to derive classes from Use Cases” (33). Use Cases are suggested to be better than the RAD at describing system activity (11) and, therefore, can facilitate the move to design and the derivation of classes (33 and 46). To resolve this issue, a clear indication of “what data is inputted and outputted and how the interaction takes place needs to be described” (11).

Conversely, whilst there is this focus on a data description (or lack thereof) in moving to design, much attention was also found to be directed at the description of notations and associated elements with regard to the ability to match real-world concepts such as *roles* and *interactions*. RADs were highlighted as being useful because they were found to be capable of capturing such complex relationships (1, 6, 12, 14, 21, 36, 37 and 44) and “can be a very good way to show dependencies” (44). Focussing on roles and interactions is “important to ensure that the specification and process model stay aligned” (18). This is vital since these relationships are required to be “captured in... design” (6). Therefore, RAD techniques (such as POSD, SystemRAD and the xMDA method) are considered appropriate mechanisms as “they are focussed around user roles and activities, not classes” (8) and “can model a very detailed level of interaction when needed for the more technical people that will be building the system” (37), whilst facilitating dependencies (44). Use Cases suggested to be flawed because they do “not capture the rigor of business process modelling like RADs” (33); specifically such behavioural dependencies (21 and 33). A mechanism suggested in Kanyaru (2006), Kanyaru and Phalp (2005, 2009) can be used to enhance the Use Case definition to capture behavioural dependencies (1, 3, 5, 6, 19, 21, 38 and 46) by introducing pre and post states to Use Case descriptions (5, 6, 10, 19, 28 and 37) “providing a smooth transition from analysis to specification” (1). This may however, introduce “unneeded complexity... and make it unusable with the client” (37). Issues that cannot be accounted for by notations are traditionally “included as notes” (7).

A further notational consideration made was identified as the notation’s ability to recognise flaws in analysis and specification. RADs were suggested by some to be very useful in uncovering process flaws (6, 13, 18, 27, 28, 35, 41 and 46) and identifying process enhancements (6 and 28). A Use Case does not necessarily cater for this and, therefore, the importance of analysis is highlighted if any transformation into a Use Case is to be conducted (28). This is because faults in the Use Case “have the potential to cause a lot of damage financially if not detected until a later date” (43).

Requirements & Specification

Richness and quality of designed systems was understood to be related to the attention that is placed on analysis and specification. "The stages are iterative and rely on the output of the previous for their production; having greater understanding of the original problem should respectively facilitate a better solution system" (30). Student 40 concurs by suggesting that "it is necessary to have sound business goals" when moving from analysis to specification, and that these goals need to be mapped across in the process (40). It was suggested that research in the area of IT project failures reports that "requirements and people problems" are integral to such failures (2), supported by Al-Neimat (2005), Kappelman et al. (2006), Lavagno and Mueller (2006), May (1998), Phalp et al. (2007), Poernomo et al. (2008), STSC (2003). Through thorough analysis, problems can be avoided (8, 11, 15, 17, 20, 27, 30, 37 and 43), leading to models that account for "real-world" activities (8) and the alignment of the business process with IT implementations (17), rather than the use of software, "which far exceeds the business needs" (11). When the problem descriptions are "detailed and clear", the process of moving from analysis to specification can be "fairly straightforward" (8). However, it is common that descriptions are not so clear; requirements are "volatile" (19) and reliance is placed on solution mechanisms to assist in the resolution (43). It is plausible to move directly into Use Case modelling where systems are not complex (24 and 36). However, RADs are useful in that they can expose "complicated issues" (24 and 36). The xMDA method makes considerable effort to guide the user from analysis through to design (31) and offers "a way of producing a firm foundation of analysis through to specification and design" (14). Other methods, such as the Problem Frames approach, rely on a prior understanding of the application domain (46). However, one student highlighted that it is important to recognise that many alternate definitions for requirements and analysis can make it "very hard to see where analysis ends and specification begins" (34).

System Boundary & Abstraction

Specification is the stage where the system boundary is decided (37 and 17) and this is an important consideration (19 and 43) because only some behaviours described in analysis involve the system (4, 5, 10, 21, 24, 29 and 30); supported by Génova et al. (2005), Nuseibeh and Easterbrook (2000). A difficulty is presented when a semantically rich notation, such as the RAD, is used during analysis to provide business process models, but is not rich enough to describe the system boundary (1, 3, 10, 14, 16, 24 and 36) and transforming a RAD directly to a Use Case can result in the system boundary being omitted or inadequately described (6). Use Cases are rich with notation sufficient for describing the system boundary (1, 4, 5, 14, 16, 21, 24 and 25) and one solution to defining the system boundary is via the intermediary technique POSD (1, 5, 14 and 41), which facilitates the transformation of RADs to Use Cases (36). However, the system boundary of a large system was perceived to be very hard to demonstrate using POSD (3) and, as previously noted, the POSD was not seen by some as being a "complete" solution to identifying system boundaries (19, 36, 39 and 41) to which the xMDA method addresses (36). An important recognition made by one student was that, because POSD acts as an intermediary between an analysis process and a specification Use Case, the resultant

Use Case is not specification at all, but a Use Case representation of an analysis model, and therefore some further evaluation of the Use Case is required (44). The SystemRAD approach and xMDA method are designed to apply the system boundary notion to the RAD (1, 4, 5, 6, 10, 14, 16, 19, 21, 24, 25, 28, 34, 36, 39, 40 and 42), the main difference being that the SystemRAD suggests a transformation to Use Case to complete specification (which is similar to POSD) (10), whereas the xMDA method, whilst requiring three related RAD descriptions, does not (5, 6 and 25). The xMDA method “can identify the system boundary and illustrate relations and processes to depict Use Cases, Class Diagrams or Activity models” (36), having the added benefit of being aligned with the MDA, meeting “the needs of the IT system by providing a solid architecture of the objects within the system boundary” (3). One student highlighted that “the same issue is coming through in all the possible different mechanisms, which is how to define where the system boundary is for a RAD” (24).

Another interesting observation in consideration of the movement of models from analysis to design was the richness included in the context of model abstraction. The RAD includes complex interactions and actions that detail the entire process by role (13). The Use Case abstraction is “closer to the user than to the implementation” (2), with activities being assigned to actors. Moving from RAD to Use Case is challenging because the complexity of the RAD is not always helpful (13, 30, 35, 38 and 41). The POSD method employs “a range of viewpoints” (41) and a grouping mechanism which seeks to alleviate this difficulty by abstracting away the process detail, simplifying the RAD and preserving mappings as shared components (1, 2, 5, 8, 12, 13, 14, 15, 17, 24, 25, 30, 31, 32, 35, 36, 37, 38, 39, 41, 42, 44 and 46). A weakness of the POSD method of abstraction is that “there can be inconsistent levels of abstraction in the transformation; Use Cases are usually made up of multiple actions and interactions but there can also be some that only model a single interaction” (44). An important point was made by Student 27 where Cox et al. (2005b) were quoted as saying that “abstracting away the hard or confusing parts of the problem will only lead to an inadequate delivered system” (Cox et al. 2005b), which could be true of the POSD method if used incorrectly. One student suggests that “a consistent level of intensity would allow for an ease of understanding between different types of modelling, enabling for an accurate representation of the problem within the specification and design stage” (17).

Transformation, Traceability and Interoperability

Issues in moving from analysis to design were found to be centralised around the transformation, traceability and interoperability of solutions. In particular, almost 47% of students highlighted issues surrounding the preservation of information during model transformations, with the majority (57%) suggesting solution mechanisms can address the issue. 57% also highlighted that the difficulty in such preservation is related to modelling notations being orthogonal, with over 46% believing that available mechanisms can help avoid this

danger via intermediate techniques, facilitating the move from RADs to Use Cases (62%) and those which move directly to design models such as the UML (51%).

Transformation & Mapping

When transforming between models and domains, it is important to retain as much detail as possible so that important requirements do not become lost in translation (4, 6, 7, 9, 10, 14, 18, 19, 20, 23, 28, 29, 30, 31, 32, 33, 35, 36, 37, 38, 39, 41, 42 and 44). "An issue in moving through the phases is that they are different in their levels of abstraction and therefore create the problem of establishing a level of detail" (30). This is a complex and difficult task due to the fact that source and target models commonly originate from alternate environments. For example, a role-based analysis model (RAD), a procedural specification model (Use Case) and an object design model (Class) (2, 3, 4, 5, 8, 10, 12, 13, 15, 17, 18, 19, 23, 25, 26, 30, 35, 37, 38, 42 and 46). "Naturally this provides a problem when trying to preserve mappings. This is what is known as being an orthogonal problem. Essentially these models are used in the knowledge that they can be unrelated" (13); this is compounded by the available "array of modelling techniques" (2). Therefore, it is argued that moving directly from a RAD to Use Case can result in a loss of information (6, 10, 18, 20, 24, 30, 37, 38 and 44) because the complexity of the RAD is difficult to realise in a Use Case (7, 10, 28, 34, 38, 44 and 46). "The mappings between the states, action and communications are impossible to represent within a Use Case" (38). Process models may also include many PD related ambiguities which further complicates the mapping for developers (14, 15, 32, 43, 44 and 47).

Mechanisms are available that propose to assist in this mapping process in the preservation of information (4, 5, 6, 7, 8, 9, 10, 13, 15, 17, 18, 19, 20, 21, 25, 28, 29, 31, 32, 34, 35, 37, 39, 41, 44 and 46). POSD is a method that attempts to resolve the issue of moving from a RAD to a Use Case to reduce the risk of information loss (4, 6, 12, 13, 15, 17, 19, 20, 23, 25, 28, 29, 31, 32, 39, 44 and 46). "By expressing roles sharing behaviours with other roles (via connections) the transition becomes more simplified as the mappings are preserved between the models used" (13). However, a loss of information can still be experienced in the grouping of interactions (29, 31 and 42) where it is found that "certain actions of the system can become so generalised that they are lost in the process and aren't identified in the Use Case" (38). This is made easier by the use of meaningful naming conventions when mapping (3). As previously noted, the POSD method also neglects the system boundary (39). A SystemRAD can be used as an alternate intermediate technique in moving to a Use Case, which provides a mechanism to define the system boundary and traceability from the Use Case back to the RAD, ensuring information loss is minimised (4, 5, 6, 16, 19, 25, 28 and 29). However, "when it comes to moving from the SystemRAD to a specification format like Use Cases – there will be a loss of information that can be displayed on a RAD but not on a Use Case" (10). The Problem Frames approach proposes to assist the move from RADs to Problem Frames (12 and 34). "It has been suggested using a set of specific activities, in an iterative approach, loss of information is reduced" (13). However, this method "has

issues relating to the potential loss of important design domains” (33), and a context diagram is not always helpful with this (7). *Framing* for some, proved to be difficult (14). Use Cases can be enhanced with the addition of pre and post conditions and structured English to help reduce the loss of information by including a greater amount of detail inspired by the process model (5, 7, 10, 18 and 39). However, Use Cases are typically “ineffective as they do not have a precise section for requirements of different stakeholders” (31). The xMDA method provides a *direct mapping* mechanism from model to model through specification, rather than the OMG’s argument for objects, ensuring the preservation of information (21, 33, 39 and 25). The method “models all the information required to produce effective designs” (4), “although there are a lot of rules to follow to ensure detail is not lost” (7). This makes it easier to transform from analysis to system level models (1, 6, 25 and 38), providing a conduit to the PIM of the MDA (21 and 44).

It is further suggested by (6 and 8) that the use of methods in combination can help to reduce the loss of information when moving from analysis to specification notations. However, no matter the method or combination of methods chosen, it is considered important to guarantee that “effective mapping is undertaken to ensure consistency and accuracy between models” (9).

Traceability

As discussed in the previous section, the correct transformation and preservation of mappings between models is an important consideration. It follows that the traceability of such transformation and mapping is also highlighted as an important issue in moving from analysis to design. This ideal relates to that of ensuring what is required by stakeholders, results in implementations, and that “when managing the creation of a new system, issues often arise due to the lack of techniques used to ensure traceability between certain phases of the project lifecycle” (23). Traceability is difficult to achieve for the same reason mapping is; because of the orthogonal nature of modelling notations involved (2, 3, 8, 21 and 37). One student writes of RADs and Use Cases that it is possible to “map between the two methods but ultimately they represent different things and this is one of the major problems as modellers we have with traceability” (2).

Further to this, process models are likely to experience changes (Kavakli 2004) “and it is unrealistic to expect it not to” (10). Ideally, changes made in a source or target model will reflect in all related models automatically (Koehler et al. 2007). The complicated nature of mapping makes this task difficult, if not impossible (2, 3, 8, 21 and 37). However, it can “be aided with automated case tools and other modelling applications” (8). As previously mentioned, SystemRADs can cater for such traceability through simple visual mechanisms such as colour coding each related RAD interaction and then relating it to a Use Case (10). The BSCP approach addresses traceability with the inclusion of the three themes of strategy, context and process and “having a means of cross referencing” (8) between themes and notations. Refactored transformations (like the xMDA method) approach traceability by focussing on a single form of notation (the RAD) (9 and 20). Use

Case states can enable traceability to the RAD using information taken directly from the process model (18 and 21). POSD approaches traceability by ensuring there are connections between the Use Case and RAD by creating “smaller behaviours” that reflect the detail of the RAD onto the Use Cases (21 and 23). “A possible solution of transitioning from the analysis stage to the specification stage is through gathering the analysis interactions into specification events and bundling the related actions in the analysis phase. The process of grouping lower level actions and interactions that are related from the analysis stage not only helps to create the specification but also helps to ensure traceability between the two phases” (30).

Interoperability

The solution for successful transformation, mapping and traceability of models may lay in the ability of modelling techniques to be interoperable with other techniques and software design processes, since the idea of a single generic language that is adaptable to all is unrealistic (Jouault and Kurtev 2006; Mattsson et al. 2009; Rombach 1988). It is possible to move from a RAD directly to a Use Case, the enactable Use Case tool can also be used to move from process model to Use Case (7 and 16). However, as mentioned previously this can result in a loss of information (15), even with the application of the enactable Use Case tool. “Although direct mapping can bring benefits, it is also worth considering whether it would be more beneficial to use an intermediate notation to assist with translation from analysis to specification and design” (16).

POSD was highlighted as an intermediate technique by most, enabling the transition from a RAD to a Use Case (1, 2, 4, 5, 6, 7, 18, 19, 20, 21, 23, 24, 25, 28, 29, 30, 31, 33, 34, 36, 38, 39 and 40). Use Cases can then be used for class discovery. However, it was noted that the POSD method does not offer the explicit transition to design models, leaving class discovery to the developer (33). SystemRADs again allow the transformation to Use Case (6, 7 and 16) and have the benefit of sharing similar notational constructs with the RAD. However, a Use Case is still the target platform. The xMDA method does not require transition to Use Cases (4, 5, 6, 23, 25, 29, 31, 40 and 44) to complete the move to specification (although it can generate a Use Case from specification if required (20, 36 and 40)) because the thoroughness of refactored transformations negates the need for other intermediate diagrams (such as Use Cases) (9); this is a subject that “causes a lot of debate as Use Cases are popular in industry” (7). The “method provides a decreasing level of abstraction and an increasing level of system requirements detail as it moves towards its culmination in *Machine RADs* and [could] possibly [be] augmented with [the] analysis of *Strategic Dependency* and *Strategic Rationale* when defining requirements alongside SEAM (Strategic Enterprise Architecture Method) to ensure alignment with business goals” (40) - see Wegmann et al. (2005, 2007) for more information on the SEAM. Further to this, the xMDA method defines specification right into design, allowing for the derivation of specification based system models (3, 4, 5, 6, 7, 19, 20, 21, 23, 24, 25, 29, 30, 31, 32, 33, 36, 37, 38, 39, 41, 42 and 47) via Transformation and Platform Information. Student 39 suggested that design models could in theory be extracted anywhere along the process (39) with the application of such information. Student 8 reported a

successful application of the transformation rules in reverse, deriving a RAD from a Use Case, giving an indication of the dynamics of this method in terms of interoperability (8).

Approach

Analysis of the student feedback resulted in the identification of noteworthy areas surrounding the application of approaches that have been discussed in the previous sections of this appendix. In particular, almost 45% of students felt on reflection of what was presented to them that alignment plays an important role in moving from analysis to design. This is perhaps expected considering the importance placed upon transformation and traceability in the previous section.

Software process

Surrounding the issues of moving from analysis to design via specification is the consideration of the actual software processes that stakeholders must also contend with. Alternate software processes will be used within different organisations and projects, albeit traditional or current model driven approaches. The selection of software process was felt to influence the success or failure of the project (2 and 15). This is because some software processes will facilitate approaches and mechanisms better (or worse) than others (26). For example, in the MDA, the CIM is described as disconnected from the transformation process of the PIM and PSM (21), supported by Section 6.1, Fouad et al. (2011), OMG (2003b), and therefore any approach involving the MDA will find difficulty in making this connection, for which the xMDA method is designed to apply (33).

In view of the difficulties in moving from analysis to design, it is suggested that the selection of method, approach and/or combination should be dependent on the situation (33). One student writes that “the question of moving from analysis to design presents many arguments from many alternative domains, such as RE; BPM; and the Software Process... the answer is in putting together all the pieces of this puzzle, from each of the domains” (2).

Alignment

The crux of the issues surrounding the move from analysis to design according to the students seems to be the need for an alignment between what software systems represent and what is required by business (Fouad et al. 2009; Fouad et al. 2011). This was experienced in examination of texts relating to transformation and traceability, and now corroborated with evidence suggesting such alignment to be the resulting ideal (1, 2, 3, 5, 8, 9, 11, 12, 15, 16, 17, 18, 26, 28, 29, 30, 33, 34, 40, 44 and 45). Student 44 writes that “it’s important when moving from analysis to specification and design to ensure that there is alignment between the IT system and the business goals and needs” (44). Student 15 concurs by saying that “business needs, business processes (including business process models) need to be in correlation to the IT system and have an accurate

representation, in order to meet goals, requirements and expectations of the client and provide a successful architecture" (15).

Such representation is addressed by the discussed solution mechanisms, mainly in the form of transformative capabilities. B-SCP integrates techniques to ensure the alignment of IT with business strategy (1, 8, 33, 34 and 45). POSD attempts to maintain abstract connections with logical naming conventions in a view to address business/IT alignment (3, 8, 15, 16 and 17). Use Cases can be enhanced with pre and post conditions to help with alignment issues (16, 18 and 29). The SystemRAD addresses alignment by insuring process information is transferred to specification along with the system boundary (16). The xMDA method maintains this alignment specifically via a model driven approach (6, 18, 26 and 33) by guiding the user through different RAD incarnations based on analysis, ensuring alignment is addressed in design (1, 3, 18, 30 and 40).

Applying such solutions "will remove any inconsistencies and ensure alignment between analysis and specification leaving us with an accurate overview of the business' system requirements" (16). However, achieving alignment can, with most methods, be a time consuming task (2, 7, 18, 19, 23, 24, 30, 33, 34, 37 and 47) and may result in a combination of methods being applied for successful alignment (12). POSD was considered favourable to Student 23 since "it will be less time-consuming" (23). However, in support of the xMDA method over other methods, Student 24 writes that the method seems "more flexible in the long run and will save time on larger projects" (24), perhaps being automated to reduce the time and risk involved in attaining alignment (30). Student 33 goes a step further suggesting that the time taken during an xMDA transformation is "the strength of this technique because it forces the analysis team to put considerable thought into the problem" (33). Of course, all of this remains to be proven in commercial application.

Tooling

As previously mentioned, tool support could facilitate the integration of these mechanisms into real implementations. Although far less of a concern, a number of students discussed the available approaches and suggested that alignment could be helped with the introduction of tooling.

The xMDA method described as currently in a conceptual stage with no tool support available (3, 4). This means that the method is manual (30, 31, 32, 37 and 44). Suggestion given is that an automatic approach would not be possible due to the nature of transformations and importance of human interactive decision making in the move from analysis to specification (3, 4, 23, 38 and 44). Therefore, a semi-automatic approach is considered as an appropriate direction for further research in this area (3, 4, 23, 32 and 44). Others suggested an automatic approach was viable, and would reduce both the time taken and involved risk (30). An advantage of this is that any model output from the xMDA method (via XMI) could be used with MDA applications for future model-to-code generation (3, 9, 26 and 33). The enactable Use Case tool is an

automatic approach (3) and is available to allow users to investigate process and specification solutions via the modelling of dependencies (3, 7 and 47). Although Student 47 mentions further work is required since, if used incorrectly, the resulting Use Case could simply represent the process and the output from automation may be “pointless” (47) because it does not save time; Student 18 mentions that no “real” (18) tools are available, perhaps eluding to the fact that this mechanism is not included in any COTS Use Case software. POSD has a manual approach to transformation (3); no transformative tool support is available.

It was notable that most of the attention for tooling was placed on the xMDA method; few ventured to consider that most mechanisms lack architectural and tool support which could be a general concern for research in this area. The focus on the xMDA method could also suggest it is at least viable as an approach and a candidate for tooling.

Maturity

The last area of consideration when discussing approaches to move from analysis to specification follows directly from the end of the last section of this appendix, where it was highlighted that support for architecture types and tooling was missing, or at least immature, with respect the presented solution mechanisms. It was argued that further research into guidelines was required if the Problem Frames approach was to be considered further (33). Proof of the alignment capabilities of the xMDA method needs to be made before an acceptance is made that this method can be helpful in deriving design models (39). The issue being that “it isn’t a mature model or a complete method” (47). Much concern was raised regarding the application of this approach to enterprise systems (3, 23, 31 and 37), with another feeling the method could “save time on larger projects” (24). The POSD method is also compromised by the lack of academic work (47), let alone real application, and came upon the same criticism regarding application to large systems (3). It appears difficult to know exactly when to stop including behaviours when creating a POSD. That is, no direction is given in ascertaining the most suitable abstraction level. If everything is included, especially in large systems, most behaviours will contain promises with each other, and numerous sub-behaviours; this compounds the issue and will certainly be difficult, if not impossible to depict. The authors call this a “topological” issue which remains unresolved (Henderson and Pratten 1995). Others argued that POSD is suitable for large implementations (24 and 37). Indeed, this notion of concern for application to large enterprise systems has been generalised and applied to all identified mechanisms, with one student saying that they will “only work with small less complex systems” (25).

It is these larger organisations that require “solutions for business integration... unbiased in regards to the technology and manufacturer” (26) and it is important to realise that “all of these mechanisms have been created to satisfy the academic community and not the business as a whole” (42). This illustrates how, whilst

methods may be academically sound, practical application and demonstration is required to substantiate claims.

Solution

Overall it was expected that most students would suggest that a combination of alternate methods, depending on the project and software process, would provide the best solution in moving from analysis to specification and design. A striking observation was that over 31% of students reflected positively on the xMDA method as a solution; which was more than the expectation that a combination of methods might be the desirable (28%).

POSD

Several students (12%) identified POSD as a preferred method of moving from analysis to specification (15, 17, 23, 34, 37 and 44), citing the methods attention to preserving alignment between business process and IT models (15, 17 and 23). One student wrote that the POSD method seemed “the clearest and most obvious way to transform [the] RAD into Use Cases” (44).

SystemRAD

The reception of the SystemRAD was a little more disappointing in comparison with the other techniques with only two students highlighting the benefit of the SystemRAD, and even then, suggesting that it might be beneficial to be augmented with other techniques in order that a complete specification is derived (19 and 28). This was perhaps due to the overall lack of publication and awareness regarding the technique.

xMDA

The xMDA method received the strongest support with over 31% of students identifying the potential of this method (1, 5, 7, 14, 20, 24, 25, 30, 31, 32, 33, 36, 38, 39 and 47). One student suggested that the xMDA method “is a logical and relatively simple approach to moving from the analysis stage all the way to design” (30).

Others

Other students looked towards alternate techniques to solve the difficulties in moving from analysis to design. Student 45 suggested that Goal Modelling “would be suitable” (45). Another felt that the architectural choice, such as the MDA, could provide sufficient facilitation in moving from analysis to design (26). One went further in questioning whether or not a transformative move from analysis to specification was necessary in the first place. “Arguably, a analysis model such as a RAD could be used to create a set of requirements and details in the form of something like a checklist, which could then be referred to when building Use Cases

from the specification to ensure nothing is missed out and forgotten” (44). However, it is noteworthy that no strong feedback was received to support the use of other techniques rather than those presented.

Combination

It was found that, despite some students considering one technique as more appropriate than others, the majority of those not identifying the xMDA method as an overall solution to the presented issues felt that a combination of techniques, dependent on the situation, would be fitting (1, 6, 10, 12, 13, 14, 18, 19, 28, 29, 32, 41 and 46). This is because using any technique alone “could result in neglecting critical aspects of IT requirements analysis” (1) with one student suggesting that “the key is to use a combination of these methods” (10).

Appendix III

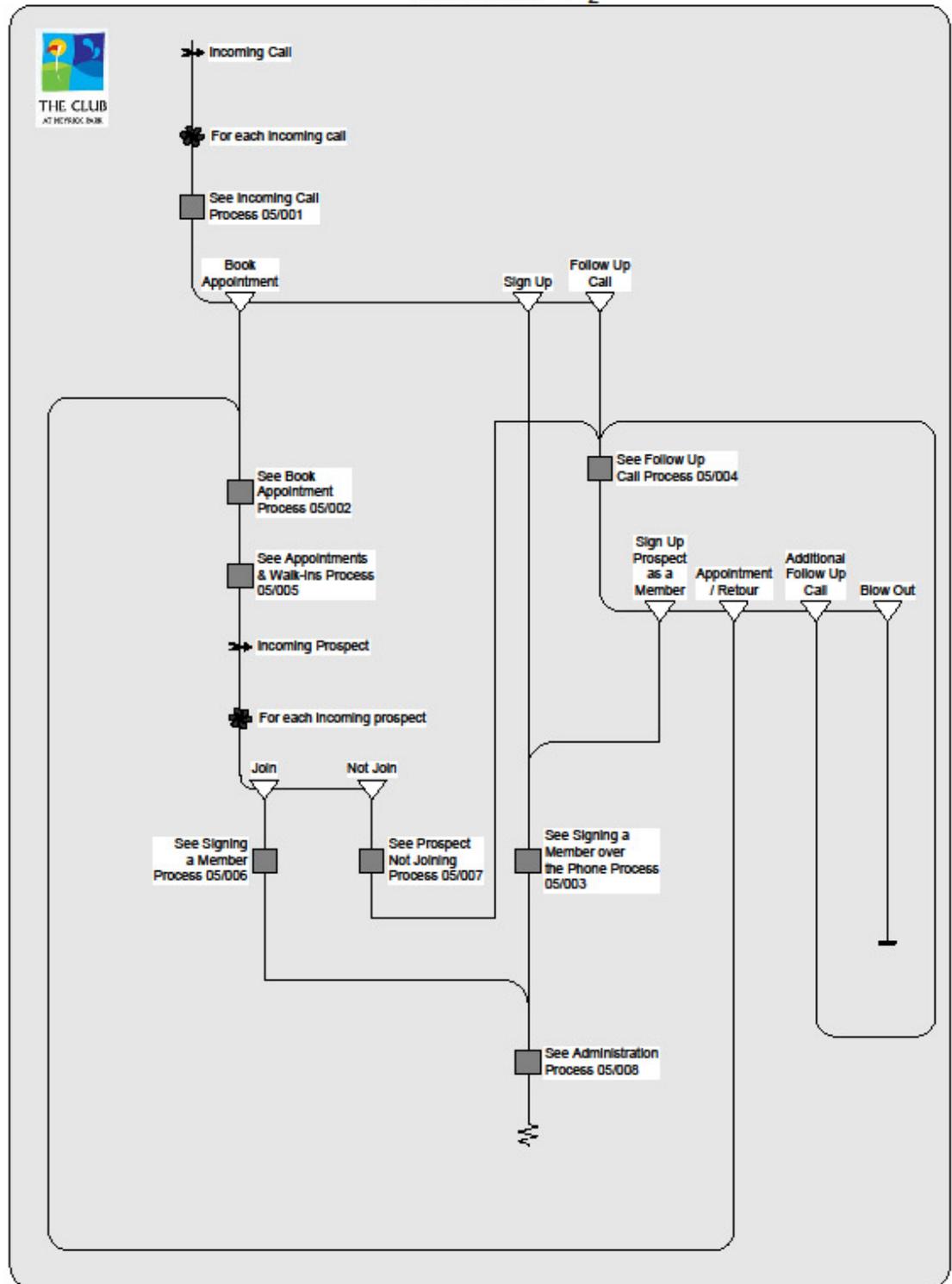
Case Study: The Club at Meyrick Park

- Process Models
- Summary of Discussions

The Club at Meyrick Park

Sales Advisor Process Overview

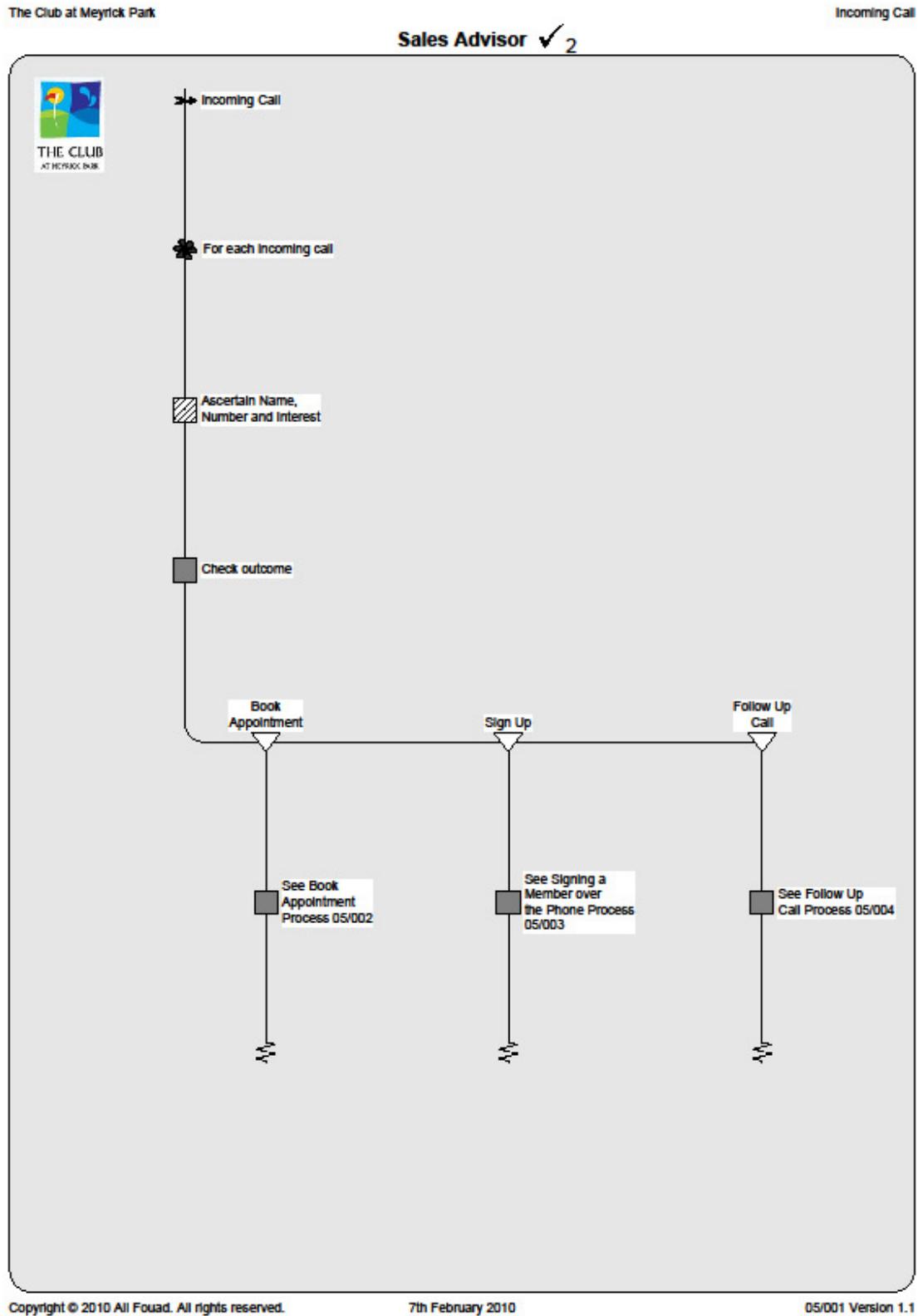
Sales Advisor ✓ 2



Copyright © 2010 Ali Fouad. All rights reserved.

7th February 2010

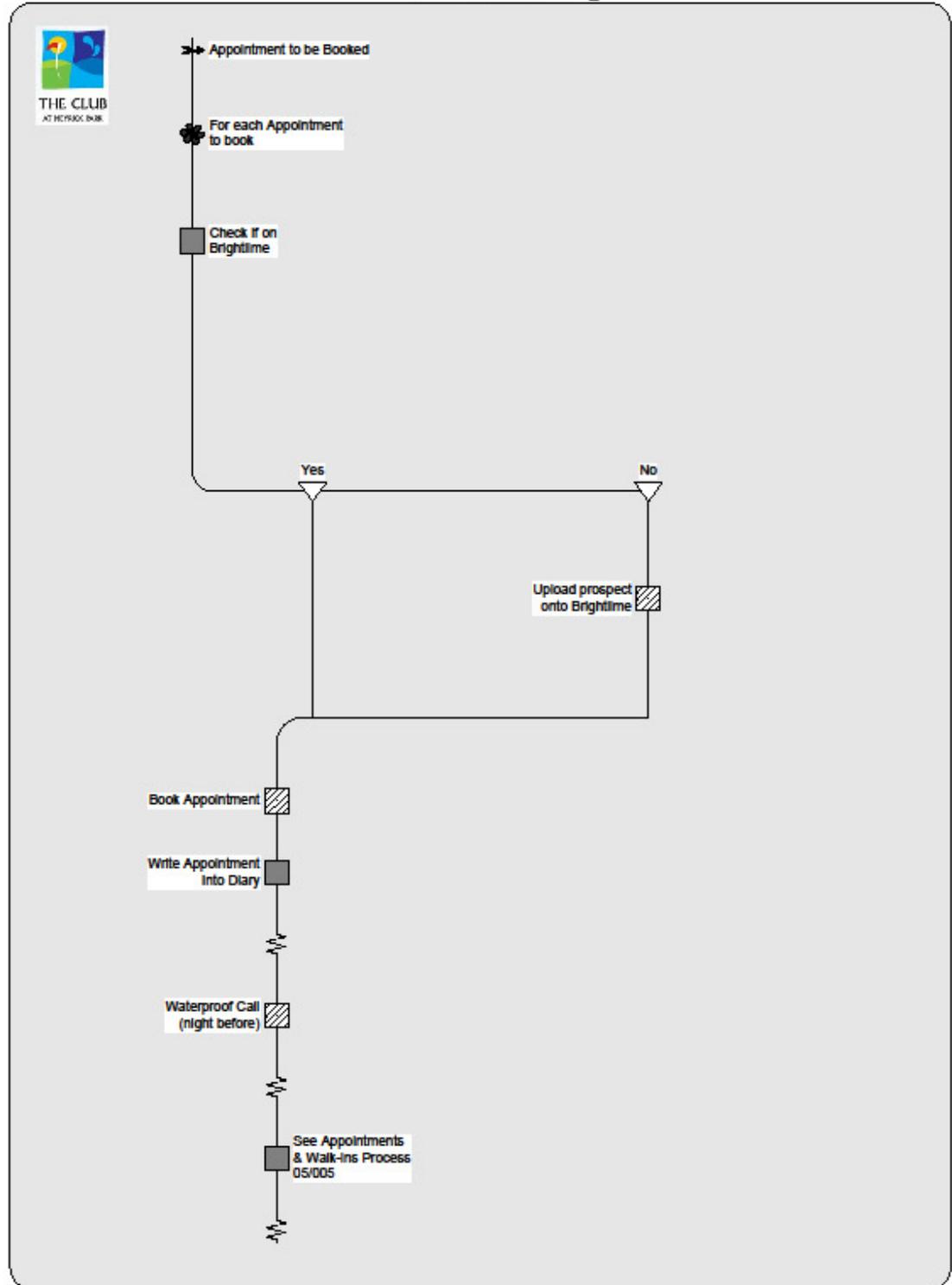
05/000 Version 2.0



The Club at Meyrick Park

Book Appointment

Sales Advisor ✓ 2



Copyright © 2010 Ali Fouad. All rights reserved.

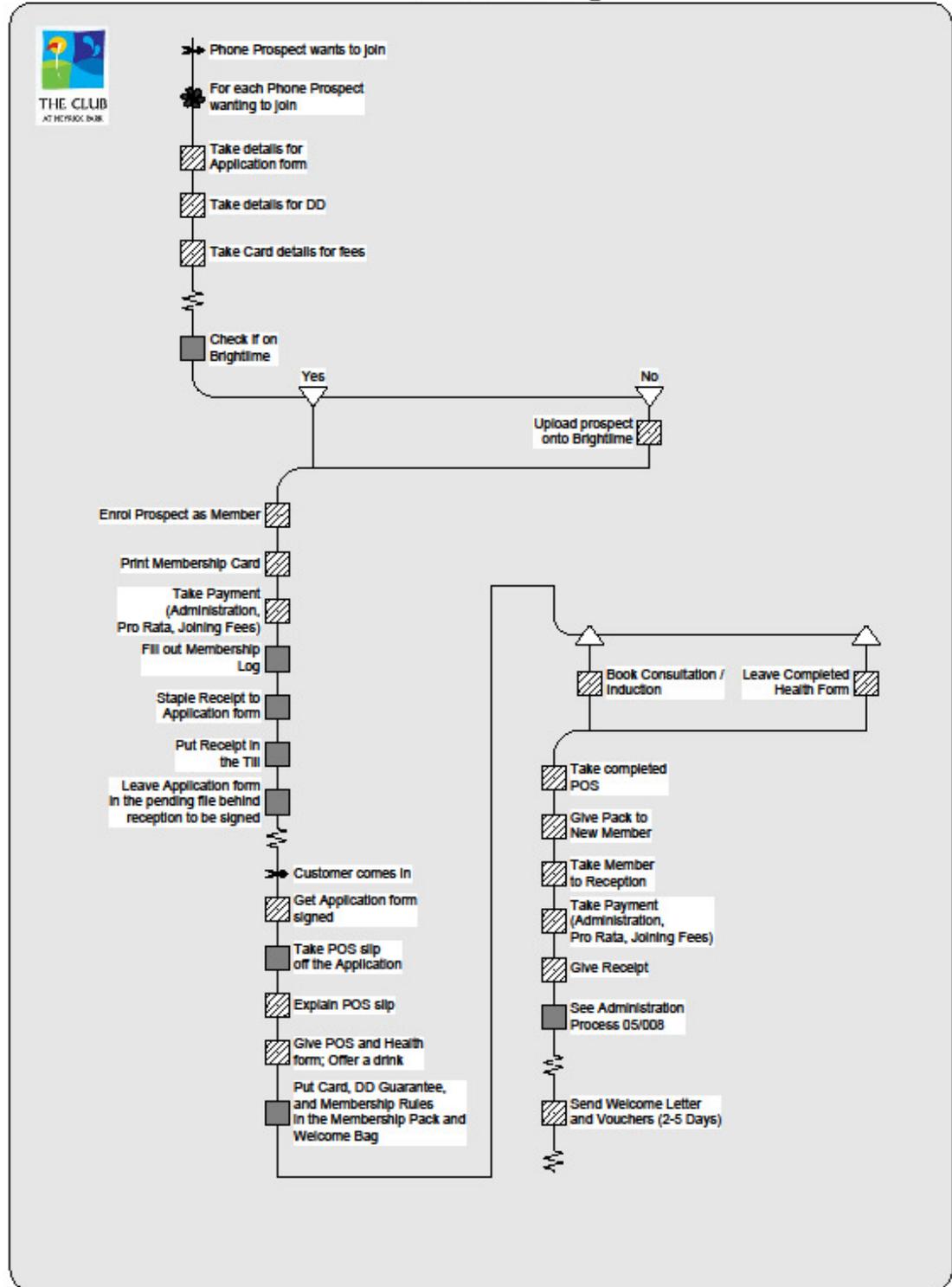
7th February 2010

05/002 Version 1.0

The Club at Meyrick Park

Signing a Member over the Phone

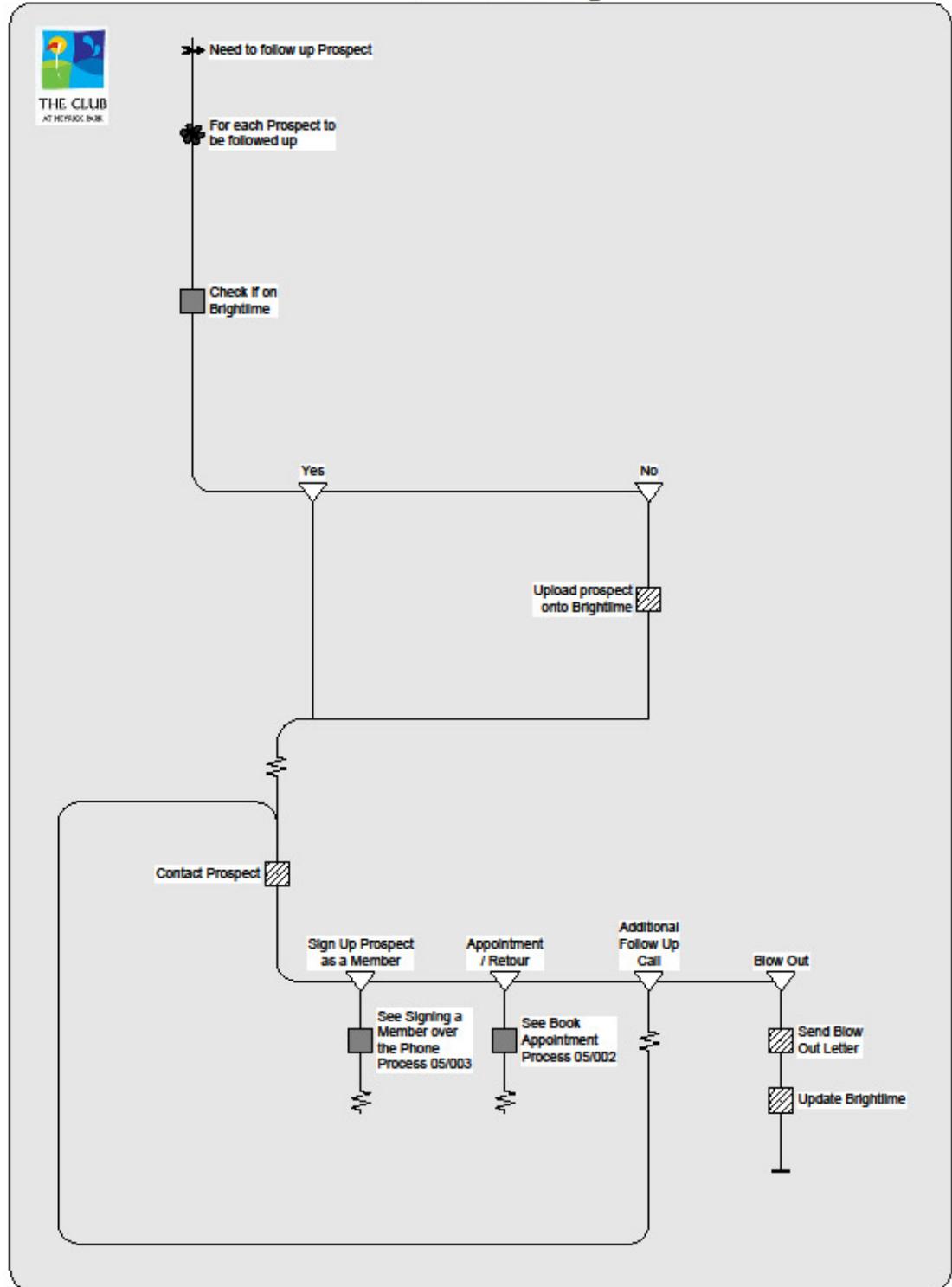
Sales Advisor ✓ 2



The Club at Meyrick Park

Follow Up Call

Sales Advisor ✓ 2



Copyright © 2010 Ali Fouad. All rights reserved.

7th February 2010

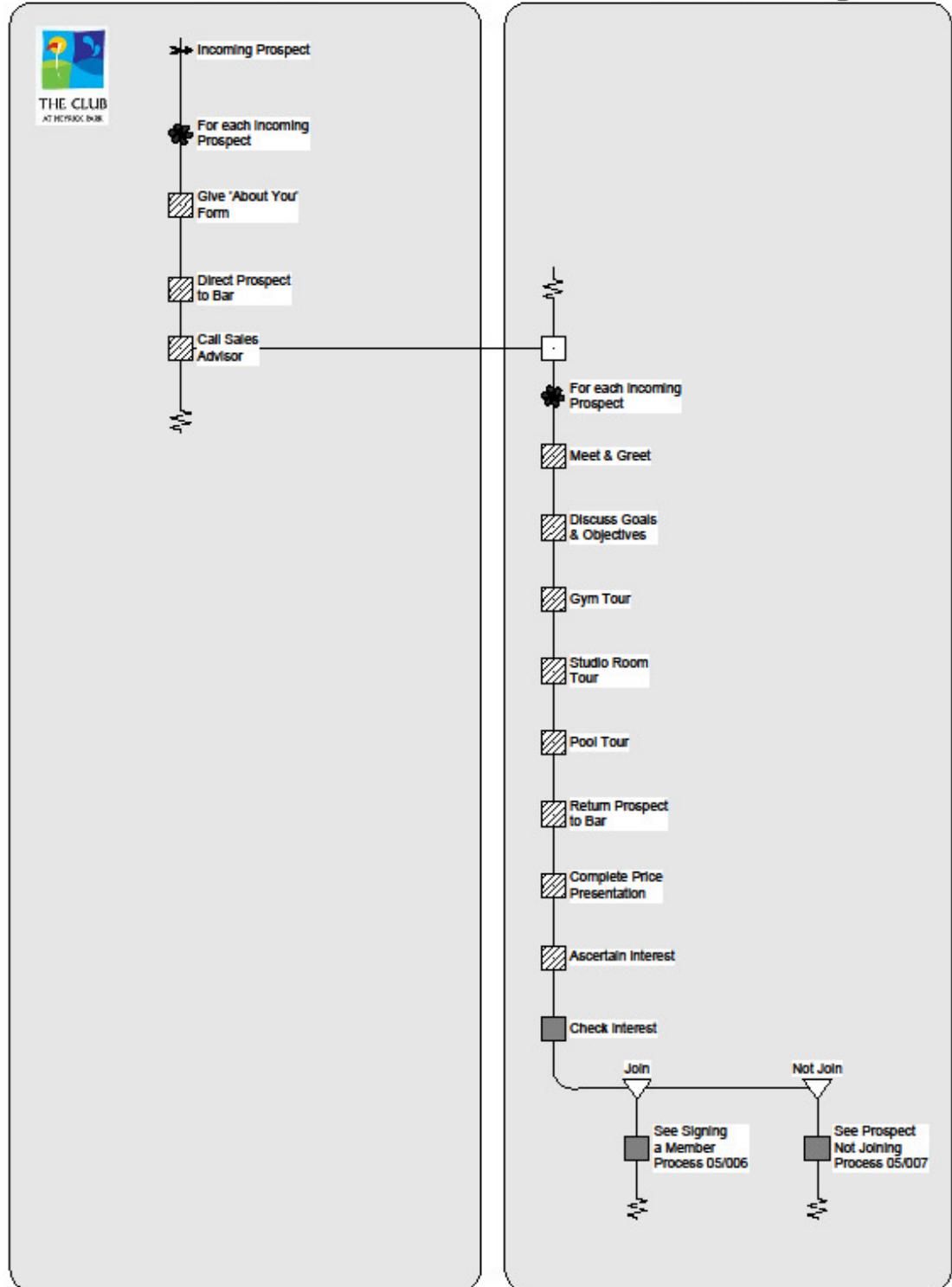
05/004 Version 1.0

The Club at Meyrick Park

Appointments and Walk-Ins

Reception ✓

Sales Advisor ✓ 2



Copyright © 2010 Ali Fouad. All rights reserved.

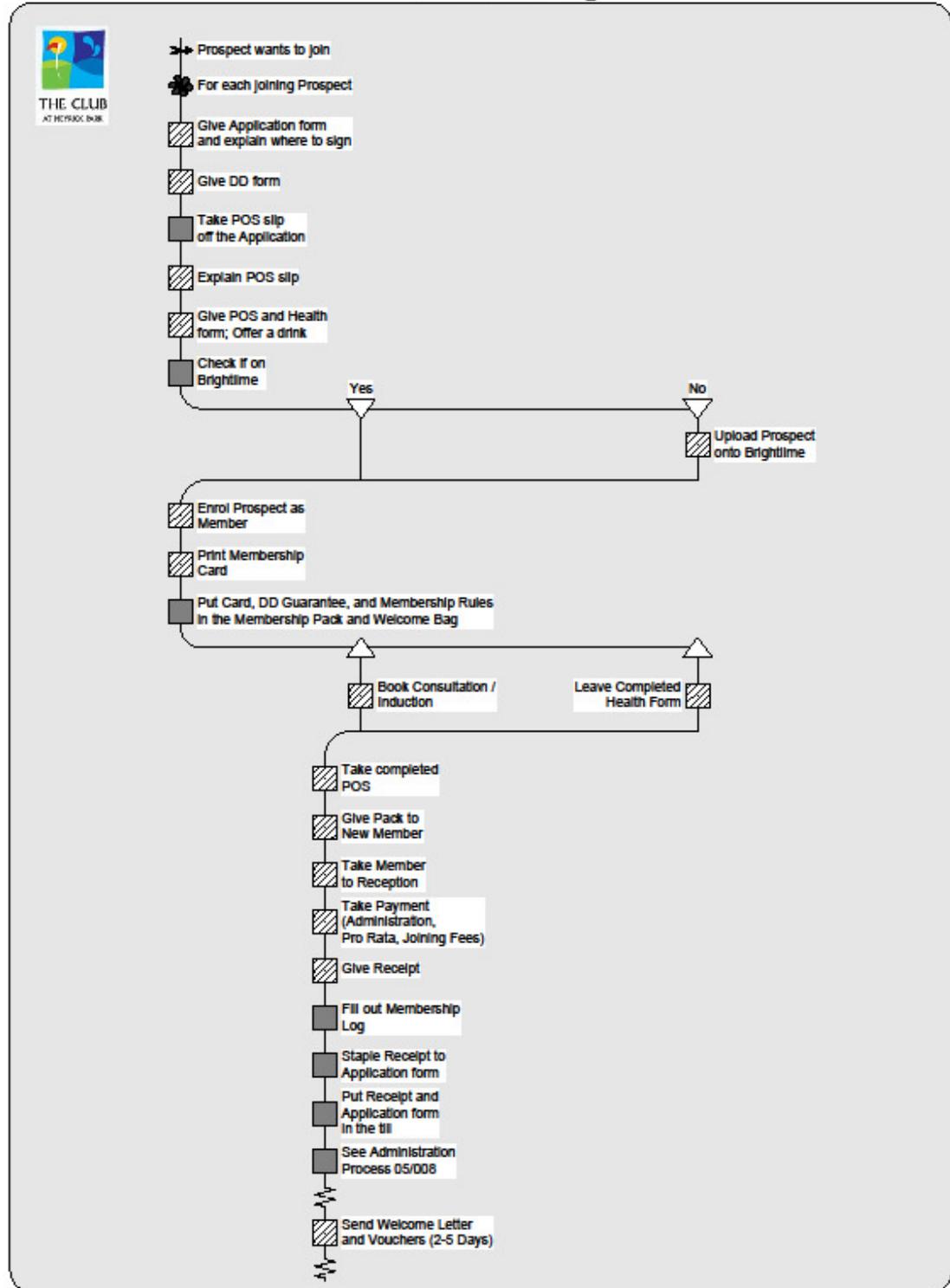
7th February 2010

05/005 Version 1.1

The Club at Meyrick Park

Signing a Member

Sales Advisor ✓ 2



Copyright © 2010 Ali Fouad. All rights reserved.

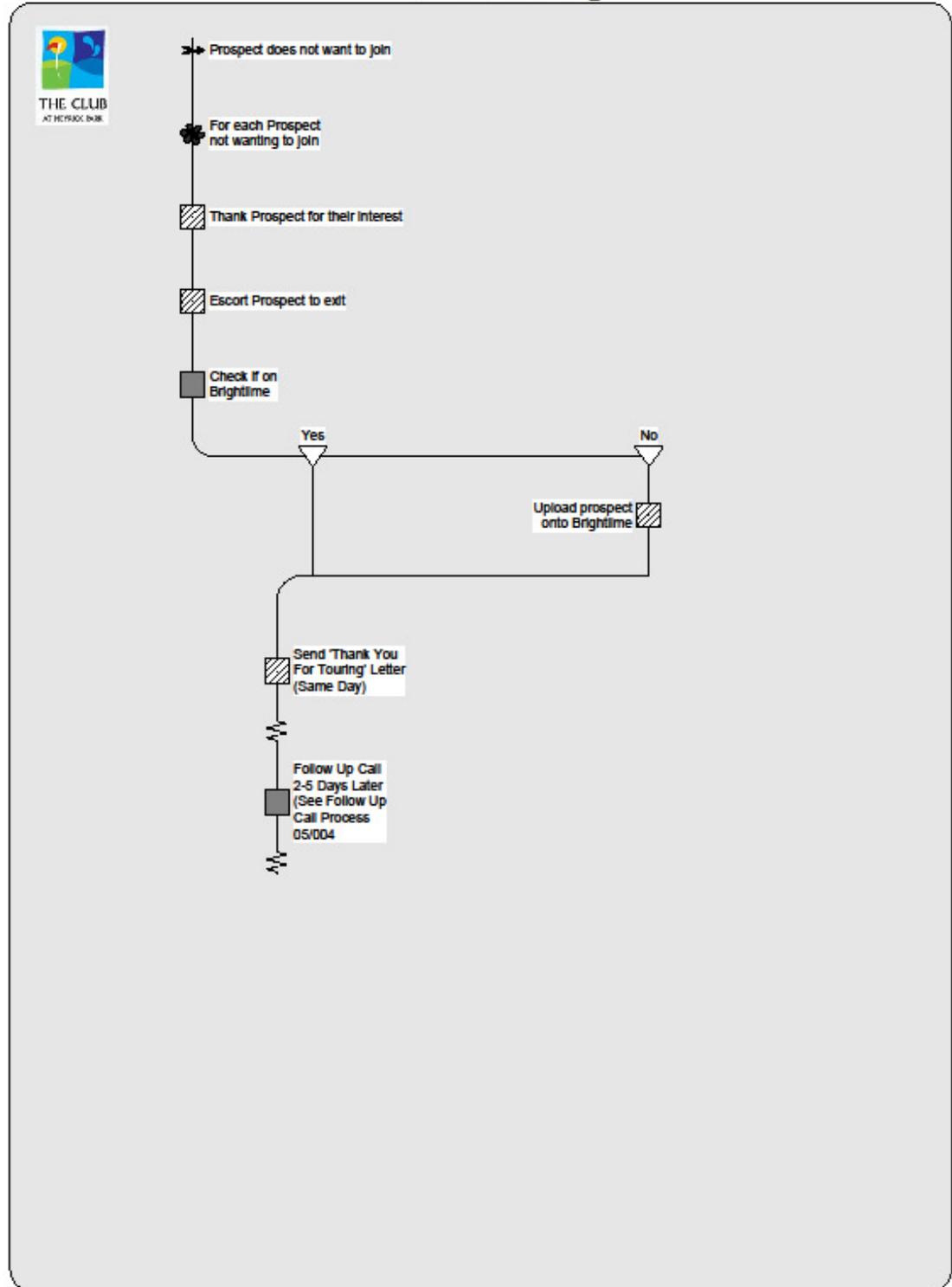
7th February 2010

05/006 Version 1.1

The Club at Meyrick Park

Prospect not Joining

Sales Advisor ✓ 2



Copyright © 2010 Ali Fouad. All rights reserved.

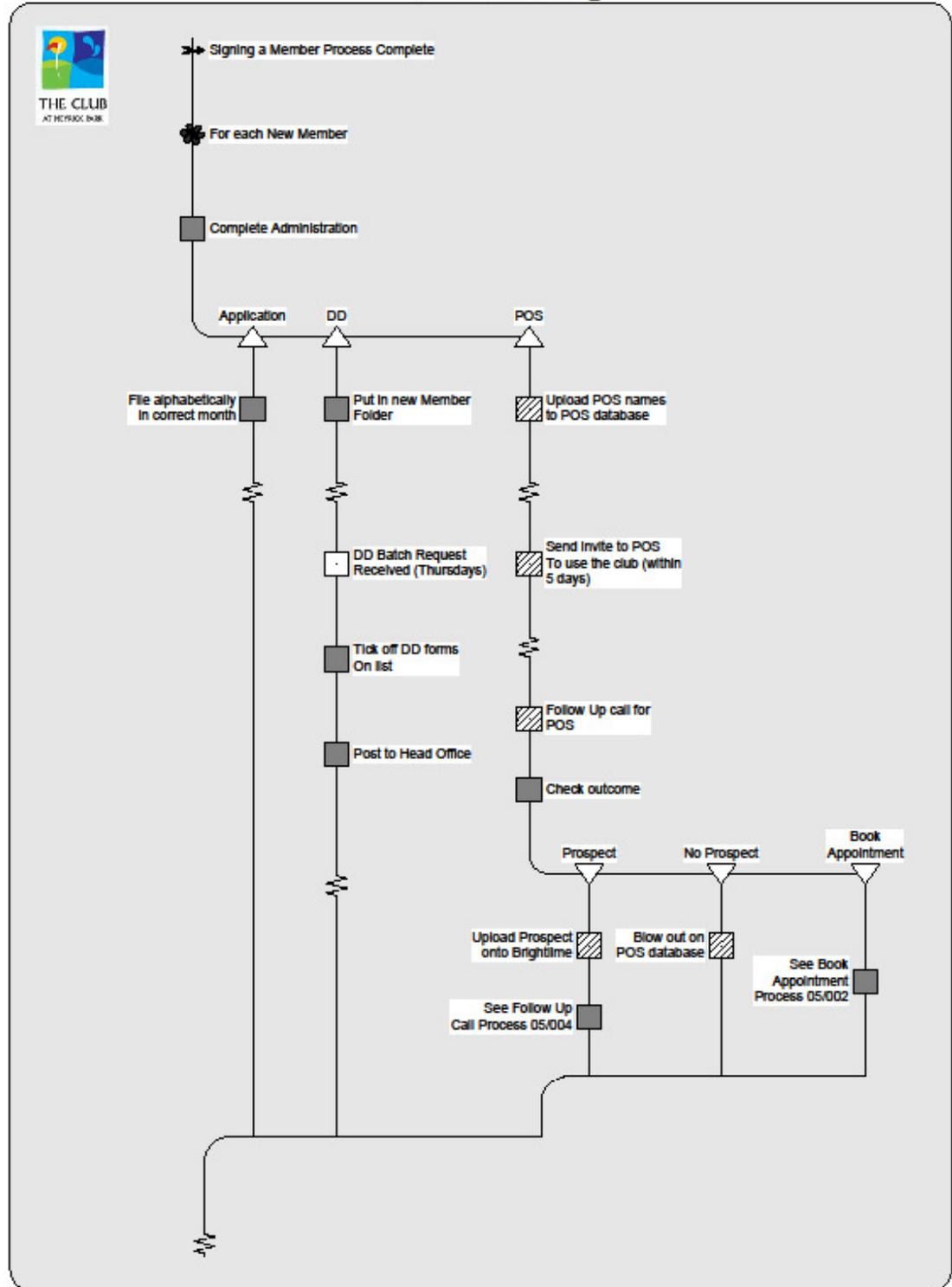
7th February 2010

05/007 Version 1.0

The Club at Meyrick Park

Administration

Sales Advisor ✓ 2



Copyright © 2010 Ali Fouad. All rights reserved.

7th February 2010

05/008 Version 1.0

Summary of Discussions

After a series of elicitation meetings with the Membership Manager of The Club at Meyrick Park over a two month period early in 2010, eight process models resulted. These models are included in the first part of this appendix, capturing the complete *Membership Sales* process. The purpose of this section is to provide a summary of the elicitation process and some feedback from The Club at Meyrick Park in terms of this process and the application of produced reference models.

Initially, process discovery and capture was investigated with the Membership Manager using a middle-out analytical technique. That is, the Membership Manager identified several procedures which were investigated in turn by identifying inter-process dependencies and expanding the level of detail defined for each. Internal activities and interactions were built upon until external entities and collaborations were reached. This resulted in the achievement and verification of a complete process architecture for the *Membership Sales* process by the Membership Manager.

Simple natural English text documents were first used to describe behaviours until enough information became available to produce foundation RADs of those behaviours. In an iterative nature, the Membership Manager produced text and received RADs in return to be verified. Upon receiving the initial RADs and a brief explanation of the notation, the Membership Manager was able to review and adjust the RADs to their satisfaction, thereby demonstrating the benefit of the notation being both easily understandable and requiring little or no training to use. The Membership Manager described the process of elicitation as being relatively pain-free; meetings were easily accommodated and documentation was able to be shared and discussed via email and in person.

Initial textual descriptions outlined by the Membership Manager produced rather sequential RADs. However, upon reviewing and applying the RAD notation, the Membership Manager was able to account for event-based and non-deterministic activity using the RAD trigger and the undefined notational elements - for example, see the *Appointments and Walk-Ins Process Model (05/005)*. The Membership Manager considered this to be useful since the processes, when described sequentially, became overly complicated. This was because certain activity did not necessarily flow in a sequential manner; the undefined notation allowed activity to be described without such constraint.

Upon completion, the process models (as included in this appendix) were formally presented to the Membership Manager of the Club at Meyrick Park in February 2010. From initial conversations with the Membership Manager, the models were successfully used to train new Sales Advisor employees in the Membership Sales department. The Membership Manager at the time said "we had a new starter [just after the

process models were presented]... As a manager, they [the process models] were very helpful in demonstrating the process... and outlining [the new starter's] responsibilities”.

Further discussions over the period after the initial application revealed that both the new Sales Advisors and the General Manager had reflected positively on the documented processes. The Membership Manager advised that the new Sales Advisors had said the process models made it easy for them to visualise their roles and were very useful for them to refer to as a checklist to ensure they had completed all tasks relating to the process, and as guidance to which tasks were required to be completed when uncertain about the process. The General Manager, who had initially requested that the Membership Manager undertake the task of documenting complete *Membership Sales* process, was said to be very impressed by the quality of process models produced and the use of the diagrams for internal documentation and training purposes. The Membership Manager advised that the process models would remain documented internally for that purpose.

In the morning of the 4th August 2011, a final meeting took place with the Membership Manager who took part in the original elicitation process to discuss the success and future of the process model application within the organisation. The Membership Manager spoke positively in terms of the application within The Club at Meyrick Park and advised that they had since moved on to manage a larger Membership Sales department in an another organisation related to Health and Fitness facilities. Opportunities of documenting new processes for that organisation using techniques described by this research were raised by the Membership Manager and discussed. However, this time there was interest in moving a step further; since the organisation lacks any documented processes and, with process flaws being noted, the suggestion was to document the current processes and use the process models to re-engineer streamlined processes for efficiency gains, perhaps using ideas of the LSS method. This is useful in demonstrating the successful application of this research and lays a foundation for future collaborations related to this research.

Appendix IV

xMDA and The Club at Meyrick Park

- Environment, Shared and Machine RADs
- Class Diagram

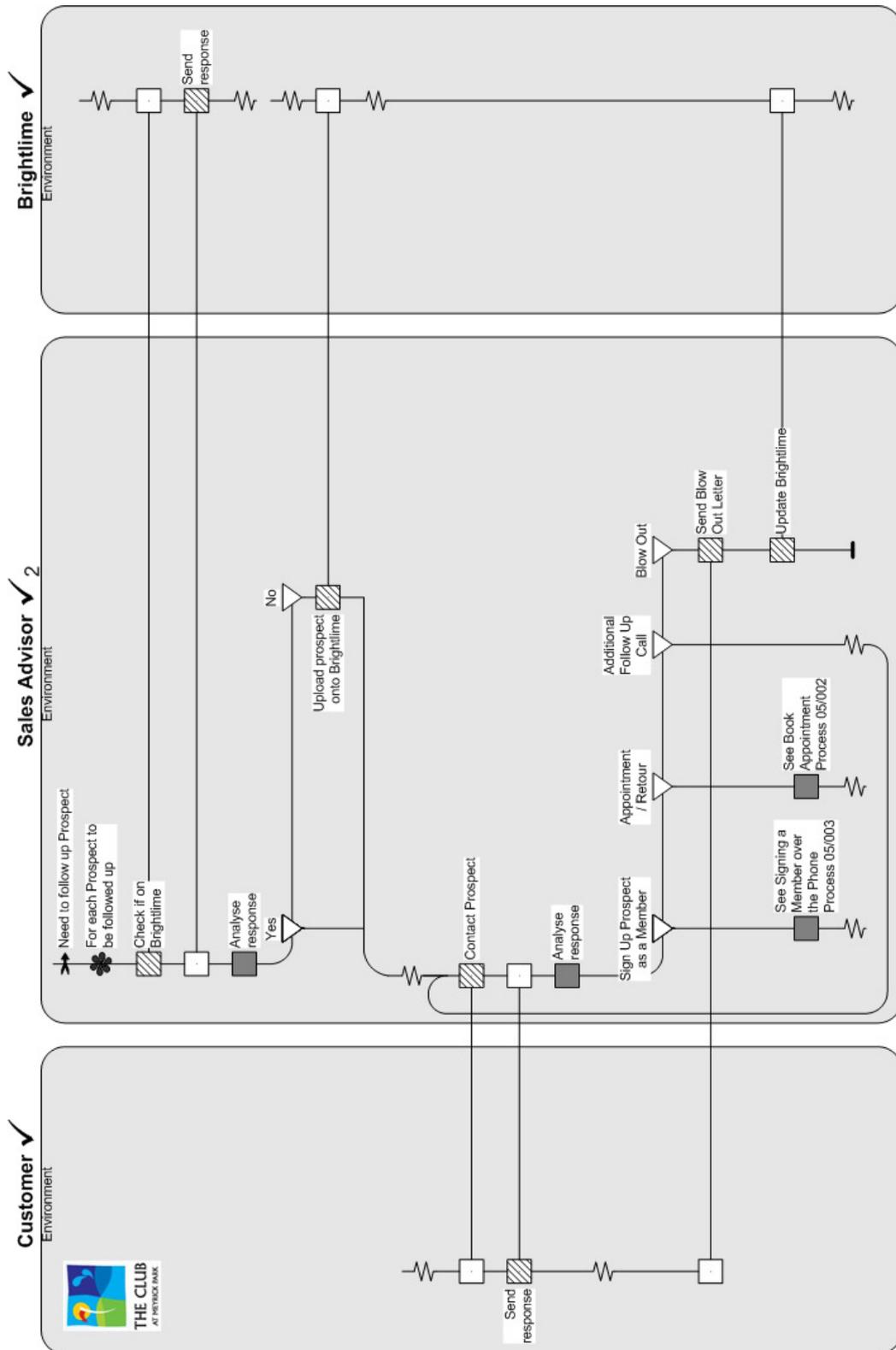


figure 4-1, Environment RAD for the Follow Up Call process.

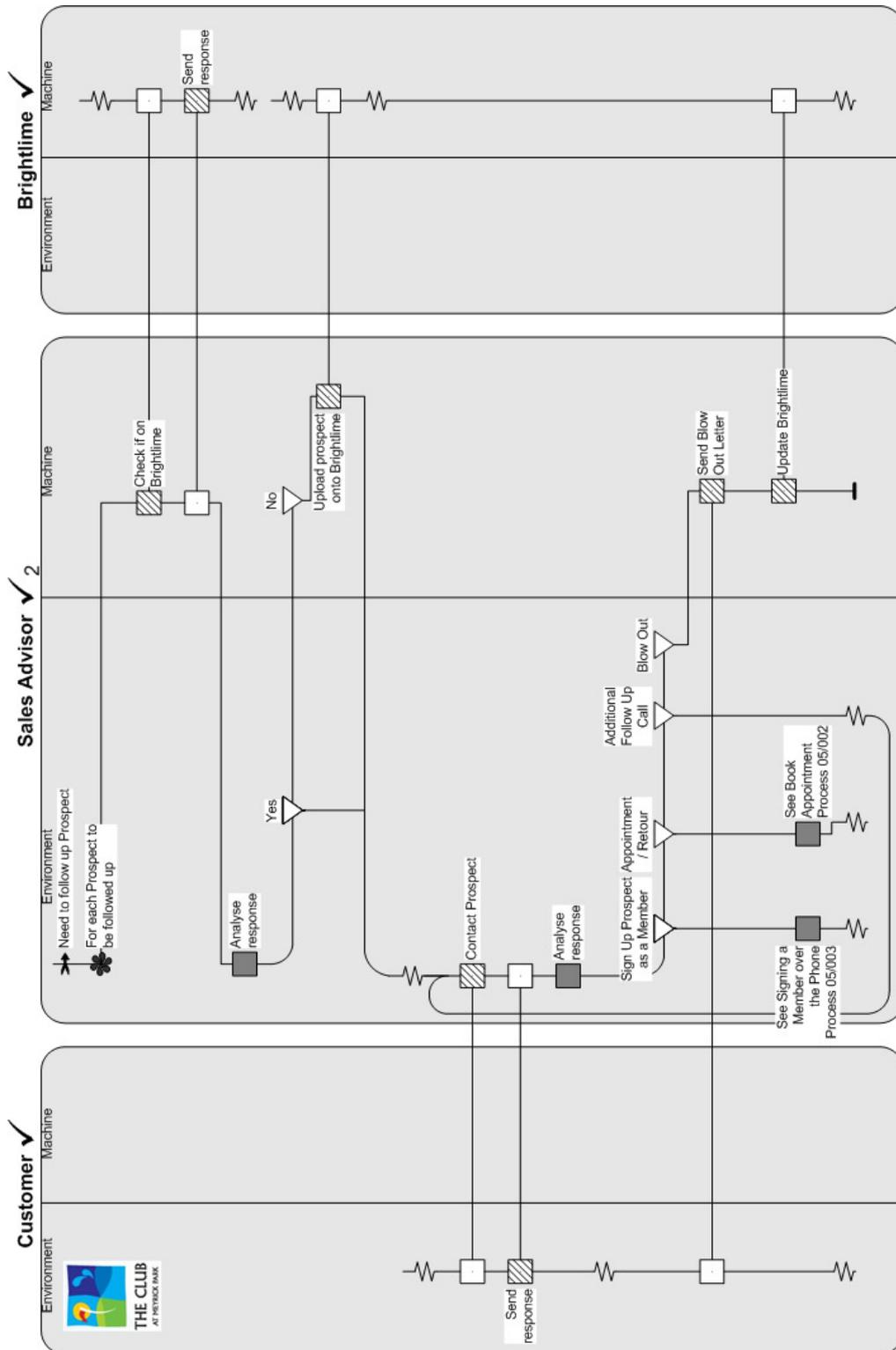


figure 4-2, Shared RAD for the Follow Up Call process.

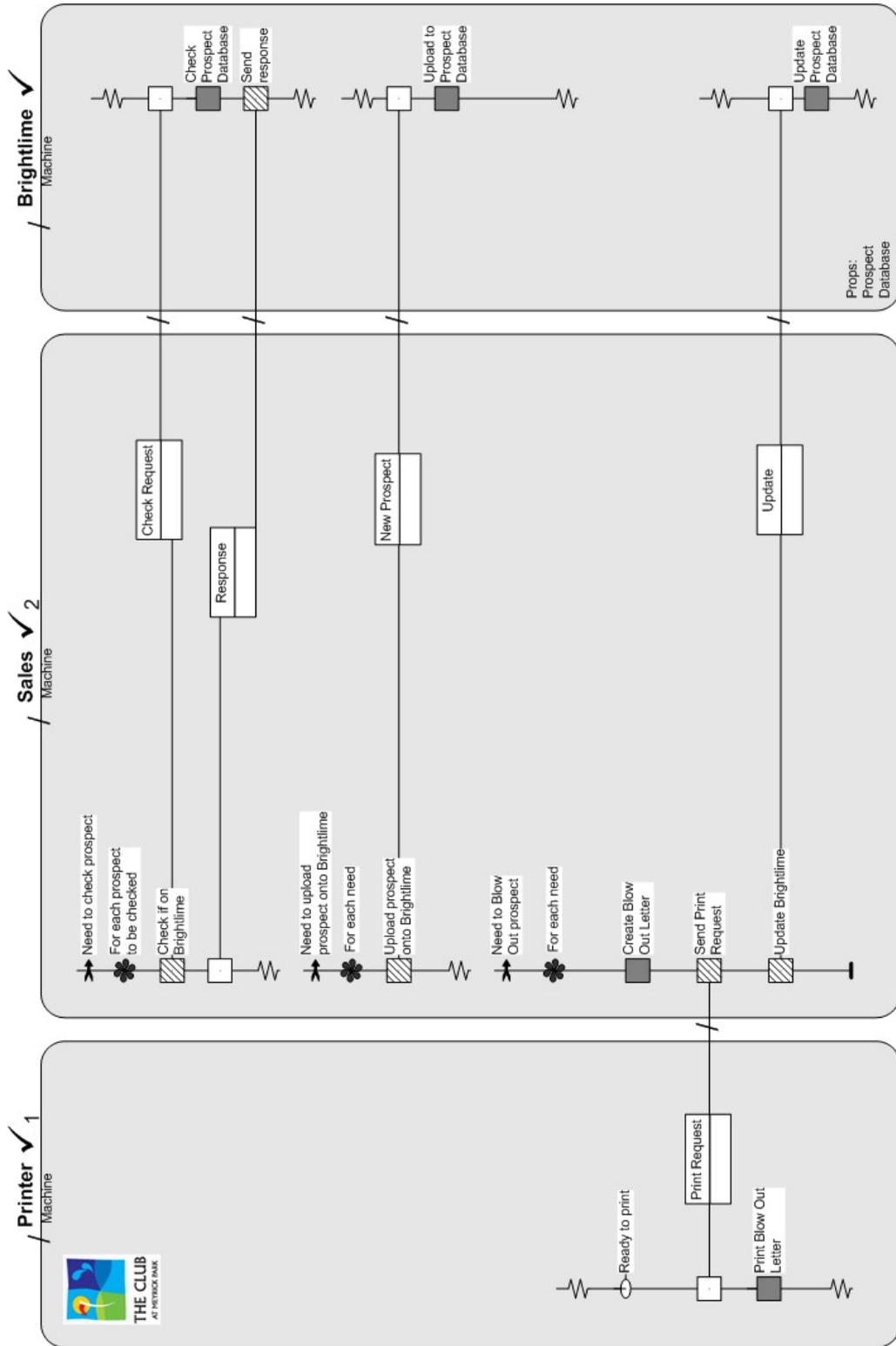


figure 4-3, Machine RAD for the Follow Up Call process.

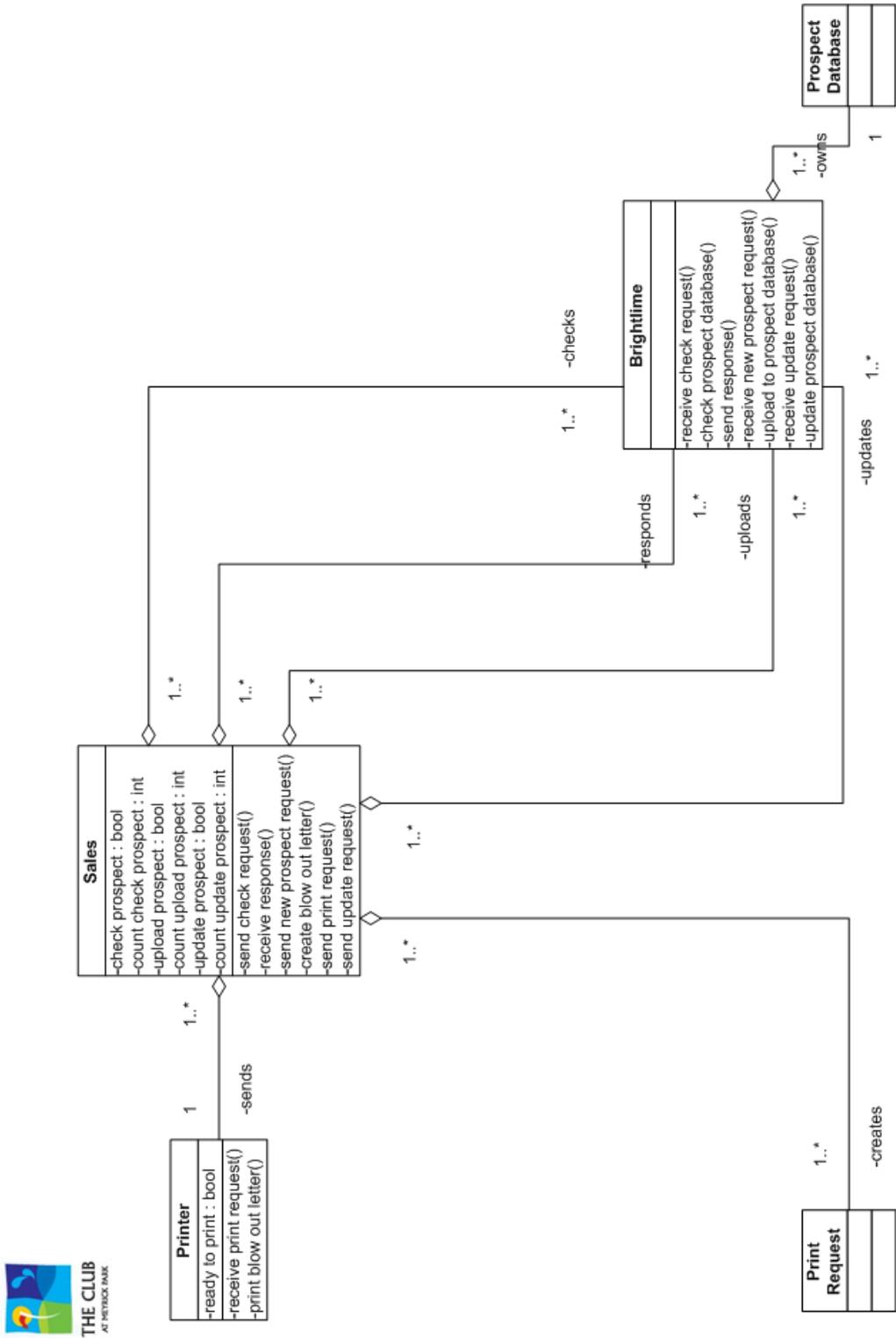


figure 4-4, UML Class Diagram for the Follow Up Call process Machine RAD.

Appendix V

QVT-Relations for rad2umlcd Transformations

Role2Class

The first rule describes an unconditional mapping of a RAD *role* to a UML *class* as illustrated in figure 5-1.

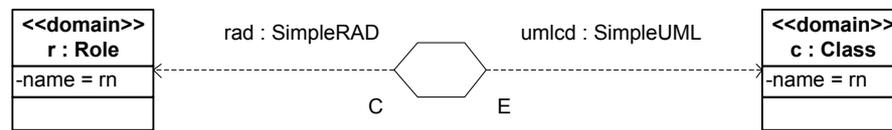


figure 5-1, a RAD role to a UML class relation.

Most cases will be required to hold true when the *Role2Class* relation holds between the role and the class containing the activity. Since this is the case, the *Role2Class* relation is defined as a top-level relation, requiring to be held true for *all* relations in a transformation. This relation is demonstrated in QVT-R textual syntax below.

```

top relation Role2Class /* map each role to a class*/
rn = 'String';
  {
    checkonly domain rad r:Role
      {
        name = rn
      }
    enforce domain umlcd c:Class
      {
        name = rn
      }
  }

```

That is, pattern *r* binds the variable *rn* to the role model element *name* and pattern *c* binds the same variable *rn* to the class model element *name*, resulting in both *name* model elements in the role and class elements of the *rad* and *umlcd* candidate models containing the same information, i.e. *rn*. Each domain is annotated as *Checkonly* (for verification against the rules) and *Enforce* (to create or change the target model according to the relation – that is from the RAD to the UML).

There is no complication involved in defining *Role2Class* transformation; only a single attribute (*name*) is required in the relation. *When* and *Where* clauses can also be applied to “explicitly constrain the relation” (Ignjatovic 2006) and may be formed using “arbitrary OCL expressions in addition to... relation invocation expressions” (OMG 2008b). However, no such expressions are required to further elucidate this rule.

IndependentActivity2Operation

Since the *Role2Class* relation has been defined as a top top-level relation, the *IndependentActivity2Operation* relation is required to hold only when the *Role2Class* relation holds between the role containing the independent activity and the class containing the operation. This relation is illustrated in figure 5-2.

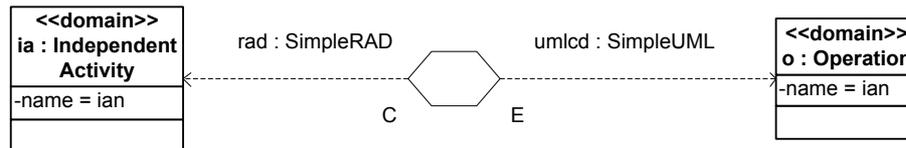


figure 5-2, a RAD independent activity to a UML operation relation.

The QVT-R that represents this relation is not defined as a top level relation, as it is not a requirement for all other relations, since the transformation is dependent on only the involved independent activity and associated operation. This is reflected in the following description.

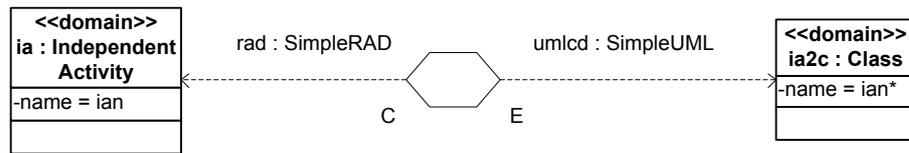
```

relation IndependentActivity2Operation /*map each independent activity to an operation*/
ian = 'String';
  {
    checkonly domain rad ia:IndependentActivity
      {
        name = ian
      }
    enforce domain umlcd o:Operation
      {
        name = ian
      }
  }

```

Note that in the guiding rules, it is advised that a new class and association may be generated should an object result from the independent activity. For example, the independent activity *writes report* could generate a *report* class and an association to that class. Therefore, additional relations need to be defined.

In this case, the QVT is described by the following two relations.



— where —

IndependentActivity2Association (ia, ia2a)

figure 5-3, a RAD independent activity to a UML class relation.

Figure 5-3 highlights the relation that describes how a class might be generated from an *independent activity*. When the relation is true, the *Where* clause is used to apply further conditions. It is used in this case to describe that whenever the *IndependentActivity2Class* holds, the *IndependentActivity2Association* relation must also hold. The associated QVT-R description is provided below.

relation IndependentActivity2Class /*map each independent activity to a Class*/

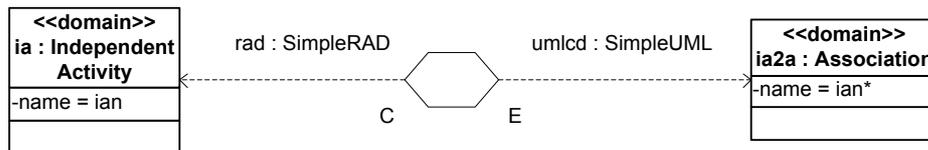
ian = 'String';

```

{
    checkonly domain rad ia:IndependentActivity
    {
        name = ian
    }
    enforce domain umlcd ia2c:Class
    {
        name = ian* /*must be amended to reflect object (noun)*/
    }
    where
    {
        IndependentActivity2Association (ia, ia2a);
    }
}

```

Figure 5-4 illustrates the graphical syntax for the second relation of *IndependentActivity2Association*.



— when _____

IndependentActivity2Class (ia, ia2c)

— where _____

✓2Multiplicity (ti, mu)

figure 5-4, a RAD independent activity to a UML association relation.

Similarly, this relation is defined with the associated *Where* clause requiring that where the *IndependentActivity2Association* relation holds, *IndependentActivity2Class* must hold. The description of the *✓2Multiplicity* relation is included here to account for multiplicity in an association, which is described later in this appendix. The QVT-R for this relation is described below.

```

relation IndependentActivity2Association /*map each independent activity to an association*/
ian = 'String';
  {
    checkonly domain rad ia:IndependentActivity
      {
        name = ian
      }
    enforce domain umlcd ia2a:Association
      {
        name = ian* /*must be amended to reflect transitive verb*/
      }
    when
      {
        IndependentActivity2Class (ia, ia2c)
      }
    where
      {
        ✓2Multiplicity (ti, mu);
      }
  }

```

Although these two relations are useful in describing the rule, the choice of which rule to follow is the onus of the user and therefore any software implementing these rules needs to account for that by employing a mechanism to accept user input in deciding transformation outcomes via on screen interactions, perhaps in the form of a *wizard*. Such option is not really accounted for in QVT-R. Therefore, it is proposed that the *IndependentActivity2Operation* relation be applied in the absence of user input.

DescriptorState2Attribute

The next rule to be described applies to the descriptor state of a RAD. It is stated that only a descriptor state is to be realised in a Class Diagram as an attribute of a related class. This is because the *SimpleUML* metamodel has no representation for other state types. Since this is the case, no rule is required to be defined in QVT to relate looping or line states. This relation is visualised in figure 5-5.

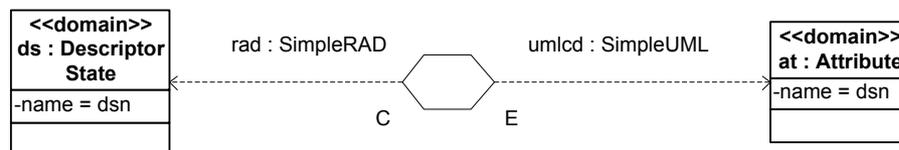


figure 5-5, a RAD descriptor state to a UML attribute relation.

This visualisation corresponds to the QVT-R description below.

```

relation DescriptorState2Attribute /* map each descriptor state to an attribute*/
dsn = 'String';
  {
    checkonly domain rad ds:DescriptorState
      {
        name = dsn
      }
    enforce domain umlcd at:Attribute
      {
        name = dsn
      }
  }

```

A class to which the attribute is attached is required to have already been created which is addressed by the top relation *Role2Class*, thereby negating the need for the *When* clause. Since this relation is a direct one-to-one translation, further constructs are not required, although this relation could be extended via the use of

operator types in similar to the *CaseRefinement2Attribute* relation (see the following section for a detailed description of such an extension).

CaseRefinement2Attribute

This rule is used in similar to the previous in that a UML class attribute is arbitrarily applied to represent every case refinement (alternative) appearing in the RAD. This is so that the system will have an awareness of state. Figure 5-6 demonstrates this rule in visual QVT-R, with the QVT-R textual form following it.

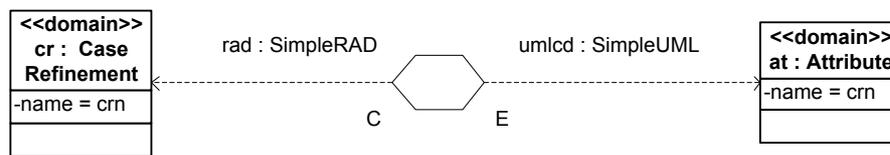


figure 5-6, a RAD case refinement to a UML attribute relation.

relation CaseRefinement2Attribute /* map each case refinement to an attribute*/

crn = 'String';

```

{
    checkonly domain rad cr:CaseRefinement
        {
            name = crn
        }
    enforce domain umlcd at:Attribute
        {
            name = crn
        }
}

```

Again, a class to which the attribute is attached is required which is addressed by the top relation *Role2Class* and therefore the *When* clause is not required in this case. In reality, this rule could be further complicated due to the nature of the case refinement construct. For example, the case refinement can have many alternate elements and therefore, an attribute may be required to be defined for each case refinement element (or option) rather than just a single attribute. Further to this, the data type of the element could also have been defined as a *Boolean* type to represent the true or false logic system associated with this type of attribute. The QVT-R for extending this definition follows the graphical syntax given in figure 5-7.

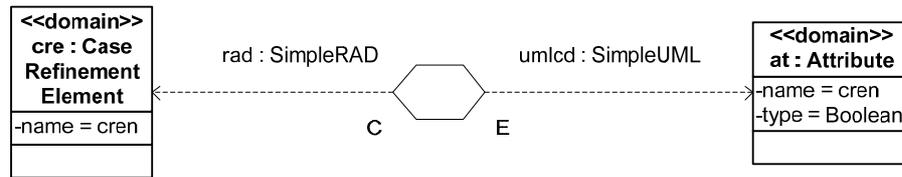


figure 5-7, a RAD case refinement element to a UML attribute relation.

```

relation CaseRefinementElement2Attribute /* map each case refinement element to an
attribute*/
cren = 'String';
  {
    checkonly domain rad cre:CaseRefinementElement
    {
      name = cren
    }
    enforce domain umlcd at:Attribute
    {
      name = cren,
      type = 'Boolean'
    }
  }

```

However, these extensions have been discounted here as they are inherently design issues and the scope of this rule suggests only that the case refinement is addressed by an attribute in the UML (for review in subsequent design), and therefore, further constructs are not required, or could be included only as part of an interactive definition dependent on user input.

PartRefinement2Attribute

This next rule can be applied in much the same way as the previous, in that for every occurrence of a part refinement, an attribute is applied within the class defined by the top relation. This relation is visualised graphically in figure 5-8, with the QVT-R description below it.

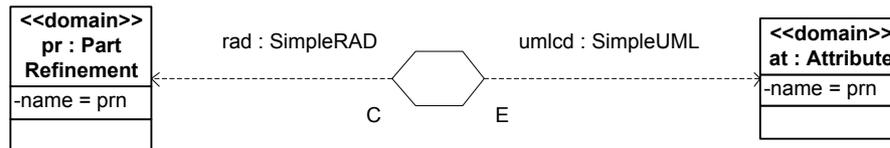


figure 5-8, a RAD part refinement to a UML attribute relation.

```

relation PartRefinement2Attribute /* map each part refinement to an attribute*/
prn = 'String';
  {
    checkonly domain rad pr:PartRefinement
      {
        name = prn
      }
    enforce domain umlcd at:Attribute
      {
        name = prn
      }
  }

```

Similarly, as with case refinements, this description could be extended to account for each part refinement (concurrent thread) element and *Boolean* operator as shown in figure 5-9, with the associated QVT description following it.

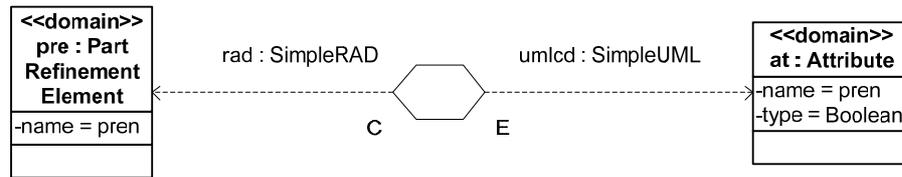


figure 5-9, a RAD case refinement element to a UML attribute relation.

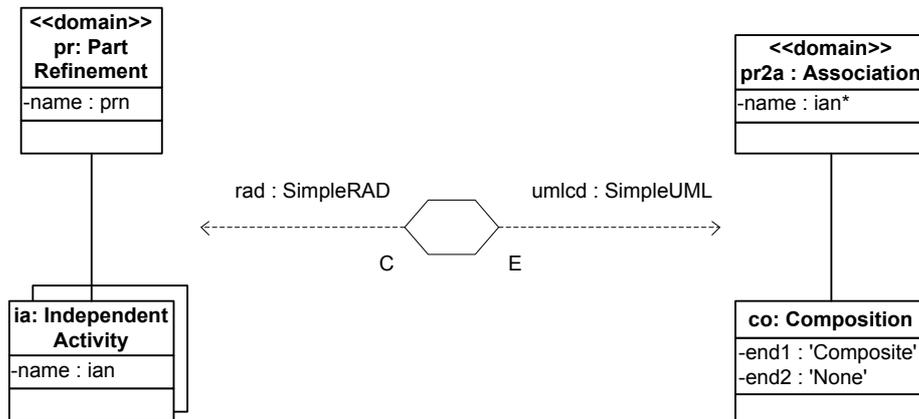
```

relation PartRefinementElement2Attribute /* map each part refinement element to an
attribute*/
pre = 'String';
  {
    checkonly domain rad pre:PartRefinementElement
      {
        name = pren
      }
    enforce domain umlcd at:Attribute
      {
        name = pren,
        type = 'Boolean'
      }
  }

```

Again, this extension is discounted since the scope of the rule suggests only that the part refinement be addressed by an attribute in the UML, which can then later be modified at design-time, if required.

A further construct is required given the consideration that composition may also be derived from relationships that are in part-refinement. For example, in the case that *Report A* and *Report B* are both written simultaneously and then merged into another entity, *Report C*. What this is really suggesting is that for each independent activity that results in the merging of part-refinement activities, composition is implied. Therefore, the relation given in figure 5-10 is required to hold.



— when _____

PartRefinement2Attribute (pr, at)

— where _____

IndependentActivity2Class (ia, ia2c)
IndependentActivity2Operation (ia, o)
✓2Multiplicity (ti, mu)

figure 5-10, a RAD part refinement to a UML composition relation.

figure 5-10 shows that for each set of independent activities contained within a part refinement, where classes and operations are also generated for those activities, composite associations must also result. The following is the QVT-R textual description that relates to figure 5-10.

```

relation PartRefinement2Composition /* map each part refinement to a composition*/
prn = 'String';
ian = 'String';
{
    checkonly domain rad pr:PartRefinement
    {
        name = prn,
        independentactivity = ia: Set (IndependentActivity)
        {
            name = ian
        }
    }
    enforce domain umlcd pr2a:Association
    {

```

```

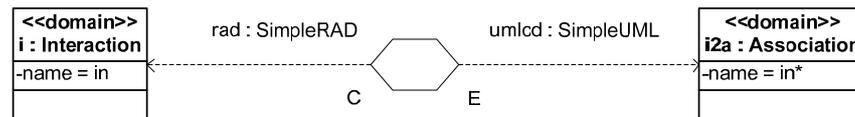
name = ian*, /*must be amended to reflect transitive verb*/
composition = co: Composition
    {
        end1 = 'Composite',
        end2 = 'None'
    }
}
when
{
    PartRefinement2Attribute (pr, at);
}
where
{
    IndependentActivity2Class (ia, ia2c) and
IndependentActivity2Operation (ia, o) and  $\surd$ 2Multiplicity (ti,
mu);
}
}

```

When specified conditions are required to be true, the *When* clause is used as a condition to apply the relation. In this case, the relation *PartRefinement2Composition* needs to hold only when the *PartRefinement2Attribute* relation holds (that is, when a part refinement is identified and associated with an attribute). However, as with previous examples, this type of relation is dependent on the guidance of the user.

Interaction2Association

This rule suggests that for each RAD interaction, the UML Class Diagram will exhibit an association between classes involved (as defined by the top relation *Role2Class*) and operations for both driving and passive RAD interaction nodes within those classes. Therefore, two relations are described visually in figures 11 and 12, with the textual counterpart following each.



— where —

Interaction2Operation (i, i2o)
✓2Multiplicity (ti, mu)

figure 5-11, a RAD interaction to a UML association relation.

relation Interaction2Association /*map each independent activity to an association*/

in = 'String';

{

checkonly domain rad i:Interaction

{

name = in

}

enforce domain umlcd i2a:Association

{

name = in* /*must be amended to reflect transitive verb*/

}

where

{

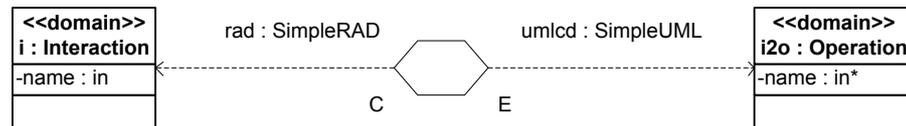
Interaction2Operation (i, i2o) and

✓2Multiplicity (ti, mu);

}

}

Note: As previously mentioned, the *Where* clause here states that this relation is extended by the \checkmark *2Multiplicity* relation to account for multiplicity in an association. The description of this relation is found later in this appendix.



— when —

Interaction2Association (i, i2a)

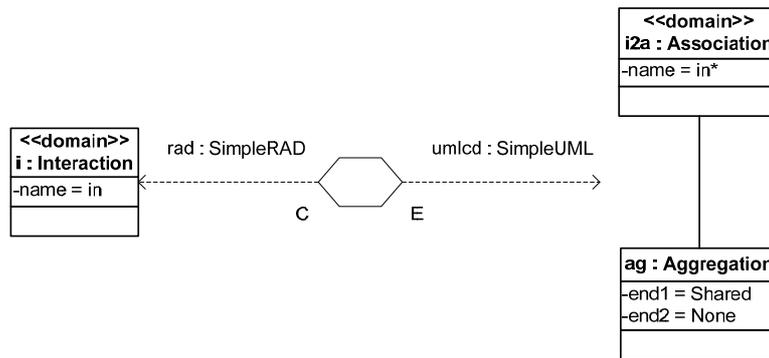
figure 5-12, a RAD interaction to a UML operation relation.

```

relation Interaction2Operation /*map each interaction to an operation*/
in = 'String';
  {
    checkonly domain rad i:Interaction
    {
      name = in
    }
    enforce domain umlcd i2o:Operation
    {
      name = in* /*must be amended to reflect transitive verb and
      direct object (noun)*/
    }
    when
    {
      Interaction2Association (i, i2a);
    }
  }

```

It is stated that aggregation may exist in association between classes if role interactions are exclusively between two roles. Therefore, further user input is required to derive such aggregation in defining the direction of aggregation. In this case, the relation given in figure 5-13 and description below it would be required to hold.



— where —

Interaction2Operation (i, i2o)
✓2Multiplicity (ti, mu)

figure 5-13, a RAD interaction to a UML operation relation.

relation Interaction2Aggregation /* map each interaction to an aggregation*/

in = 'String';

```

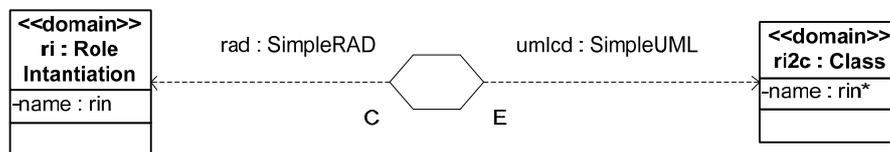
{
    checkonly domain rad i:Interaction
    {
        name = in
    }
    enforce domain umlcd i2a:Association
    {
        name = in* /*must be amended to reflect transitive verb*/
        aggregation = ag: Aggregation
        {
            end1 = 'Shared',
            end2 = 'None'
        }
    }
    where
    {
        Interaction2Operation (i, i2o) and
        ✓2Multiplicity (ti, mu);
    }
}

```

Again, the existence of such aggregation ought to be defined by the user, since it may not be the case that aggregation should arbitrarily exist in all cases that roles interact with such exclusivity since some interactions may be hidden, or be part of alternate processes.

RoleInstantiation2Class

The role instantiation RAD notation describes an instance where a role may start an alternate role instance, such as a *Project Manager* role instantiating an instance of a *Contractor* role (to work on a particular project). In this case, a new class representing that instance is required and therefore the following relation given in figure 5-14 with the textual description following it is required to hold.



— where —

RoleInstantiation2Operation (ri, ri2o)
RoleInstantiation2Association (ri, ri2a)

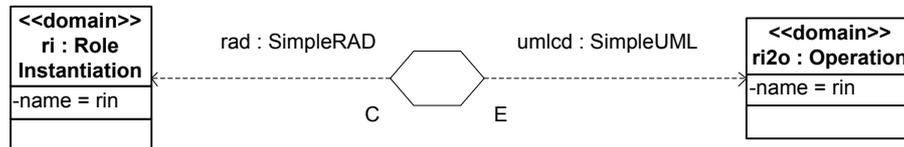
figure 5-14, a RAD role Instantiation to a UML class relation.

```

relation RoleInstantiationRole2Class /*map each role instantiation to a class*/
rin = 'String';
  {
    checkonly domain rad ri:RoleInstantiation
      {
        name = rin
      }
    enforce domain umlcd ri2c:Class
      {
        name = rin* /*must be amended to reflect object (noun)*/
      }
    where
      {
        RoleInstantiation2Operation (ri, ri2o) and
        RoleInstantiation2Association (ri, ri2a);
      }
  }

```

Where this relation holds, relations to create an operation in the top relation class, and an association between them, must also hold. Therefore, the *RoleInstantiation2Operation* and *RoleInstantiation2Association* are also required. Graphical descriptions for these relations are given in figures 5-15 and 5-16, with associated textual descriptions following them.



— when —————

RoleInstantiation2Class (ri, ri2c)

— where —————

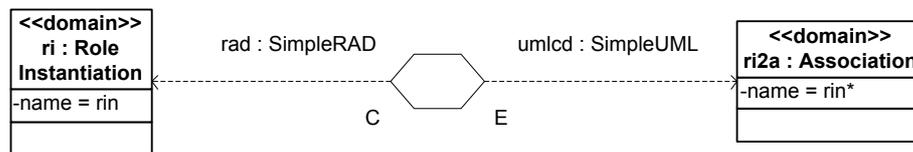
RoleInstantiation2Association (ri, ri2a)

figure 5-15, a RAD role instantiation to a UML operation relation.

```

relation RoleInstantiation2Operation /*map each role instantiation to an operation*/
rin = 'String';
{
    checkonly domain rad ri:RoleInstantiation
    {
        name = rin
    }
    enforce domain umlcd ri2o:Operation
    {
        name = rin
    }
    when
    {
        RoleInstantiation2Class (ri, ri2c);
    }
    where
    {
        RoleInstantiation2Association (ri, ri2a);
    }
}

```



— when —————

RoleInstantiation2Class (ri, ri2c)
RoleInstantiation2Operation (ri, ri2o)

— where —————

✓2Multiplicity (ti, mu)

figure 5-16, a RAD role instantiation to a UML class relation.

relation RoleInstantiation2Association /*map each role instantiation to an association*/
rin = 'String';

```

{
    checkonly domain rad ri:RoleInstantiation
        {
            name = rin
        }
    enforce domain umlcd ri2a:Association
        {
            name = rin* /*must be amended to reflect transitive verb*/
        }
    when
        {
            RoleInstantiation2Class (ri, c) and
            RoleInstantiation2Operation (ri, ri2o);
        }
    where
        {
            ✓2Multiplicity (ti, mu);
        }
}

```

Trigger2Attribute

Triggers are typically used to start a string of activity within a role on the basis of some event. A Class Diagram has no equivalent notation for a trigger. However, it could be useful to record whether or not a trigger is active (i.e. whether or not the event that instigates the trigger has occurred) to reflect on the state of the class. Therefore, it is suggested that each trigger element should be transformed into an attribute of the class created by the top relation. Figure 5-17 describes the required relation graphically, followed by the textual QVT-R description.

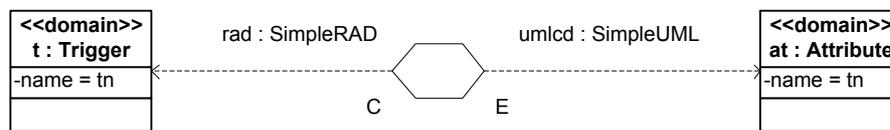


figure 5-17, a RAD trigger to a UML attribute relation.

relation Trigger2Attribute /* map each trigger to an attribute*/

tn = 'String';

```

{
    checkonly domain rad t:Trigger
    {
        name = tn
    }
    enforce domain umlcd at:Attribute
    {
        name = tn
    }
}

```

As previously noted with the *CaseRefinement2Attribute* and *PartRefinement2Attribute* relations, the definition of the attribute can, if required, be extended to reflect the *Boolean* operator type that would be associated with the true or false logic system of this attribute (see sections of this appendix on *CaseRefinement2Attribute* and *PartRefinement2Attribute* for more detailed insights into this extension).

Replication2Attribute

A replication node is used to record a count and is commonly associated with a particular string of activity associated with a particular role. This replication can be recorded as an *Integer* attribute within the class which resulted from the top relation *Role2Class*. Figure 5-18 presents the graphical syntax for this relation, with the textual description following it.

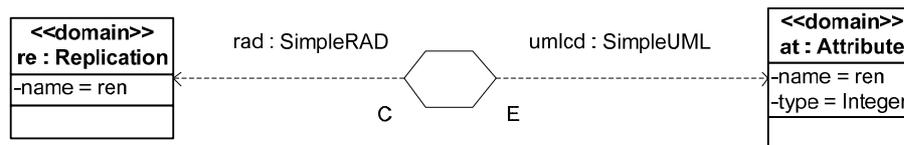


figure 5-18, a RAD replication to a UML attribute relation.

```

relation Replication2Attribute /* map each replication to an attribute*/
ren = 'String';
  {
    checkonly domain rad re:Replication
      {
        name = ren
      }
    enforce domain umlcd at:Attribute
      {
        name = ren,
        type = 'Integer'
      }
  }

```

Of course, the decision of which type to use at design time will be the onus of the developer, and eventually the chosen platform. For instance, an *Unsigned Integer* may be the preference in the case that a negative value is not required since it is representative of a positive count, resulting in an increase in the count range.

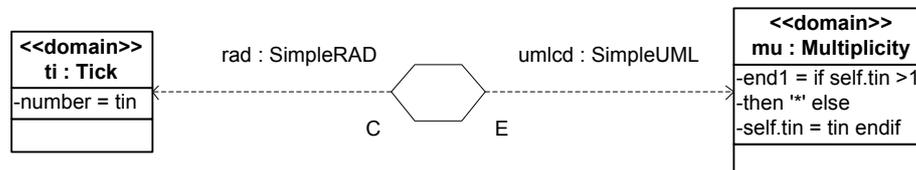
Undefined

The undefined element of the RAD has no counterpart representation in the *SimpleUML* metamodel and therefore no relation is required to hold when this element is encountered. It is nevertheless important to check the context in which it is used as it may define an alternate Class Diagram. That is, a string of activity

unrelated to the requirement of the Class Diagram in question, but significant and required by another Class Diagram. Of course, the decision of what might occur in such an instance relies on user input and such, must be a consideration of any software being developed to implement these rules; for the scope of this definition however, they are discounted.

✓2Multiplicity

In a RAD, the ✓ element is indicative of the number of instances particular to a role. It is therefore adequate to derive UML multiplicity associated with the classes enforced by the top level relation *Role2Class*. This is depicted in figure 5-19 with textual QVT below it.



— when —

**IndependentActivity2Association (ia, ia2a) or
PartRefinement2Composition (pr, pr2a) or
Interaction2Association (i, i2a) or
Interaction2Aggregation (l, l2a) or
RoleInstantiation2Association (ri, ri2a) or
Prop2Association (p, p2a)**

figure 5-19, a RAD ✓ to UML multiplicity relation.

```

relation ✓2Multiplicity /* map each ✓ to multiplicity*/
tin = 'String';
{
    checkonly domain rad ti:Tick
    {
        number = tin
    }
    enforce domain umlcd mu:Multiplicity
    {
        end1 = if self.tin > 1 then '**' else self.tin = tin endif
    }
}
when

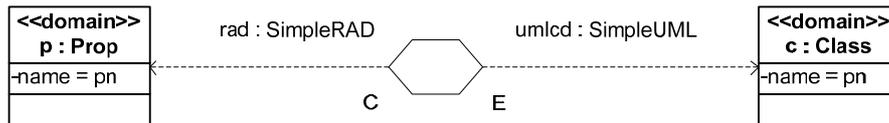
```

```
    {  
        IndependentActivity2Association (ia, ia2a) or  
        PartRefinement2Composition (pr, pr2a) or  
        Interaction2Association (i, i2a) or  
        Interaction2Aggregation (l, i2a) or  
        RoleInstantiation2Association (ri, ri2a) or  
        Prop2Association (p, p2a);  
    }  
}
```

Since the Multiplicity element of the *SimpleUML* metamodel is related to an association, the \checkmark 2Multiplicity relation must hold whenever the *IndependentActivity2Association*, *PartRefinement2Composition*, *Interaction2Association*, *Interaction2Aggregation*, *RoleInstantiation2Association* or *Prop2Association* relations hold (i.e. whenever an association is enforced, multiplicity is also defined).

Prop2Class

The *Prop2Class* relation is defined to show how, for each RAD prop used by a role, a new UML class is enforced to represent that prop. Typically, such a class would represent a data object that is manipulated by the class generated by the top relation *Role2Class*. The graphical syntax for this relation is given in figure 5-20, with the textual description following it.



— where —

Prop2Association (p, p2a)

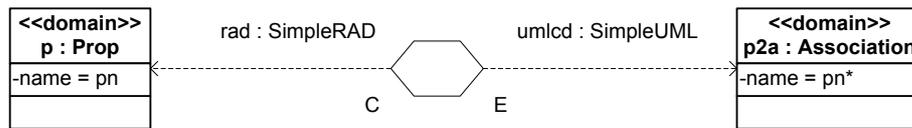
figure 5-20, a RAD prop to a UML class relation.

```

relation Prop2Class /*map each prop to a class*/
pn = 'String';
  {
    checkonly domain rad p:Prop
      {
        name = pn
      }
    enforce domain umlcd c:Class
      {
        name = pn
      }
    where
      {
        Prop2Association (p, p2a);
      }
  }

```

Where this relation holds, a relation to create an association between the new class and the class derived from the top relation *Role2Class* must also hold. Therefore, the *Prop2Association* relation is required, and defined below in figure 5-21, with the textual description following it.



— when —————

Prop2Class (p, c)

— where —————

✓2Multiplicity (ti, mu)

figure 5-21, a RAD prop to a UML class relation.

```

relation Prop2Association /*map each prop to an association*/
pn = 'String';
{
    checkonly domain rad p:Prop
    {
        name = pn
    }
    enforce domain umlcd p2a:Association
    {
        name = pn* /*must be amended to reflect transitive verb*/
    }
    when
    {
        Prop2Class (p, c);
    }
    where
    {
        ✓2Multiplicity (ti, mu);
    }
}

```

Stop2Attribute

There is no real direct relationship between RAD stop elements and the UML Class Diagram since the concept of stop is to stop a running thread in a state based system. It could however be useful to record the state in classes derived from the top relation *Role2Class* since events may depend on the knowledge of the completion event. For example, certain operations may be required to be locked until the stop state of a thread has been reached. This relation is presented graphically in figure 5-22, with the textual description that follows.

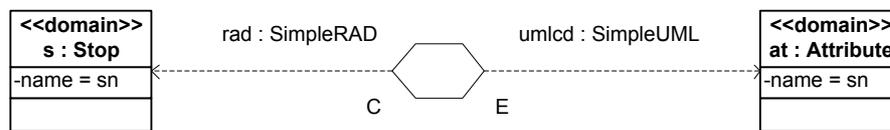


figure 5-22, a RAD stop a UML attribute relation.

relation Stop2Attribute /* map each stop to an attribute*/

sn = 'String';

```

{
    checkonly domain rad s:Stop
    {
        name = sn
    }
    enforce domain umlcd at:Attribute
    {
        name = sn
    }
}

```

Again, this is very contextual and it may be the case that the stop node could be ignored completely, dependent on user intervention. Further to this, as previously noted in other examples, this relation could be extended to define the attribute type *Boolean*, but again, this requires intervention from the user in context of the application and/or is a decision better made at design-time (see the sections on *CaseRefinement2Attribute* and *PartRefinement2Attribute* for further detail).

Note2UMLNote

A note on a RAD could refer to anything relating to the RAD, or other. Therefore, this relation is wholly dependent on the situation and requires user action in assessing the context and deciding on whether or not the note is required to be represented in the resulting Class Diagram. If the relation is required to hold, the following QVT-R definition is to be used (see figure 5-23 for the graphical description, and the textual description that follows it).

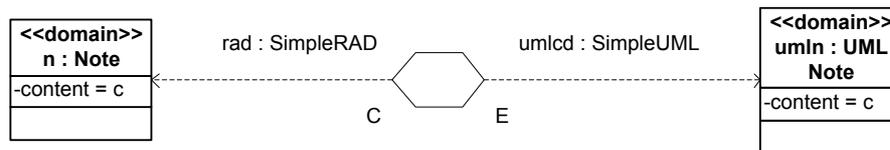


figure 5-23, a RAD note to a UML note relation.

relation Note2UMLNote /* map each rad note to a UML note*/

c = 'String';

```

{
    checkonly domain rad n:Note
    {
        content = c
    }
    enforce domain umlcd umlIn:UMLNote
    {
        content = c
    }
}

```

Since this represents a direct relation between note elements of the *SimpleRAD* and *SimpleUML* metamodels without calling other relations, no further constructs are required to define it. In a larger context, this relation could be defined as a top-relation, but since the relation is not called by and does not call any other relation, it is discounted in isolation. Of course, if the relation is not required, then any software implementation of these rules should accommodate this.

Appendix VI

VIDE PPT JAVA Transformation Engine

Source Code (VIDE 2010a)

```

package ppt;

import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.StringWriter;
import java.util.ArrayList;
import java.util.Random;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.ParserConfigurationException;
import javax.xml.transform.OutputKeys;
import javax.xml.transform.Result;
import javax.xml.transform.Source;
import javax.xml.transform.Transformer;
import javax.xml.transform.TransformerException;
import javax.xml.transform.TransformerFactory;
import javax.xml.transform.dom.DOMSource;
import javax.xml.transform.stream.StreamResult;
import org.w3c.dom.Attr;
import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.NamedNodeMap;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;
import org.xml.sax.SAXException;
import org.xml.sax.SAXParseException;
import java.util.*;

public class ParseVCLLFile {
    String packagedElementname;
    Document idoc;

    public ParseVCLLFile(String ixmlFile, String Package){
        File iF = new File(ixmlFile);
        idoc = parseXmlFile(iF, false);
        if(idoc !=null){
            packagedElementname = Package;
        }
    }

    public void createActivityDoc(String oxmlFile){
        Map<String,Number> idlist = new HashMap<String,Number>();
        Map<String,Number> splist = new HashMap<String,Number>();
        Map<String,Number> artifactsmap = new HashMap<String,Number>();
        Map<String,Number> verticesmap = new HashMap<String,Number>();
        artifactsmap.put("DataObject", 1);
        verticesmap.put("Activity", 1);
        verticesmap.put("SubProcess", 2);
        if (idoc == null)
            return;
        try {
            DocumentBuilder builder = DocumentBuilderFactory.newInstance().newDocumentBuilder();
            Document doc = builder.newDocument();
            Element root = doc.createElement("UML:Model");
            root.setAttribute("xmi:version", "2.1");
            root.setAttribute("xmlns:xmi", "http://schema.omg.org/spec/XMI/2.1");
            root.setAttribute("xmlns:UML", "http://www.eclipse.org/uml2/2.1.0/UML");
            root.setAttribute("xmi:id", getID());
            doc.appendChild(root);
            Element packagedElement = doc.createElement("packagedElement");
            packagedElement = doc.createElement("packagedElement");
        }
    }
}

```

```

packagedElement.setAttribute("xmi:type", "UML:Activity");
packagedElement.setAttribute("xmi:id", getID());
packagedElement.setAttribute("name", packagedElementname);
root.appendChild(packagedElement);

// set up all data here
NodeList poolist=idoc.getElementsByTagName("pools");
for (int i=0; i<poolist.getLength(); i++) {
    Element element = (Element)poolist.item(i);
    NodeList artifacts = element.getElementsByTagName("artifacts");
    for (int j=0; j<artifacts.getLength(); j++) {
        Element element2 = (Element)artifacts.item(j);
        Attr att = element2.getAttributeNode("xmi:type");
        if (att != null){
            if (artifactsmap.containsKey(att.getNodeValue())){
                Element node = doc.createElement("node");
                node.setAttribute("name",element2.getAttribute("name"));
                node.setAttribute("xmi:id",element2.getAttribute("xmi:id"));
                switch (artifactsmap.get(att.getNodeValue()).intValue()){
                    case 1:
                        node.setAttribute("xmi:type","UML:DataStoreNode");
                        Element node1=doc.createElement("upperBound");
                        node1.setAttribute("xmi:type", "UML:LiteralUnlimitedNatural");
                        node1.setAttribute("value", "");
                        node1.setAttribute("xmi:id", getID());
                        node.appendChild(node1);
                        break;
                    case 2:
                        node.setAttribute("xmi:type","UML:DecisionNode");
                        break;
                }
                packagedElement.appendChild(node);
                idlist.put(node.getAttribute("xmi:id"), 1);
            }
        }
    }
}
NodeList vertices = element.getElementsByTagName("vertices");
for (int j=0; j<vertices.getLength(); j++) {
    Element element2 = (Element)vertices.item(j);
    Attr att = element2.getAttributeNode("xmi:type");
    if (att != null){
        if (verticesmap.containsKey(att.getNodeValue())){
            switch (verticesmap.get(att.getNodeValue()).intValue()){
                case 1:
                    Attr att2 = element2.getAttributeNode("activityType");
                    if (att2 != null){
                        if (att2.getNodeValue().endsWith("Start Event")){
                            Element startnode = doc.createElement("node");
                            startnode.setAttribute("xmi:type","UML:InitialNode");
                            startnode.setAttribute("xmi:id",element2.getAttribute("xmi:id"));
                            startnode.setAttribute("name",att2.getNodeValue());
                            packagedElement.appendChild(startnode);
                            idlist.put(startnode.getAttribute("xmi:id"), 1);
                            Element element3 = (Element)element2.getParentNode();
                            if ((element3 != null) && splist.containsKey((element3.getAttribute("xmi:id")))){
                                // Comment this element
                                Element comment = doc.createElement("ownedComment");
                                comment.setAttribute("annotatedElement",element2.getAttribute("xmi:id"));
                                comment.setAttribute("xmi:id", getID());
                                Element body = doc.createElement("body");
                                body.setTextContent(element3.getAttribute("name"));
                            }
                        }
                    }
                }
            }
        }
    }
}

```

```

comment.appendChild(body);
packagedElement.appendChild(comment);
}
}
else
if (att2.getNodeValue().endsWith("End Event")){
Element endnode = doc.createElement("node");
endnode.setAttribute("xmi:type", "UML:ActivityFinalNode");
endnode.setAttribute("xmi:id", element2.getAttribute("xmi:id"));
endnode.setAttribute("name", att2.getNodeValue());
packagedElement.appendChild(endnode);
idlist.put(endnode.getAttribute("xmi:id"), 1);
}
else
/*if (att2.getNodeValue().endsWith("Event") ||
att2.getNodeValue().equals("Task"))*/{
Element activitynode = doc.createElement("node");
activitynode.setAttribute("xmi:type", "UML:StructuredActivityNode");
activitynode.setAttribute("xmi:id", element2.getAttribute("xmi:id"));
activitynode.setAttribute("name", att2.getNodeValue());
packagedElement.appendChild(activitynode);
idlist.put(activitynode.getAttribute("xmi:id"), 1);
}
} else
if (element2.getAttributeNode("name") != null){
Element activitynode = doc.createElement("node");
activitynode.setAttribute("xmi:type", "UML:StructuredActivityNode");
activitynode.setAttribute("xmi:id", element2.getAttribute("xmi:id"));
activitynode.setAttribute("name", element2.getAttribute("name"));
packagedElement.appendChild(activitynode);
idlist.put(activitynode.getAttribute("xmi:id"), 1);
}
break;
case 2:
Element activitynode = doc.createElement("node");
activitynode.setAttribute("xmi:type", "UML:StructuredActivityNode");
activitynode.setAttribute("xmi:id", element2.getAttribute("xmi:id"));
activitynode.setAttribute("name", element2.getAttribute("name"));
packagedElement.appendChild(activitynode);
idlist.put(activitynode.getAttribute("xmi:id"), 1);
splist.put(activitynode.getAttribute("xmi:id"), 1);
break;
}
}
}

// process all edges
NodeList edgeslist=idoc.getElementsByTagName("sequenceEdges");
for (int i=0; i<edgeslist.getLength(); i++) {
Element element = (Element)edgeslist.item(i);
Attr att = element.getAttributeNode("xmi:type");
if (att != null){
if (att.getNodeValue().equals("SequenceEdge")){
Element edgenode=doc.createElement("edge");
edgenode.setAttribute("xmi:type", "UML:ControlFlow");
edgenode.setAttribute("xmi:id", element.getAttribute("xmi:id"));
edgenode.setAttribute("source", element.getAttribute("source"));
edgenode.setAttribute("target", element.getAttribute("target"));
if (idlist.containsKey(edgenode.getAttribute("source")) &&
idlist.containsKey(edgenode.getAttribute("target"))){

```

```

        Element node=doc.createElement("guard");
        node.setAttribute("xmi:type", "UML:LiteralBoolean");
        node.setAttribute("value", "true");
        node.setAttribute("xmi:id", getID());
        edgenode.appendChild(node);
        node=doc.createElement("weight");
        node.setAttribute("xmi:type", "UML:LiteralInteger");
        node.setAttribute("value", "1");
        node.setAttribute("xmi:id", getID());
        edgenode.appendChild(node);
        packagedElement.appendChild(edgenode);
    }
}

//set up a transformer
TransformerFactory transfac = TransformerFactory.newInstance();
Transformer trans = transfac.newTransformer();
trans.setOutputProperty(OutputKeys.OMIT_XML_DECLARATION, "yes");
trans.setOutputProperty(OutputKeys.INDENT, "yes");
trans.setOutputProperty(OutputKeys.DOCTYPE_PUBLIC, "publicId");

// Prepare the DOM document for writing
Source source = new DOMSource(doc);

// Prepare the output file
File file = new File(oxmlFile);
Result result = new StreamResult(file);

// Write the DOM document to the file
Transformer xformer = TransformerFactory.newInstance().newTransformer();
//xformer.setOutputProperties(offormat)
xformer.transform(source, result);

}
catch (ParserConfigurationException e) {
    e.printStackTrace();
}
catch (TransformerException e){
    e.printStackTrace();
}
catch(Exception e){
    e.printStackTrace();
}
}

public void createClassDoc(String oxmlFile){
    if (idoc == null)
        return;
    try {
        DocumentBuilder builder = DocumentBuilderFactory.newInstance().newDocumentBuilder();
        Document doc = builder.newDocument();
        Element root = doc.createElement("UML:Model");
        root.setAttribute("xmi:version", "2.1");
        root.setAttribute("xmlns:xmi", "http://schema.omg.org/spec/XMI/2.1");
        root.setAttribute("xmlns:UML", "http://www.eclipse.org/uml2/2.1.0/UML");
        root.setAttribute("xmi:id", getID());
        doc.appendChild(root);
        Element packagedElement = doc.createElement("packagedElement");
        packagedElement.setAttribute("xmi:type", "UML:Package");
        packagedElement.setAttribute("xmi:id", getID());
    }
}

```

```

packagedElement.setAttribute("name", packagedElementname);
root.appendChild(packagedElement);

// extract pool elements
NodeList poolist = idoc.getElementsByTagName("pools");

// process all pool elements and create appropriate classes
for (int i=0; i<poolist.getLength(); i++) {
    Element element = (Element)poolist.item(i);
    Attr att = element.getAttributeNode("xmi:type");
    if (att != null){
        if (att.getNodeValue().equals("Pool")){
            Element child = doc.createElement("packagedElement");
            child.setAttribute("xmi:type", "UML:Class");
            child.setAttribute("xmi:id", element.getAttribute("xmi:id"));
            child.setAttribute("name", element.getAttribute("name"));

            // collect lanes and append them to pool
            NodeList lanelist = element.getElementsByTagName("lanes");
            for (int j=0; j<lanelist.getLength(); j++) {
                Element element2 = (Element)lanelist.item(j);
                Attr att2 = element2.getAttributeNode("xmi:type");
                if (att2 != null){
                    if (att2.getNodeValue().equals("Lane")){
                        Element lanechild = doc.createElement("ownedAttribute");
                        lanechild.setAttribute("xmi:id", element2.getAttribute("xmi:id"));
                        lanechild.setAttribute("name", String.format("Lane %d", j+1));
                        lanechild.setAttribute("visibility", "private");
                        child.appendChild(lanechild);
                    }
                }
            }

            // collect subprocesses and append them to pool
            NodeList vertices = element.getElementsByTagName("vertices");
            for (int j=0; j<vertices.getLength(); j++) {
                Element element2 = (Element)vertices.item(j);
                Attr att2 = element2.getAttributeNode("xmi:type");
                if (att2 != null){
                    if (att2.getNodeValue().equals("SubProcess")){
                        Element spchild = doc.createElement("ownedOperation");
                        spchild.setAttribute("xmi:id", element2.getAttribute("xmi:id"));
                        spchild.setAttribute("name", element2.getAttribute("name"));
                        child.appendChild(spchild);
                    }
                }
            }
            packagedElement.appendChild(child);
        }
    }

    // collect all roles and data objects and create appropriate classes
    NodeList artifactslist = element.getElementsByTagName("artifacts");
    for (int j=0; j<artifactslist.getLength(); j++) {
        Element element2 = (Element)artifactslist.item(j);
        Attr att2 = element2.getAttributeNode("xmi:type");
        if (att2 != null){
            if (att2.getNodeValue().equals("Role") || att2.getNodeValue().equals("DataObject")){
                Element child = doc.createElement("packagedElement");
                child.setAttribute("xmi:type", "UML:Class");
                child.setAttribute("xmi:id", element2.getAttribute("xmi:id"));
                child.setAttribute("name", element2.getAttribute("name"));
            }
        }
    }
}

```

```

packagedElement.appendChild(child);

// link roles with pools
if (att2.getNodeValue().equals("Role")){
Element link = doc.createElement("packagedElement");
link.setAttribute("xmi:type", "UML:Association");
link.setAttribute("xmi:id", getID());
Element ownedend = doc.createElement("ownedEnd");
ownedend.setAttribute("xmi:id", getID());
ownedend.setAttribute("type", element2.getAttribute("xmi:id"));
ownedend.setAttribute("name", "target");
ownedend.setAttribute("association", link.getAttribute("xmi:id"));
ownedend.setAttribute("aggregation", "composite");
link.appendChild(ownedend);
ownedend = doc.createElement("ownedEnd");
ownedend.setAttribute("xmi:id", getID());
ownedend.setAttribute("type", element.getAttribute("xmi:id"));
ownedend.setAttribute("name", "source");
ownedend.setAttribute("association", link.getAttribute("xmi:id"));
link.appendChild(ownedend);
packagedElement.appendChild(link);

// Comment this element
Element comment = doc.createElement("ownedComment");
comment.setAttribute("annotatedElement", child.getAttribute("xmi:id"));
comment.setAttribute("xmi:id", getID());
Element body = doc.createElement("body");
body.setTextContent(att2.getNodeValue());
comment.appendChild(body);
root.appendChild(comment);

//packagedElement.appendChild(comment);
}

}

// collect other links and annotations
}

//set up a transformer
TransformerFactory transfac = TransformerFactory.newInstance();
Transformer trans = transfac.newTransformer();
trans.setOutputProperty(OutputKeys.OMIT_XML_DECLARATION, "yes");
trans.setOutputProperty(OutputKeys.INDENT, "yes");
trans.setOutputProperty(OutputKeys.DOCTYPE_PUBLIC, "publicId");

// Prepare the DOM document for writing
Source source = new DOMSource(doc);

// Prepare the output file
File file = new File(oxmlFile);
Result result = new StreamResult(file);

// Write the DOM document to the file
Transformer xformer = TransformerFactory.newInstance().newTransformer();
//xformer.setOutputProperties(offormat)
xformer.transform(source, result);

}
catch (ParserConfigurationException e) {
    e.printStackTrace();
}
catch (TransformerException e){

```

```
        e.printStackTrace();
    }
    catch(Exception e){
        e.printStackTrace();
    }
}

public Document parseXmlFile(File xmlFile, boolean validating) {
    try {
        DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
        factory.setValidating(validating);
        Document doc = factory.newDocumentBuilder().parse(xmlFile);
        return doc;
    }
    catch (SAXException e) {
        System.out.println("File is not in the valid format");
        e.printStackTrace();
    }
    catch (ParserConfigurationException e) {
        e.printStackTrace();
    }
    catch (IOException e) {
        e.printStackTrace();
    }
    return null;
}

public String getID(){
    Random rand = new Random();
    String code = "_";
    for (int i = 0; i < 10 ; i++){
        int number = rand.nextInt(26)+65;
        code+= new Character((char)number).toString();
    }
    return code;
}
}
```

Appendix VII

Auto-generated XML for the modified UML
Class Diagram (figure 9.3.5.1)

```

<?xml version="1.0" encoding="UTF-8"?>
<UML:Model xmi:version="2.1" xmlns:xmi="http://schema.omg.org/spec/XMI/2.1" xmlns:UML="http://www.eclipse.org/uml2/2.1.0/UML"
xmi:id="_CMDJVBLDCQ">
  <packagedElement xmi:type="UML:Package" xmi:id="_SNMPMCKIP" name="Follow Up Call Process">
    <packagedElement xmi:type="UML:Class" xmi:id="s" name="Sales">
      <ownedAttribute xmi:id="l" name="check Prospect : bool" visibility="private"/>
      <ownedAttribute xmi:id="_imP44pi-Ed-bDfRognGdXA" name="count check prospect : int" visibility="private"/>
      <ownedAttribute xmi:id="_IIUNApi-Ed-bDfRognGdXA" name="upload prospect : bool" visibility="private"/>
      <ownedAttribute xmi:id="_nfHsUpi-Ed-bDfRognGdXA" name="count upload prospect : int" visibility="private"/>
      <ownedAttribute xmi:id="_7xUaMpi-Ed-bDfRognGdXA" name="update prospect : bool" visibility="private"/>
      <ownedAttribute xmi:id="_U04wpi-Ed-bDfRognGdXA" name="count update prospect : int" visibility="private"/>
      <ownedOperation xmi:id="scr" name="send check request" visibility="private"/>
      <ownedOperation xmi:id="rr" name="receive response" visibility="private"/>
      <ownedOperation xmi:id="snpr" name="send new prospect request" visibility="private"/>
      <ownedOperation xmi:id="cbol" name="create blow out letter" visibility="private"/>
      <ownedOperation xmi:id="spr" name="send print request" visibility="private"/>
      <ownedOperation xmi:id="sur" name="send update request" visibility="private"/>
    </packagedElement>
    <packagedElement xmi:type="UML:Class" xmi:id="p" name="Printer">
      <ownedAttribute xmi:id="_6mt6Mji8Ed-bDfRognGdXA" name="ready to print : bool" visibility="private"/>
      <ownedOperation xmi:id="rpr" name="receive print request" visibility="private"/>
      <ownedOperation xmi:id="pbol" name="print blow out letter" visibility="private"/>
    </packagedElement>
    <packagedElement xmi:type="UML:Class" xmi:id="pr" name="Print Request"/>
    <packagedElement xmi:type="UML:Class" xmi:id="bl" name="Brightlime">
      <ownedOperation xmi:id="rcr" name="receive check request" visibility="private"/>
      <ownedOperation xmi:id="cpd" name="check prospect database" visibility="private"/>
      <ownedOperation xmi:id="sr" name="send response" visibility="private"/>
      <ownedOperation xmi:id="rnpr" name="receive new prospect request" visibility="private"/>
      <ownedOperation xmi:id="utpd" name="upload to prospect database" visibility="private"/>
      <ownedOperation xmi:id="rur" name="receive update request" visibility="private"/>
      <ownedOperation xmi:id="upd" name="update prospect database" visibility="private"/>
    </packagedElement>
    <packagedElement xmi:type="UML:Class" xmi:id="pd" name="Prospect Database"/>
    <packagedElement xmi:type="UML:Association" xmi:id="_O0mlSji8Ed-bDfRognGdXA" name="Association1" memberEnd="_O0mlSZi8Ed-
bDfRognGdXA _O0mlTji8Ed-bDfRognGdXA">
      <ownedEnd xmi:id="_O0mlSZi8Ed-bDfRognGdXA" name=" " type="s" association="_O0mlSji8Ed-bDfRognGdXA">
        <upperValue xmi:type="UML:LiteralUnlimitedNatural" xmi:id="_O0mlSpi8Ed-bDfRognGdXA" value="1"/>
        <lowerValue xmi:type="UML:LiteralInteger" xmi:id="_O0mlS5i8Ed-bDfRognGdXA" value="1"/>
      </ownedEnd>
      <ownedEnd xmi:id="_O0mlTji8Ed-bDfRognGdXA" name="target" type="p" aggregation="shared" association="_O0mlSji8Ed-
bDfRognGdXA">
        <upperValue xmi:type="UML:LiteralUnlimitedNatural" xmi:id="_O0mlTZi8Ed-bDfRognGdXA" value="1"/>
        <lowerValue xmi:type="UML:LiteralInteger" xmi:id="_O0mlTpi8Ed-bDfRognGdXA" value="1"/>
      </ownedEnd>
    </packagedElement>
    <packagedElement xmi:type="UML:Association" xmi:id="_Pa5wSji8Ed-bDfRognGdXA" name="Association2" memberEnd="_Pa5wSZi8Ed-
bDfRognGdXA _Pa5wTji8Ed-bDfRognGdXA">
      <ownedEnd xmi:id="_Pa5wSZi8Ed-bDfRognGdXA" name="source" type="s" association="_Pa5wSji8Ed-bDfRognGdXA">
        <upperValue xmi:type="UML:LiteralUnlimitedNatural" xmi:id="_Pa5wSpi8Ed-bDfRognGdXA" value="1"/>
        <lowerValue xmi:type="UML:LiteralInteger" xmi:id="_Pa5wS5i8Ed-bDfRognGdXA" value="1"/>
      </ownedEnd>

```

```

    <ownedEnd xmi:id="_Pa5wTJi8Ed-bDfRognGdXA" name="target" type="pr" aggregation="shared" association="_Pa5wSji8Ed-
bDfRognGdXA">
      <upperValue xmi:type="UML:LiteralUnlimitedNatural" xmi:id="_Pa5wTZi8Ed-bDfRognGdXA" value="1"/>
      <lowerValue xmi:type="UML:LiteralInteger" xmi:id="_Pa5wTpi8Ed-bDfRognGdXA" value="1"/>
    </ownedEnd>
  </packagedElement>
  <packagedElement xmi:type="UML:Association" xmi:id="_Po2Fp5i8Ed-bDfRognGdXA" name="Association3" visibility="private"
memberEnd="_Po2FqJi8Ed-bDfRognGdXA _Po2Fq5i8Ed-bDfRognGdXA">
    <ownedEnd xmi:id="_Po2FqJi8Ed-bDfRognGdXA" name=" " visibility="private" type="s" isUnique="false" association="_Po2Fp5i8Ed-
bDfRognGdXA">
      <upperValue xmi:type="UML:LiteralUnlimitedNatural" xmi:id="_Po2FqZi8Ed-bDfRognGdXA" value=""/>
      <lowerValue xmi:type="UML:LiteralInteger" xmi:id="_Po2Fqpi8Ed-bDfRognGdXA" value="1"/>
    </ownedEnd>
    <ownedEnd xmi:id="_Po2Fq5i8Ed-bDfRognGdXA" name="uploads" visibility="private" type="bl" isUnique="false" aggregation="shared"
association="_Po2Fp5i8Ed-bDfRognGdXA">
      <upperValue xmi:type="UML:LiteralUnlimitedNatural" xmi:id="_Po2FrJi8Ed-bDfRognGdXA" value=""/>
      <lowerValue xmi:type="UML:LiteralInteger" xmi:id="_Po2FrZi8Ed-bDfRognGdXA" value="1"/>
    </ownedEnd>
  </packagedElement>
  <packagedElement xmi:type="UML:Association" xmi:id="_Q4-QyJi8Ed-bDfRognGdXA" name="Association4" visibility="private"
memberEnd="_Q4-QyZi8Ed-bDfRognGdXA _Q4-QzJi8Ed-bDfRognGdXA">
    <ownedEnd xmi:id="_Q4-QyZi8Ed-bDfRognGdXA" name=" " visibility="private" type="s" isUnique="false" association="_Q4-QyJi8Ed-
bDfRognGdXA">
      <upperValue xmi:type="UML:LiteralUnlimitedNatural" xmi:id="_Q4-Qypi8Ed-bDfRognGdXA" value=""/>
      <lowerValue xmi:type="UML:LiteralInteger" xmi:id="_Q4-Qy5i8Ed-bDfRognGdXA" value="1"/>
    </ownedEnd>
    <ownedEnd xmi:id="_Q4-QzJi8Ed-bDfRognGdXA" name="responds" visibility="private" type="bl" isUnique="false" aggregation="shared"
association="_Q4-QyJi8Ed-bDfRognGdXA">
      <upperValue xmi:type="UML:LiteralUnlimitedNatural" xmi:id="_Q4-QzZi8Ed-bDfRognGdXA" value=""/>
      <lowerValue xmi:type="UML:LiteralInteger" xmi:id="_Q4-Qzpi8Ed-bDfRognGdXA" value="1"/>
    </ownedEnd>
  </packagedElement>
  <packagedElement xmi:type="UML:Association" xmi:id="_REiAiJi8Ed-bDfRognGdXA" name="Association5" visibility="private"
memberEnd="_REiAiZi8Ed-bDfRognGdXA _REiAjJi8Ed-bDfRognGdXA">
    <ownedEnd xmi:id="_REiAiZi8Ed-bDfRognGdXA" name=" " visibility="private" type="s" isUnique="false" association="_REiAiJi8Ed-
bDfRognGdXA">
      <upperValue xmi:type="UML:LiteralUnlimitedNatural" xmi:id="_REiAipi8Ed-bDfRognGdXA" value=""/>
      <lowerValue xmi:type="UML:LiteralInteger" xmi:id="_REiAi5i8Ed-bDfRognGdXA" value="1"/>
    </ownedEnd>
    <ownedEnd xmi:id="_REiAjJi8Ed-bDfRognGdXA" name="updates" visibility="private" type="bl" isUnique="false" aggregation="shared"
association="_REiAiJi8Ed-bDfRognGdXA">
      <upperValue xmi:type="UML:LiteralUnlimitedNatural" xmi:id="_REiAjZi8Ed-bDfRognGdXA" value=""/>
      <lowerValue xmi:type="UML:LiteralInteger" xmi:id="_REiAjpi8Ed-bDfRognGdXA" value="1"/>
    </ownedEnd>
  </packagedElement>
  <packagedElement xmi:type="UML:Association" xmi:id="_RNOCeJi8Ed-bDfRognGdXA" name="Association6" visibility="private"
memberEnd="_RNOCeZi8Ed-bDfRognGdXA _RNOCFji8Ed-bDfRognGdXA">
    <ownedEnd xmi:id="_RNOCeZi8Ed-bDfRognGdXA" name=" " visibility="private" type="s" isUnique="false" association="_RNOCeJi8Ed-
bDfRognGdXA">
      <upperValue xmi:type="UML:LiteralUnlimitedNatural" xmi:id="_RNOCEpi8Ed-bDfRognGdXA" value=""/>
      <lowerValue xmi:type="UML:LiteralInteger" xmi:id="_RNOCe5i8Ed-bDfRognGdXA" value="1"/>
    </ownedEnd>

```

```

    <ownedEnd xmi:id="_RNOcfJi8Ed-bDfRognGdXA" name="checks" visibility="private" type="bl" isUnique="false" aggregation="shared"
association="_RNOceJi8Ed-bDfRognGdXA">
    <upperValue xmi:type="UML:LiteralUnlimitedNatural" xmi:id="_RNOcfZi8Ed-bDfRognGdXA" value=""/>
    <lowerValue xmi:type="UML:LiteralInteger" xmi:id="_RNOcfpi8Ed-bDfRognGdXA" value="1"/>
    </ownedEnd>
</packagedElement>
<packagedElement xmi:type="UML:Association" xmi:id="_Rk0DGJi8Ed-bDfRognGdXA" name="Association7" visibility="private"
memberEnd="_Rk0DGZi8Ed-bDfRognGdXA _Rk0DHJi8Ed-bDfRognGdXA">
    <ownedEnd xmi:id="_Rk0DGGZi8Ed-bDfRognGdXA" name=" " visibility="private" type="bl" isUnique="false" association="_Rk0DGJi8Ed-
bDfRognGdXA">
    <upperValue xmi:type="UML:LiteralUnlimitedNatural" xmi:id="_Rk0DGPi8Ed-bDfRognGdXA" value=""/>
    <lowerValue xmi:type="UML:LiteralInteger" xmi:id="_Rk0DGG5i8Ed-bDfRognGdXA" value="1"/>
    </ownedEnd>
    <ownedEnd xmi:id="_Rk0DHJi8Ed-bDfRognGdXA" name="owns" visibility="private" type="pd" isUnique="false" aggregation="shared"
association="_Rk0DGGJi8Ed-bDfRognGdXA">
    <upperValue xmi:type="UML:LiteralUnlimitedNatural" xmi:id="_Rk0DHZi8Ed-bDfRognGdXA" value="1"/>
    <lowerValue xmi:type="UML:LiteralInteger" xmi:id="_Rk0DHpi8Ed-bDfRognGdXA" value="1"/>
    </ownedEnd>
</packagedElement>
<packagedElement xmi:type="UML:DataType" xmi:id="_5DS_oJi8Ed-bDfRognGdXA" name="DataType1"/>
<packagedElement xmi:type="UML:Association" xmi:id="_Mwh_R5i9Ed-bDfRognGdXA" name="Association8" visibility="private"
memberEnd="_Mwh_SJi9Ed-bDfRognGdXA _Mwh_S5i9Ed-bDfRognGdXA">
    <ownedEnd xmi:id="_Mwh_SJi9Ed-bDfRognGdXA" name=" " visibility="private" type="s" isUnique="false" association="_Mwh_R5i9Ed-
bDfRognGdXA">
    <upperValue xmi:type="UML:LiteralUnlimitedNatural" xmi:id="_Mwh_SZi9Ed-bDfRognGdXA" value=""/>
    <lowerValue xmi:type="UML:LiteralInteger" xmi:id="_Mwh_Spi9Ed-bDfRognGdXA" value="1"/>
    </ownedEnd>
    <ownedEnd xmi:id="_Mwh_S5i9Ed-bDfRognGdXA" name="creates" visibility="private" type="pr" isUnique="false" aggregation="shared"
association="_Mwh_R5i9Ed-bDfRognGdXA">
    <upperValue xmi:type="UML:LiteralUnlimitedNatural" xmi:id="_Mwh_TJi9Ed-bDfRognGdXA" value=""/>
    <lowerValue xmi:type="UML:LiteralInteger" xmi:id="_Mwh_TZi9Ed-bDfRognGdXA" value="1"/>
    </ownedEnd>
</packagedElement>
<packagedElement xmi:type="UML:Association" xmi:id="_OY6SWZi9Ed-bDfRognGdXA" name="Association9"
memberEnd="_OY6SWpi9Ed-bDfRognGdXA _OY6SXZi9Ed-bDfRognGdXA">
    <ownedEnd xmi:id="_OY6SWpi9Ed-bDfRognGdXA" name="source" type="p" association="_OY6SWZi9Ed-bDfRognGdXA">
    <upperValue xmi:type="UML:LiteralUnlimitedNatural" xmi:id="_OY6SW5i9Ed-bDfRognGdXA" value="1"/>
    <lowerValue xmi:type="UML:LiteralInteger" xmi:id="_OY6SXJi9Ed-bDfRognGdXA" value="1"/>
    </ownedEnd>
    <ownedEnd xmi:id="_OY6SXZi9Ed-bDfRognGdXA" name="target" type="s" aggregation="shared" association="_OY6SWZi9Ed-
bDfRognGdXA">
    <upperValue xmi:type="UML:LiteralUnlimitedNatural" xmi:id="_OY6SXpi9Ed-bDfRognGdXA" value="1"/>
    <lowerValue xmi:type="UML:LiteralInteger" xmi:id="_OY6SX5i9Ed-bDfRognGdXA" value="1"/>
    </ownedEnd>
</packagedElement>
<packagedElement xmi:type="UML:Association" xmi:id="_R91xqJi9Ed-bDfRognGdXA" name="Association10" visibility="private"
memberEnd="_R91xqZi9Ed-bDfRognGdXA _R91xrJi9Ed-bDfRognGdXA">
    <ownedEnd xmi:id="_R91xqZi9Ed-bDfRognGdXA" name=" " type="s" association="_R91xqJi9Ed-bDfRognGdXA">
    <upperValue xmi:type="UML:LiteralUnlimitedNatural" xmi:id="_R91xqpi9Ed-bDfRognGdXA" value=""/>
    <lowerValue xmi:type="UML:LiteralInteger" xmi:id="_R91xq5i9Ed-bDfRognGdXA" value="1"/>
    </ownedEnd>
    <ownedEnd xmi:id="_R91xrJi9Ed-bDfRognGdXA" name="sends" visibility="private" type="p" isUnique="false" aggregation="shared"
association="_R91xqJi9Ed-bDfRognGdXA">

```

```

    <upperValue xmi:type="UML:LiteralUnlimitedNatural" xmi:id="_R91xrZi9Ed-bDfRognGdXA" value="1"/>
    <lowerValue xmi:type="UML:LiteralInteger" xmi:id="_R91xrpi9Ed-bDfRognGdXA" value="1"/>
  </ownedEnd>
</packagedElement>
<packagedElement xmi:type="UML:Association" xmi:id="_BAV5Sji-Ed-bDfRognGdXA" name="Association11" memberEnd="_BAV5SZi-Ed-bDfRognGdXA _BAV5Tji-Ed-bDfRognGdXA">
  <ownedEnd xmi:id="_BAV5SZi-Ed-bDfRognGdXA" name="source" type="s" association="_BAV5Sji-Ed-bDfRognGdXA">
    <upperValue xmi:type="UML:LiteralUnlimitedNatural" xmi:id="_BAV5Spi-Ed-bDfRognGdXA" value="1"/>
    <lowerValue xmi:type="UML:LiteralInteger" xmi:id="_BAV5S5i-Ed-bDfRognGdXA" value="1"/>
  </ownedEnd>
  <ownedEnd xmi:id="_BAV5Tji-Ed-bDfRognGdXA" name="target" type="p" aggregation="shared" association="_BAV5Sji-Ed-bDfRognGdXA">
    <upperValue xmi:type="UML:LiteralUnlimitedNatural" xmi:id="_BAV5TZi-Ed-bDfRognGdXA" value="1"/>
    <lowerValue xmi:type="UML:LiteralInteger" xmi:id="_BAV5Tpi-Ed-bDfRognGdXA" value="1"/>
  </ownedEnd>
</packagedElement>
<packagedElement xmi:type="UML:Association" xmi:id="_EOJ5Gzi-Ed-bDfRognGdXA" name="Association12" visibility="private" memberEnd="_EOJ5Gpi-Ed-bDfRognGdXA _EOJ5HZi-Ed-bDfRognGdXA">
  <ownedEnd xmi:id="_EOJ5Gpi-Ed-bDfRognGdXA" name=" " visibility="private" type="s" isUnique="false" association="_EOJ5Gzi-Ed-bDfRognGdXA">
    <upperValue xmi:type="UML:LiteralUnlimitedNatural" xmi:id="_EOJ5G5i-Ed-bDfRognGdXA" value=""/>
    <lowerValue xmi:type="UML:LiteralInteger" xmi:id="_EOJ5Hji-Ed-bDfRognGdXA" value="1"/>
  </ownedEnd>
  <ownedEnd xmi:id="_EOJ5HZi-Ed-bDfRognGdXA" name="sends" visibility="private" type="p" isUnique="false" aggregation="shared" association="_EOJ5Gzi-Ed-bDfRognGdXA">
    <upperValue xmi:type="UML:LiteralUnlimitedNatural" xmi:id="_EOJ5Hpi-Ed-bDfRognGdXA" value="1"/>
    <lowerValue xmi:type="UML:LiteralInteger" xmi:id="_EOJ5H5i-Ed-bDfRognGdXA" value="1"/>
  </ownedEnd>
</packagedElement>
</packagedElement>
</UML:Model>

```

Glossary

.NET	Software technology platform.
3GL	Third Generation Language, referring collectively to high-level structured programming languages.
Abstraction	Level of perception relative to content.
Action Semantic Language	The language of UML Action Semantics.
Activity	Notational element describing a behaviour or task.
Activity Diagram	UML diagram; used to model system behaviour.
Agile Development	Software development method focussed on being adaptable and dynamic. <i>See Agile Manifesto.</i>
Agile Manifesto	Doctrine of Agile Development. <i>See Agile Development.</i>
Aggregation	Object oriented association type describing a close relationship between classes. <i>See Object Oriented, Association and Class.</i>
Alignment	IT systems designed and maintained in line with business strategy. <i>See Strategy.</i>
AMDD	Agile Model Driven Development, MDD following the Agile Manifesto. <i>See Agile Manifesto and Model Driven Development.</i>
Analysis	Investigative stage of software development. <i>See Systems of Prime Concern.</i>
API	<i>See Application Programming Interface.</i>
Application Domain	The area in which the solution system exists. <i>See Solution System.</i>
Application Programming Interface	Interface used to support the construction of applications.
Assembly Language	Programming language abstraction on machine code and binary. <i>See Machine Code.</i>
Assertion	True or false statement.
ASSIST	A Shrewd Sketch Interpretation and Simulation Tool, sketch recognition technology. <i>See Sketch Recognition.</i>
Association	Object oriented structural mechanism describing relationships between classes. <i>See Object Oriented and Class.</i>
ATL	ATLAS Transformation Language for model transformation.
Attributes	<i>See Properties.</i>
B-SCP	Business Strategy Context Process, framework for aligning business with IT.
Balanced Scorecard	Tool for managing organisational strategy based on performance

	indication.
BBPF	Basic Business Process Flow, used to formalise business requirements in the BPDLC. <i>See BPDLC.</i>
Behavioural Model	Pre-CIM model to highlight specific functional requirements and the behaviour of the involved process. <i>See pre-CIM.</i>
Bigraph	<i>See Bipartite Graph.</i>
Binary	Numbering system used to connect to hardware.
Bipartite Graph	Graph in which vertices can be divided into two.
BM	<i>See Behavioural Model.</i>
BPD	<i>See Business Process Diagram.</i>
BPDLC	Business Process Developing Life Cycle, approach to formalise business requirements in terms of BBPF. <i>See BBPF.</i>
BPDM	<i>See Business Process Definition Metamodel.</i>
BPEL	<i>See Business Process Execution Language.</i>
BPEL4WS	Business Process Execution Language for Web Services.
BPM	<i>See Business Process Management.</i>
BPMN	<i>See Business Process Modelling Notation.</i>
BPMS	<i>See Business Process Management System.</i>
BPR	Business Processes and Requirements, unit on the Software Systems framework at Bournemouth University.
BSC	<i>See Balanced Scorecard.</i>
Business Analyst	The PD expert. <i>See Business User and Problem Domain.</i>
Business Process Definition Metamodel	Metamodel used to define business processes.
Business Process Diagram	Artefact of BPM.
Business Process Execution Language	Programming language used to define interactive business processes.
Business Process Management	Business management field whereby processes are the central focus for efficiency gains.
Business Process Management System	Software system that employs tools and techniques used to manage business processes.
Business Process Model	Visualisation of the Business Process. <i>See Business Process Management.</i>
Business Process Modelling Notation	Visualisation technique to define business process workflows.
Business Rule	Statement used to define the operating condition of a business process or the requirements for a software system.
Business User	<i>See Business Analyst.</i>

C#	Object oriented programming language.
CASE	Computer Aided Software Engineering, automated tool sets used within the software process. <i>See Software Process.</i>
Case Management	The control of process instances.
CIM	<i>See Computation Independent Model.</i>
Class	Element of the UML used to represent things that exist within the context of the system. <i>See Class Diagram.</i>
Class Diagram	UML diagram; Static view of system design.
Code	Instruction set within a programming language or program.
Collaboration	A relationship between two or more entities.
Collaboration Diagram	UML diagram; used to model communication.
Common Warehouse Metamodel	Standard for modelling metadata.
Communicating Sequential Processes	Formal language for defining concurrent processes.
Compiler	Software implementation typically used prior to execution to convert from high-level languages to low level machine code. <i>See 3GL, Java, Machine Code and Interpreter.</i>
Composition	Object oriented association type describing a relationship between classes where one is composed of another. <i>See Object Oriented, Association and Class.</i>
Computation Independent Model	Analysis stage of the MDA
Concurrency	Describes two or more activities occurring at the same time.
Connection	Used to illustrate a relationship between two or more entities.
Control Class	Object oriented functional communication management class stereotype. <i>See Object Oriented, Stereotype and Class.</i>
COTS	Common-Off-The-Shelf, software that is typically widely available.
CP Style Rules	A direction offered for the guidance on the format and construction of Use Case descriptions.
CSP	<i>See Communicating Sequential Processes.</i>
CTO	Chief Technical Officer, organisational position.
CWM	<i>See Common Warehouse Metamodel.</i>
Data Flow Diagram	Functional visualisation technique for modelling information systems.
DDL	Data Definition Language, part of SQL used to describe data structures. <i>See SQL.</i>
DDM	<i>See Domain Description Model.</i>
Dependency	Strong association between elements.

Deployment Descriptor	File used to describe component content and configuration.
Design	Inventive stage of software development. <i>See Systems of Prime Concern.</i>
DFD	<i>See Data Flow Diagram.</i>
Domain Description Model	Pre-CIM model to define the PD context as it is. <i>See pre-CIM.</i>
Domain Specific Language	Programming language specific to a particular domain. <i>See Domain Specific Modelling.</i>
Domain Specific Modelling	Modelling in terms of DSL. <i>See Domain Specific Language.</i>
DSL	See Domain Specific Language.
DSM	See Domain Specific Modelling.
Eclipse	<i>See Eclipse Modelling Framework.</i>
Eclipse Modelling Framework	MDA development framework.
EJB	Enterprise JavaBeans, Java API for building enterprise systems. <i>See Java and Application Programming Interface.</i>
Elaborationist	Developer that uses MDA modelling and code as a template to elaborate on.
Elicitation	The act of obtaining correct and complete requirements from stakeholders in RE.
Enterprise	Organisation-wide integration of software systems.
Enterprise Modelling	Modelling in terms of the enterprise and related systems and resources as a whole.
Entity	Article with perceived existence and relevance to the system context.
Entity Class	Object oriented data access and retrieval class stereotype. <i>See Object Oriented, Stereotype and Class.</i>
Environment RAD	Specification phase of the xMDA method directed at resolving environmental issues. <i>See eXtended Model Driven Architecture.</i>
ERD	Entity Relationship Diagram, notation for data modelling.
Event Flow	Order of events in a Use Case description.
Experimental Learning Cycle	Research framework to examine the relationship between theory and practice.
eXtended Model Driven Architecture	An extension to the MDA, proposed to cater for Requirements Engineering.
eXtensible Mark-up Language	Standard used to encode electronic artefacts for communication within software applications. <i>See XML Metadata Interchange.</i>
eXtensible Stylesheet Language Transformation	Standard used to transform documents in XML. <i>See eXtensible Mark-up Language.</i>

eXtreme Programming	Software development method focussed on the delivery of code.
FBCM	Fact Based Collaboration Modelling, methodology for IT strategic alignment.
Flowchart	Generic visualisation technique to define processes.
Forward Engineer	The re-construction of a model from originating documentation. <i>See Reverse Engineer.</i>
Functional Modelling	Modelling in terms of function.
Generator	System used to deliver code from design models.
Goal Modelling	Modelling in terms of strategic objectives.
GORE	Goal-Oriented Requirements Engineering, technique for aligning IT with business goals.
GUI	Graphical User Interface, computer component used to interface visually with the end user.
GUIDE	Goal, Use, Investment, Deliverables, Experience/Environment, technique for modelling goals using the DFD. <i>See Data Flow Diagram.</i>
Hanging Thread	String of activity within a RAD role which is activated on event. <i>See Role Activity Diagram.</i>
Hardware	Physical computer components designed to respond to software instructions.
HIM	<i>See Human Interaction Management.</i>
HIMS	<i>See Human Interaction Management System.</i>
HTTP	Hypertext Transfer Protocol, internet protocol for data transfer.
Human Interaction Management	Philosophy for accommodating humanistic processes.
Human Interaction Management System	A software system used in HIM.
Human-Driven Process	<i>See Humanistic Process.</i>
Humanistic Process	A process that specifically involves human collaborative behaviours.
IBM	Software technology provider.
ICAM	Integrated Computer Aided Manufacturing, U.S. Air Force program resulting in the IDEF family of languages. <i>See Integration DEFinition.</i>
ICOM	Inputs, Controls, Outputs and Mechanisms, IDEF concepts. <i>See Integration DEFinition.</i>
IDEF	<i>See Integration DEFinition.</i>
IDL	Interface Definition Language, for describing component

	interfaces. <i>See Interface.</i>
Implementation	Execution stage of software development.
Inclusive Techniques	Methods of elicitation particular to Agile Development. <i>See Agile Development.</i>
Information System	Software system that is focussed on the delivery of data and information, rather than the management of business and process.
Information Technology	Field of study relating to hardware, software and their integration.
Inheritance	Object oriented structural mechanism describing associations between parent super-class types and child sub-class types. <i>See Object Oriented, Association and Class.</i>
Initial Requirements Model	Pre-CIM model to define any requirements (both functional and non-functional) to be imposed by the new system. <i>See pre-CIM.</i>
Instruction Set	Basic commands understood by and hardwired within the CPU.
Integration DEFINITION	Family of information and process modelling standards. Also known as ICAM DEFINITION. <i>See ICAM.</i>
Interaction	Relationship between two or more entities.
Interface	Java programming element. <i>See Application Programming Interface.</i>
Interface Class	Object oriented connection class stereotype. <i>See Object Oriented, Stereotype and Class.</i>
Interoperability	Ability to communicate and exchange data between components.
Interpreter	Software implementation typically used on execution to convert from high-level languages to low level machine code. <i>See 3GL, Java, Machine Code and Compiler.</i>
IPSE	Integrated Project Support Environment, architecture used to support and define software development and the software process. <i>See Initial Requirements Model.</i>
IRM	
ISO	International Organisation for Standardisation.
ISO12207	ISO standard for software lifecycle processes. <i>See ISO.</i>
IT	<i>See Information Technology.</i>
Iteration	The process of repeating development activities with the objective of refining solutions.
J2EE	Software technology platform.
Java	Object oriented programming language.
Lean Six Sigma	Method for managing organisational strategy based on value chain analysis.
Legacy System	Systems which currently exist within an organisation in context of

	new implementations.
Loop	Element of language within which instructions or activities are processed until specific conditions are met.
LOTOS	Language of Temporal Ordering Specification, formal specification language.
LSS	<i>See Lean Six Sigma.</i>
Machine Code	Language based on binary to interface with the CPU. <i>See Binary and Instruction Set.</i>
Machine RAD	Specification phase of the xMDA method directed at resolving machine issues. <i>See eXtended Model Driven Architecture.</i>
Mapping	Each element from one model is mapped into an associated element in another model.
Markings	Identification used in tracing elements mapped or transformed from one model to the next.
Marks	<i>See Markings.</i>
MDA	<i>See Model Driven Architecture.</i>
MDABIZ	Business Support for MDA, international workshop.
MDD	<i>See Model Driven Development.</i>
MDE	<i>See Model Driven Engineering.</i>
MDSEE	<i>See Model Driven Software Engineering Environment.</i>
Mechanistic Process	A process that specifically involves computerised sequential behaviours.
MEMO	<i>See Multi-perspective Enterprise Modelling.</i>
Meta Object Facility	OMG Standard for defining metamodels. <i>See Meta-metamodel.</i>
Meta-metamodel	Language to define metamodels.
Metadata	Data that describes other data.
Metamodel	Model that describes other models.
Microsoft	Software technology provider.
MIT	Massachusetts Institute of Technology, educational establishment.
Model	An abstraction on code.
Model Driven Architecture	Conceptual framework for software development.
Model Driven Development	Software development field whereby models are the central artefact. <i>See Model Driven Engineering.</i>
Model Driven Engineering	Software development field whereby models are the central artefact. <i>See Model Driven Development.</i>
Model Driven Software Engineering Environment	MDA support environment with model and metamodel access; model transformation; simulation; process; and project definition

	for developers. <i>See MDA.</i>
Model Merging	The joining of two or more separate models to make a single model containing all originating model detail. <i>See model.</i>
Modeller	Developer involved in the modelling process.
MOF	<i>See Meta Object Facility.</i>
Multi-perspective Enterprise	Conceptual framework for Enterprise Modelling. <i>See Enterprise</i>
Modelling	<i>Modelling.</i>
Multiplicity	Object oriented mechanism to describe the numbering of associated class instances. <i>See Object Oriented, Association and Class.</i>
Node	Connecting graphical element in graphing.
Non-deterministic Process	Processes that involve assertions to which outcomes are open to indistinguishable possibilities.
Notation	Graphical representation for process definition.
Object	Class instance. <i>See Class and Object Oriented.</i>
Object Constraint Language	Language standard used by web service developers for writing business rules.
Object Management Group	Standards consortium with major contributors such as Microsoft and IBM. Creators and maintainers of standards such as the MDA and UML.
Object Oriented	Methodology concerned with the construction of systems build on the concept of objects.
OCL	<i>See Object Constraint Language.</i>
OMG	<i>See Object Management Group.</i>
Operational QVT	QVT transformation language for Eclipse. <i>See Eclipse Modelling Framework.</i>
Operations	Object oriented mechanism describing the configurable processing of classes. <i>See Object Oriented and Class.</i>
Oracle	Software technology provider.
ORMSC	Object and Reference Model Subcommittee, part of the OMG concerned with developing the MDA Guide. <i>See Object Management Group and Model Driven Architecture.</i>
Parallel Process	A process that runs in parallel to another.
Pattern	Recognisable operations that form the basis or template structures for use in the development process.
PD	<i>See Problem Domain.</i>
PERL	Textual programming language.

Persistence	Elements retained in context of the system.
Petri Net	Language used to describe mathematical modelling.
PIM	<i>See Platform Independent Model.</i>
PIM Prototyping Tool	Part of the VIDE toolset containing a Java transformation engine to map VCLL to UML. <i>See Transformation Engine.</i>
Platform	Operational environment for software systems.
Platform Independent Model	Design stage of the MDA.
Platform Specific Model	Implementation stage of the MDA.
PML	<i>See Process Modelling Language.</i>
Port	Accessibility conduit.
Portability	Systems that are compatible, or made compatible for multiple platforms.
POSD	<i>See Process Oriented Systems Design.</i>
PPT	<i>See PIM Prototyping Tool.</i>
pre-CIM	Activities upstream of the CIM conducted prior to CIM construction.
Private Information Space	Of RADs, information is controlled and maintained in and between roles.
Problem Domain	The area in which a defined problem is identified.
Process	Collection of business activities to form part of or a complete business conceptualisation.
Process Architecture	An environment defining the creation and maintenance of business processes.
Process Modelling Language	Formal language for business process execution (based on Requirements Modelling Language). <i>See Requirements Modelling Language.</i>
Process Oriented Systems Design	Structural mechanism for visually describing a system.
Process Programming	Formalisation of processes.
Process Trinity	Represented process types; Case, Management and Strategy. <i>See RIVA.</i>
Profile	Element of the UML accounting for notational extension.
Properties	Configurable parameters of objects or classes.
Prototype	A preliminary artefact of the development process used as a sample for demonstration purposes and elicitation.
Pseudo Code	Language used as an abstraction on high-level programming languages.
PSM	<i>See Platform Specific Model.</i>

Query / View / Transformation	OMG Standard for defining transformations between MOF metamodels. <i>See MOF.</i>
QVT	<i>See Query / View / Transformation.</i>
QVT-R	<i>See QVT-Relations.</i>
QVT-Relations	Declarative language of the QVT for defining relationships between MOF metamodels. <i>See Query / View / Transformation.</i>
RAD	<i>See Role Activity Diagram.</i>
Rational Rose	Software development toolkit.
Rational Unified Process	Software development framework.
RE	<i>See Requirements Engineering.</i>
REBNITA	Requirements Engineering for Business Need and IT Alignment, international workshop.
Relation	<i>See QVT-Relations.</i>
Remote Interface	Java component for declaring methods. <i>See Java and Interface.</i>
Requirements	The desired effects that the solution system imposes on the PD. <i>See Problem Domain.</i>
Requirements Engineer	Member of the software development team that interfaces with stakeholders in eliciting the requirements for software systems.
Requirements Engineering	The field of engineering within software systems whereby user needs and their realisation in solution systems are the central focus.
Requirements Modelling Language	Formal language for requirements modelling (basis of PML). <i>See Process Modelling Language.</i>
Reusability	Software component able to be implemented within or between a multitude of projects, with or without modification.
Reverse Engineer	The de-construction of a model into originating documentation. <i>See Forward Engineer.</i>
RIVA	Business process architecture. <i>See Role Activity Diagram.</i>
RM-ODP	Reference Model of Open Distributed Processing, standardisation framework for open distributed processing.
RML	<i>See Requirements Modelling Language.</i>
Role	Central notion in a RAD defining a participant's responsibility and their interactions with other participants and information. <i>See Role Activity Diagram.</i>
Role Activity Diagram	Visualisation technique to define business process workflows.
Role Utility Diagram	Notation supporting a static view of the RAD. <i>See Role Activity Diagram.</i>
RUD	<i>See Role Utility Diagram.</i>

SADT	<i>See Structured Analysis and Design Technique.</i>
SAP	Software technology provider.
Schema	Outline of models or languages that define them.
SEAM	Systemic Enterprise Architecture Method, for aligning business with IT in terms of organisation and function.
Semantic	The meaning of language.
Service Oriented Architecture	Software development method giving focus to the offering and discovery of functional services.
Shared RAD	Specification phase of the xMDA method directed at resolving issues shared between the environment and machine. <i>See eXtended Model Driven Architecture.</i>
SimpleRAD Metamodel	MOF metamodel describing a simplified version of the RAD. <i>See Metamodel and RAD.</i>
SimpleUML Metamodel	MOF metamodel describing a simplified version of the UML. <i>See Metamodel and UML.</i>
Sketch Recognition	Technology used to recognise user sketches and create XML based representation. <i>See eXtensible Mark-up Language.</i>
SOA	<i>See Software Oriented Architecture.</i>
SOAP	Simple Object Access Protocol for XML data transfer. <i>See eXtensible Mark-up Language.</i>
Software	Computer code designed for a particular purpose or application.
Software Development	The art involving the definition of software systems, from inception to implementation and maintenance.
Software Engineer	Design and implements the solution system.
Software Engineering	The field of engineering within software systems whereby the design and implementation of solution systems are the central focus.
Software Factory	The architecture of complete SPLs and software artefacts created by them. <i>See Software Product Line.</i>
Software Process	The architecture in which software implementations are developed.
Software Product Line	Methodology for industrialising the software development process in a software factory. <i>See Software Factory.</i>
Solution System	A software system that accounts for requirements. <i>See Application Domain.</i>
Source Model	Model from which a target model is transformed.
Specification	The interface between the problem and application domain. <i>See Systems of Prime Concern.</i>

SPL	<i>See Software Product Line.</i>
SQL	Structured Query Language, used for querying databases.
SQM	Software Quality Management, international conference.
Stakeholder	Any number of interested parties, including (but not limited to) business user, Systems Analyst, Software Engineer, project managers and investors.
Standards	Rules governing the development process.
Standish Group International, Inc.	IT market research organisation, focussing on project investments and value performance.
State	Condition that a process is in given context.
Static Field	Java programming element.
Stereotype	Classification of class types in object oriented technologies. <i>See Object Oriented and Class.</i>
Strategy	Action plan devised by business in consideration of environmental implications, business values and available resources to achieve business goals and needs. <i>See Alignment.</i>
STRIM	Systematic Technique for Role and Interaction Modelling. <i>See RIVA.</i>
Structured Analysis and Design Technique	Functional visualisation technique for modelling information systems.
Syntactic	The structural arrangement of text and models.
SystemRAD	Visual method for describing systems via the notion of a System participant in a RAD. <i>See Role Activity Diagram.</i>
Systems Analyst	Member of the development team concerned with the analyses of business requirements and the construction of systems specification.
Systems of Prime Concern	Collectively, the Problem Domain, Interface and Solution System; matching the development activities of Analysis, Specification and Design. <i>See Analysis, Specification and Design.</i>
Systems Thinking	School of thought that considers a system as comprised of both internal and external environments, and the relationships between internal and external systems focussing on the system as a whole.
Target Model	Model that is created from a source model.
Traceability	Mechanism whereby software information is able to be linked to the original requirement requiring that information, and vice versa.
Transformation	The process of converting a source model into a target model.
Transformation Engine	Software component used to execute a transformation.

Transformation Record	The documentation of any transformation and mapped elements.
Transformation Rules	Conventions defined to describe transformations. <i>See transformation.</i>
Translation	The process of transferring one model and meaning into another.
Translationist	Developer that uses MDA modelling and code as a finished article, therefore transformations must be complete in every respect.
Tree	Diagrammatic structure.
Tri-Step Analysis	Analysis technique described by the xMDA method to derive candidate design classes. <i>See xMDA.</i>
UCDML	Use Case Description Mark-up Language, XML language for Use Cases. <i>See Use Case and eXtensible Mark-up Language.</i>
UML	<i>See Unified Modelling Language.</i>
Unified Modelling Language	Generic modelling technique used in software systems development for defining systems design.
Use Case	UML diagram; processes defined by functionality.
Value Chain	Model of the complete business process used for analyses in the identification of efficiency gains.
VCLL	<i>See VIDE CIM Level Language.</i>
VIDE	<i>See Visualise All Model Driven Programming.</i>
VIDE CIM Level Language	Part of the VIDE toolset to create XML representations of BPMN models. <i>See eXtended Model Driven Architecture.</i>
Visualise All Model Driven Programming	European model-driven research initiative.
Web Service	Interoperable software service. <i>See Service Oriented Architecture.</i>
Web Service Definition Language	Language standard used by web service developers for writing web services. <i>See Web Service.</i>
Workflow	Sequential visualisation technique for describing process.
WS-Policy	Web Service-Policy, standard enabling XML policy definition.
WSDL	<i>See Web Service Definition Language.</i>
xMDA	<i>See eXtended Model Driven Architecture.</i>
XMI	<i>See XML Metadata Interchange.</i>
XML	<i>See eXtensible Mark-up Language.</i>
XML Metadata Interchange	OMG standard for exchanging metadata via XML. <i>See XML.</i>
XSLT	<i>See eXtensible Stylesheet Language Transformation.</i>
YAWL	Yet Another Workflow Language, pattern-based language.
Z	Formal specification language.
