The design and evaluation of a specification framework for user interface design

SIMON CROWLE

A thesis submitted in partial fulfilment of the requirements of Bournemouth University for the degree of Doctor of Philosophy

APRIL 2003

Page Numbering as Bound

Abstract

This thesis presents the design and evaluation of an interface specification meta-language (ISML) that has been developed to explicitly support metaphor abstractions in a model-based, user interface design framework. The application of metaphor to user interface design is widely accepted within the HCI community, yet despite this, there exists relatively little formal support for user interface design practitioners. With the increasing range and power of user interface technologies made widely available comes the opportunity for the design of sophisticated, new forms of interactive environments.

The inter-disciplinary nature of HCI offers many approaches to user interface design that include views on tasks, presentation and dialogue architectures and various domain models. Notations and tools that support these views vary equally, ranging from craft-based approaches through to computational or tool-based support and formal methods. Work in these areas depicts gradual cohesion of a number of these design views, but do not currently explicitly specify the application of metaphorical concepts in graphical user interface design.

Towards addressing this omission, ISML was developed based on (and extending) some existing model-based user interface design concepts. Abstractions of metaphor and other interface design views are captured in the ISML framework using the extensible mark-up language (XML). A six-month case study, developing the 'Urban Shout Cast' application is used to evaluate ISML. Two groups of four software engineers developed a networked, multi-user, virtual radio-broadcasting environment. A qualitative analysis examines both how each group developed metaphor designs within the ISML framework and also their perceptions of its utility and practicality. Subsequent analysis on the specification data from both groups reveals aspects of the project's design that ISML captured and those that were missed. Finally, the extent to which ISML can currently abstract the metaphors used in the case study is assessed through the development of a unified 'meta-object' model.

The results of the case study show that ISML is capable of expressing many of the features of each group's metaphor design, as well as highlighting important design considerations during development. Furthermore, it has been shown, in principle, how an underlying metaphor abstraction can be mapped to two different implementations. Evaluation of the case study also includes important design lessons: ISML metaphor models can be both very large and difficult to separate from other design views, some of which are either weakly expressed or unsupported. This suggests that the appropriate mappings between design abstractions cannot always be easily anticipated, and that understanding the use of model-based specifications in user interface design projects remains a challenge to the HCI community.

Contents

CD-ROM

Appendix F – The XML expression of ISML
Appendix G – A small ISML specification
Appendix H – Unified USC specification
Appendix I – Basic meta-object kernel
Appendix J – Sample XSLT transformation
Appendix K – USC Transcripts
Appendix L – USC model summaries
Appendix M – Group 1 USC Prototype
Appendix N – Group 4 USC Prototype
Appendix O – Early ISML grammar
Appendix P – Atlas.ti data files

List of figures

List of tables

Acknowledgements

I would like to thank Linda, Martin and Dan for their friendship, guidance and support. Additional thanks also go to the software engineering management undergraduate degree cohort, who so enthusiastically took part in the USC case study. Finally, I would like to thank my family and friends for all their encouragement, reassurance and love.

Author's declaration

The following publications are based on work presented in this thesis:

CROWLE, S. AND HOLE, L., 2001. Seeing the wood for the trees: A framework for the specification of metaphor in interface design. *In: Workshop on Integrating Multimedia, Metaphors and Multimodality , in PC-HCI 2001: Human Computer Interaction 2001, Patras, Greece,* Typorama Publishers, 19-24.

CROWLE, S. AND HOLE, L., 2003. An Interface Specification Meta-Language. *Accepted for publication in Proceedings of the 10th International Eurographics Workshop of Design, Specification and Verification of Interactive Systems '03, Funchal, Madeira Island, Portugal,* Springer, (in press).

CHAPTER 1 Introduction

The graphical user interface (GUI) introduced by the Xerox's Star system (Smith et al. (1982), strongly influences user interface design today. Unlike other user interfaces of the time, the Star GUI exploited new graphical technologies to present the user with an interactive environment that mimicked their world of work, coupled with novel input devices that allowed users to affect actions using gestures, rather than through commands issued via a keyboard. This system of interactive entities that assume some of the appearance and behaviour of familiar objects allows users to apply their knowledge of the real world to the interpretation and manipulation of the computer's state. The Xerox Star system is perhaps the most famous early example of this, employing what is frequently referred to as a *user interface metaphor* (Preece et al. (1994) that supports the direct manipulation interaction paradigm (Shneiderman (1983). Many reproductions of this 20 year old design can be found in modern personal computer systems today, including Microsoft's *Windows*, Apple's *Aqua* and the Linux window manager, *Gnome*.

A variety of interactive environments that employ novel metaphors to support specific task domains can be found in the literature, see Hole et al. (1998), Dieberger and Frank (1998), van Dantzich et al. (1999), Small (1996). Most, if not all, of the HCI research community is likely to be aware of the basic principle behind the application of metaphor to user interface design. Of the Xerox *Star* interface, Preece et al. (1994) explain:

> "The core aspect of the interface metaphor was to create electronic counterparts to the physical objects in an office...The effect is users will develop mental models of the system that are more like the metaphor rather than how the underlying system works".

Alty and Knott (1999) use Richards' nomenclature (Richards (1936) of 'tenor' and 'vehicle' to explain this same metaphor:

> "The real-world desktop acts as a vehicle in order to transform the tenor, in this case the operating system. Thus, a metaphor requires three concepts; the Tenor, the Vehicle and the transformation between them."

Quantitative and qualitative evaluations of the use of metaphor in design can also be found in Zajicek and Windsor (1995), Maglio and Matlock (1998),Golovchinsky and Chignell (1997) and Ark et al. (1998). Whilst these case-studies illustrate the potential application of new forms of interaction, very little work currently exists that formalises metaphor in user interface design. Presently, HCI research may turn to psychological theories of metaphor Lakoff and Johnson (1980), Lakoff (1992), Gentner et al. (2001), Gillan and Bias (1994) that provide an abstract account of the use of metaphor in design. A mathematical model describing the transference of the properties of the vehicle to the tenor can be found in Indurkhya

(1986) whilst Kuhn and Frank (1991) formally compare the properties of a real desktop to that found in a typical user interface.

With the advent of cheap and powerful interface technologies, a far wider range of interactions between the user and graphical environments can be represented at the user interface. Initial research with high performance 2D and 3D graphics fell within the remit of information visualisation and a substantial corpus of research can been found in the literature regarding its application (Card et al. (1999), (Spence (2001). This research area provides valuable insights into the application of advanced graphical technologies to the understanding of large or complex data sets, but has much less to say regarding the design and presentation of interactive environments that might support conventional application domains. Indeed, despite the widespread use of the desktop metaphor and the availability of high performance graphics technology, it is surprising to find relatively little guidance to support the development, specification and implementation of metaphors for modern GUIs (Alty et al. (2000). Work by Alty and Knott (1999) provides a high level model for applying the features of a metaphor to user interface design and a handful of guidelines are reported in the literature, see Lovgren (1994), Marcus (1994), Akoumianakis and Stephanidis (2000). To exacerbate the problem, others in the community argue against the use of metaphor altogether (Halasz and Moran (1982); (Nardi and Zarmer (1993). For interface designers and software engineers, the utility of these accounts of metaphor is limited since:

- The benefits and problems of applying metaphors to GUI design are not well understood

- Contemporary metaphor abstractions are not in a form easily accessible to support design

In the near future, it is likely that this problem will appear with increasing frequency as personal computer users demand increasingly sophisticated interactions with computing devices that are capable of delivering high fidelity, graphically complex interfaces. A number of design issues arise from the availability of these technologies, including:

- Choosing from a potentially large array of graphically and interactively 'rich' design solutions

- Implementing the design for an increasing number of hardware and software platforms

- Specifying the mapping between a user's task, the metaphorical environment, and the underlying functionality of the system.

Arguably, the gap between what is technically deliverable at the user interface and the principles, design abstractions and tools available to address such designs continues to widen. This thesis does not attempt to address all these problems. Instead, the work that follows seeks to develop support for the design and evaluation of an abstract metaphor model for user interface design.

1. Scope and limitations of research

A few words regarding the scope and ambition of this research is needed here. A number of tacit and theoretical accounts of metaphor and its application to user interface design have already been identified (see above). Despite this, no coherent, all-encompassing theory exists that maps a firm, psychological account of metaphor with a rigorous interface design methodology and expected usability outcomes. Such an account would be remarkably complex and is beyond our grasp, and indeed the scope of this thesis, at present. It is not the intention of this research to support a psychological account of metaphor. Instead, a 'proof of concept' mapping between a tenor and a vehicle (as expressed by Alty and Knott (1999), see above) within a user interface design framework is sought.

In addition to this, and as is discussed later, the model-based 'technology set' within which a metaphor abstraction may be set is substantial. It is therefore necessary to limit the scope and development of such a framework to within a tolerance that will allow meaningful evaluation within a tractable time scale (this is discussed further in chapter 3, section 7, chapter 5, section 2 and chapter 7).

2. Research aim and objectives

It is the aim of this research to develop a user interface specification framework that explicitly supports a metaphor model that can be integrated with extant user interface design views (since to not do so would run the risk of introducing just another inaccessible formalism). The determination of the nature of such framework must be guided by user interface design methods found within the literature. Research in the HCI community is characterised by collaborations between individuals working in a variety of disciplines including cognitive psychology, the social sciences and software engineering (Carroll (1997). Proposals for a scientific framework and principled application of HCI knowledge to design can be found in Dowell and Long (1989), Long (1997) and Sutcliffe (2000). Numerous design tools that support some HCI modelling techniques can also be found in the literature (see Bastide and Palanque (1999), Griffiths et al. (2001) and Paterno' (2000) for examples).

Integrating the wide range of design views and technologies used within the model-based community is considered a hard mapping problem (Puerta and Eisenstein 1999) and reports on the application of these methods in case studies are relatively few (Markopoulos et al. 1999). This research identifies the need for

a model-based abstraction of metaphorical design concepts as well as the important contribution of understanding how such an abstraction might actually be used in a real user interface project. It is therefore important to pitch the development and investigation of the user interface specification framework at a level that is most likely to generate useful insights into its application to design. To contrast possible alternative approaches: a craft-based approach to developing metaphors sheds little light on the problem of integrating metaphors with other model-based design views, whilst on the other hand, a 'sand-box' or laboratory oriented investigation is likely to yield little real-world validity[1]. For these reasons, this research seeks to develop an explicit metaphor abstraction and subsequently to validate its actual use with other design views in a software engineering case study.

To this end, this work pursues the following research objectives:

1. Identify extant HCI design models that might be extended to support metaphor abstractions

2. Develop a language that supports metaphor abstractions and integrates with models found in (1)

3. Evaluate the language developed by (2) with user interface designers/software engineers to assess the application of an abstracted metaphor layer on the design of a GUI prototype

Objective one is to identify appropriate HCI design models so as to delimit the views on user interface design (of which there are many) such that the problem becomes tractable. As outlined in section 1, constraining the number of views that are considered in this research is necessary in order to ensure the feasibility of the work. This is particularly important with respect to the case study since the software engineering participants' time and effort is at a premium. Chapters two and three set out the primary research and results concluded for this objective, identifying specific levels and types of abstraction considered potentially fruitful for a specification framework that explicitly supports metaphor descriptions.

Having identified model-based concepts to support the research goal, objective two is to synthesise a formal specification framework. It is important at this point to make a distinction between the conceptual objects and relationships that the framework embodies and its encapsulation within a formal language (the 'interface specification meta-language' or 'ISML' is described in detail in chapter 4). The former is the arrangement of existing and new model-based abstractions that will be used to cohesively describe the

---

[1] This is discussed further in chapter 5

design of a metaphor-oriented user interface. In itself, the framework is independent of any particular language but instead serves to capture and relate a variety of design concepts such as presentation, interactor and task views. An analogy might be drawn here with the MVC (Krasner and Pope, 1988) paradigm (as an abstraction) and its expression in the SmallTalk (Adams, 1988) programming language.

In the latter case, expressing the framework in machine parsable format is desirable for a number of reasons. Firstly, it is an ideal of the model-based user interface design community that the concepts utilised are machine processable and as such, many of the developments found within the literature have some degree of formalism. A formal language would also provide specific boundaries for the scope of the framework since the properties and mappings between the concepts would be explicit. Finally, the wide availability of tools for the creation and verification of models expressed in symbolic form offers the writer of a specification valuable assistance in documenting a design.

Whilst the creation of formal language is useful for the reasons described above, the primary focus of the evaluation is the *use* of the interface specification meta-language framework in a case study, not its implementation. Objective three therefore seeks to examine how the constituent concepts found within the framework are utilised by software engineers in their attempts to specify the design of a metaphor rich, graphical user interface. A number of approaches for the evaluation of the ISML framework are considered in chapter 5 and the case made for a qualitative, 'in vivo' methodology similar to recent 'action research' work reported in the software engineering community (Avison et al., 1999). In executing and analysing the results of the case study, insights will hopefully be gained into the actual use of a formalised metaphor abstraction and its integration with other model-based design views.

3. Thesis outline

3.1 Chapter 2: Introduction to model-based user interface design

This chapter provides an introduction to the evolution of model-based design in HCI. An overview of HCI as an engineering discipline is given, followed by an examination of a variety of models that exist to support the various views on interface design. An examination of the varying model-based approaches and their theoretical underpinnings provides a basis for identifying the appropriate model-based design methods for the development of a metaphor abstraction (objective 1).

3.2 Chapter 3: Architectures and tools in MB-UID

Formal user interface architectures and tool-based support for a number of model-based design approaches are examined here. The continuum from system functionality to user interaction is discussed,

outlining tool based support for input/output devices, presentation/component dialogue control, domain abstractions and tasks models. Mathematical and computational approaches to these abstractions are evaluated and used as a basis for the development of the ISML specification language (objective 1).

3.3 Chapter 4: The Interface Specification Meta-Language

In this chapter, the interface specification meta-language (ISML) is discussed in detail using a small-scale example to illustrate a complete construction (objective 2). The chapter concludes by summarising some of the lexical features and limitations of the language.

3.4 Chapter 5: Urban Shout Cast case study

Having demonstrated the specification of a simple interface on a small scale, a more realistic case study involving two teams of four software engineers each (and an interviewer) is documented. Each team had six months to develop a user interface prototype called 'Urban Shout Cast' (USC) – a 'proof of concept' system that allows remotely connected DJs to host a radio show for clients listening via an Internet connection. This chapter outlines methodology decisions and provides a qualitative analysis of the USC case study, using grounded theory (Glaser and Strauss, 1967). In the analysis, each team's reactions to the specification language are examined with respect to a) their use of ISML concepts to develop a metaphor model and b) their post-project perceptions of the usability of ISML in design.

3.5 Chapter 6: Evaluation of the USC specification

This chapter examines the specification data produced by both USC design teams during the case study with a view to a) identifying those aspects of design that the ISML framework captured and missed, and b) evaluating the extent to which ISML is capable of abstracting a metaphor independently of implementation (objective 3). ISML data generated by each group are compared by task, meta-object and interactor layers. Following this, a unified meta-object model is proposed and potential mappings to each group's implementation (interactor) solutions are examined and criticised.

3.6 Chapter 7: Conclusions

To conclude, a commentary on the over-all contribution that the ISML research has made to the model-based user interface design community is presented. The successes and failures of ISML are summarised and these findings are related to current research in this area. Changes to the specification process using ISML based on the case study experiences are discussed and proposals for further work are given. Finally, the contribution this work gives to the user interface design research community is presented.

CHAPTER 2 Introduction to model-based user interface design

1. Introduction

In this chapter, the evolution of theoretical frameworks and model-based user interface design is introduced, followed by an examination of what will be referred to as the 'products' of the methodologies described in the literature. These products are formal or informal descriptions of specific views of the user interface design problem (and in some cases used as part of a particular 'solution'[2]). The relative size and multidisciplinary nature of the HCI community means that an in-depth review of all the methodologies and their philosophical backgrounds is beyond the scope of this thesis. Instead, the focus of this chapter will rest on those design views that currently enjoy some degree of tool support (a technical review of these tools can be found in chapter 3). A broad introduction to the emergence of theoretical frameworks in HCI is provided as a backdrop to the subsequent review of task-oriented and model-based user interface design. The strengths and weaknesses of existing tools are critically appraised and considered in the context of the direction of model-based design as a whole and the challenges that face HCI design in the future.

2. Theoretical frameworks in HCI

The HCI research community struggles to find a unified framework and method with which it can apply theory to deliver specifications for designers (Sutcliffe (2000). From its inception, a number of frameworks and disciplines have been proposed to guide progression towards this goal. Moran's work identified the early coalescence of design methodologies, model generation and notations in HCI research and proposed the *Command Language Grammar* (CLG) framework to relate these concepts to design (Moran (1980). Nearly a decade later, Dowell and Long (1989) argued that HCI practice is a predominantly craft-based approach and that formal discipline knowledge is required to ensure the design of effective, interactive human-computer systems. Specifically, three deficiencies were identified: 1) a lack of integrated development practice, 2) uncertain measures of effectiveness and efficiency and 3) a lack of systematic programme to address these problems. In an attempt to put HCI research 'back on track', a concept of the general HCI design problem was proposed as a set of relationships between an interactive work system and its domain of application. The interactive work system (IWS) was described as a set of objects with attributes of varying complexity, the states of which are transformed by the

[2] Of course, it is arguably impossible to design a perfect interface since the solution for one user will almost certainly be sub-optimal for many others.

execution of actions determined by goals (Dowell and Long (1989). Work systems are said to transform such objects (which may belong to different domains) through the execution of tasks. It is principally the quality of the objects, their associated transformations (and incurred costs) that characterise the general problem of designing effective and efficient interactive systems. The IWS framework was extended by Long (1997) by specifying the relationships between research, discipline knowledge and its application to design.

The development of a broader philosophical framework for HCI knowledge described above has helped to shepherd attempts to define the relationships between practitioners from many different backgrounds who contribute to the discipline as a whole. Preece and Rombach (1994) modify and extend experimental approaches to design from the software engineering community to provide a framework for collaborating HCI practitioners and software engineers. A synthesis of methods from both design camps, the framework puts flesh on the bones of HCI philosophical structures by specifying four key dimensions, namely *goals*, *plans*, *methods* and *techniques*. Each of these dimensions encompass the ameliorating effects that a particular discipline has on HCI understanding, including a) quantitative and qualitative methods and data collection, b) objectives and focuses of the study and c) stakeholders and participators. Whilst it is still open for debate as to whether HCI can coherently be declared a science, there is now at least some informal agreement as to the methods and types of knowledge generation that each discipline contributes within the community (Carroll (1997). The emergence of these frameworks allows us to examine how laboratory based methods (an early influence that cognitive psychology has had on the HCI community) compare with the qualitative approaches of ethnography and participatory design. Whilst these attempts at normalisation improve the general description of relationships between contributors, they also highlight the hard problem of effectively communicating and integrating multidisciplinary theories and models in design at a practical level. Sutcliffe (2000) points out the apparent paradox faced by the effective delivery of HCI knowledge: that of hiding the complexity of a theory whilst at the same time providing comprehensible, theoretically sound and *generalisable* advice to designers. The evidence for this position lies in the landscape of HCI research communities; a brief examination of some of the major landmarks and their relationship to current design methods follows.

Since its inception, HCI has sought to apply the science of cognitive psychology to describe, explain and predict user behaviour. Early examples of this can be found in accounts of computer programmers' understanding of software algorithms (Shneiderman and Mayer (1979), (Kahney and Eisenstadt (1982). Later, Norman's theory of action (Norman and Draper (1986) provided a broad and high level account of human-computer interaction based on a process of the interpretation of symbols and the execution of actions through a mapping of syntactic and semantic structures. The prediction of human performance with an interactive system was led by the 'GOMS' (Card et al. (1983) framework, which provided

estimations of task execution time during error-free interaction through the quantification of *goals*, *operators*, *methods* and *selection* rules. Operators (defined as externally observable, simple actions or internal perceptual operations, such as scanning for a visual target on the screen) are used in the definition of methods (a potential set of operators used in a strategy for achieving some goal). Selection rules (IF-THEN conditions that test cognitive resources and external operations) are then used to choose methods in order to achieve the goal at hand (see Kieras (1988) for a more detailed account of this model). Other work extends this approach to simple graphics, see Lohse (1991) who reports on the prediction of performance in the readership of graphs. Despite the availability of toolkits supporting the GOMS method (see Beard et al. (1997), Khalifa and Kira (1992), and Baumeister et al. (2000) for examples) it suffers from practicality issues in deployment (Kieras (1988) and has only enjoyed success in a relatively narrow band of interaction paradigms (Carroll (1997).

More recently, a collaboration between HCI formalists and cognitive scientists (Butterworth et al. (1999) resulted in a formalised model of a display device (in this case a simplified web browser) combined with a cognitive model. A prediction of the preconditions under which user actions take place was then demonstrated, with the qualification that the assumptions underlying the cognitive model were both difficult to validate and also hard to delimit within the scope of the model. Problems and limitations like these, Sutcliffe (2000) argues, typify the problems facing cognitive psychology and HCI at present. Models such as EPIC (Kieras and Meyer (1997) and ICS (Barnard and May (1999) Sutcliffe suggests, do not easily scale to complex, multimedia systems and no effective method yet exists to translate this expert knowledge into a communicable and specific design for user interface developers. A consequence of the perceived failure of cognitive psychology to wholly underpin HCI was that other disciplines including anthropology and sociology found opportunities to address some of the problems that were found wanting by methods applied at the time. Critically, these views on design were contextual and emphasise the importance of design artefacts working within an environment of many interacting people and devices (Carroll (1997). Contextual approaches to design are frequently a mixture of qualitative and quantitative theory; Sutcliffe (2000) outlines a 'claims' framework that combines contextual descriptions of artefacts in use with theoretically informed design solutions.

Currently, there is no evidence from the literature that any large-scale, formal unification of scientific theory to inform and specify user interface design is within our reach. The implication of this is that effective design of interactive systems requires a development team that reflects expertise from many different fields. Additionally, many development projects will be faced with the prospect of having to employ 'craft experts' – those individuals who have a great deal of skill and experience in interface design. Wroblewski (1991) argues that theory applied in isolation can fail a design in context; the craft expert however, is able to use theories and apply them appropriately using his/her much deeper

understanding of the problem in its context. It seems unlikely that HCI will be entirely 'craft free' for the foreseeable future, but its eventual characterisation as a pure engineering discipline is highly desirable. To this end, a movement toward the synthesis of commonly used concepts and models that are shared by HCI sub disciplines and the software engineering community is in progress. Early adoptions of this approach, referred to as the 'enhanced software engineering' method, are identified by Wallace and Anderson (1993). In Benyon's introduction to model based design (Benyon (1996), simple interactors, task descriptions, object and data views are proposed as the foundations for design. Frequently, the synthesis of disciplines through models only affects a relatively narrow binding of features, such as input device and application integration (Accot et al. (1998). However, other work in the synthesis of models demonstrates composition of domain, task and presentation models (Griffiths et al. (2001). Not surprisingly, there are variations in the choices of model that are used in integration studies found in the literature as well as the technologies used to specify and implement them. A high level review of 14 model-based user interface development environments by da Silva (2001) examines a number of interactive system design abstractions found in the literature. These include application, task-dialogue, abstract and 'concrete' presentation components; in the following sections a variation of this framework is used to review contemporary model based design in HCI.

3. Task models

One of the main criticisms levelled at traditional software engineering methods is that insufficient attention is paid to how users will interact with the system to achieve their goals, and rather more on the underlying technical functionality of the system (Forbrig (1999). For many years, the concept of 'task' has played a major role in user centred system design (Storrs (1995). The analysis of people and their execution of tasks originate from industrial and military programmes engaged in enhancing work performance through the codification of the perceptual, motor and cognitive skills (Stammers et al. (1990). Since then, task analysis and its application in HCI design has diversified, attracting a variety of methods, notations and tools. Task models are generated from different sources and methods including cognitive psychology, formal task allocation plans from within a work context, software engineering documentation and ethnographic studies (Limbourg et al. (2001). A review of the methodologies for generating these models is beyond the scope of this work (readers should see Diaper (1989); Diaper and Stanton (2003 - in press) for details). Examples of the application of task models used throughout the design process can be found in the literature and include requirements elicitation (Richardson et al. (1998), specification and design (Navarre et al. (2001) and evaluation (Jambon et al. (1999).

In much the same way that HCI is a theoretically fragmented discipline, the task analysis community too strives for an agreement as to the constituent concepts that should make up a complete description of humans performing tasks with interactive systems (Limbourg et al. (2001). Much of the contemporary

work in this area deals with declarative and procedural models of the world of users, objects, actions and events. However, early task models, influenced by cognitive psychology, placed more emphasis on the interaction between a user's internal, cognitive knowledge of tasks and the interactive system. Comparatively 'fine grained' models such as GOMS (see above), TAG (Payne (1984) and ETIT (Moran (1983) use production rules to translate encapsulated user task knowledge into potential system interactions. More recently, the formalisation of such rule-based descriptions of task has allowed some researchers to implement machine learning techniques to develop task models from examples (Garland et al. (2001).

Whilst these models can provide some analytical power to the description of task, they offer little else to guide a designer (who is not an expert in cognitive psychology) towards a specific solution to a problem. The ADEPT toolkit (Johnson et al. (1995), alleviates this problem to a certain degree by de-coupling the task elicitation method, in this case Knowledge Analysis of Tasks (KAT) from the product of the analysis, the Task Knowledge Structures (TKS). Their toolkit allows the declarative representation of organisational, domain, problem solving and planning knowledge structures derived from the KAT analysis. Whilst it is suggested that any member of the design team may work with the toolkit during the design life cycle, the authors acknowledge the importance of the appropriate underpinning of task analysis conducted by experts. Clearly the value of any toolkit that supports task-based design will be influenced by the degree of knowledge and skill that is brought to it by the analyst. This should not prevent, however, the development of tools that allow the input and manipulation of common task concepts, and this is exactly what has happened.

Initially, tools for the development of task descriptions were tailored to just one or a very narrow range of task analysis methods (see Khalifa and Kira (1992); Bass et al. (1995); Beard et al. (1997) for examples). As the momentum for integration within the broader model-based design community has grown, variations of ontological views on the generic nature of task models have emerged. The historical build up to this position is littered with disagreements (Storrs (1995) regarding the definition of concepts core to almost all descriptions: the hierarchical decomposition of tasks. Arguably, the hierarchical task model, a product of the hierarchical task analysis method introduced by Annett and Duncan (1967), is encapsulated in some form or other in many contemporary task specifications. Terms such as 'goal', 'sub goal', 'task', 'sub task' and 'action' or 'unit action' have slightly different meanings, depending on the particular paper one might choose to read. However, the basic underlying principle remains more or less constant. The hierarchical task analysis (HTA) decomposes goals (desirable states of the interactive work system) into tasks (which may themselves be decomposed into lower order tasks) which eventually refine to a set of ordered or directed simple actions. Of course, the HTA depicts a highly simplistic view of human tasks and the shortfalls of this model (including problems associated with monolithic, inflexible,

idealised, error-free descriptions of task) are well known (Diaper (1989). Whitefield and Hill (1994) evaluate the components of HTA, TKS, GOMS and ICS models within the IWS framework (Dowell and Long (1989) to highlight the differences in task descriptions and their application to design. The comparison reveals important disparities between cognitively driven, predictive models (GOMS, ICS, and in an weak sense, TKS) and the design oriented, prescriptive descriptions (HTA and TKS). Cognitive models are psychologically informed and expressed by a vocabulary of fixed behaviours. However, argue Whitefield and Hill (1994), they suffer from either weak or no explicit definition of task decomposition and have little or no reference to domain objects. Conversely, design oriented descriptions provide an explicit 'blue print' for goals, domain objects, tasks decomposition and sequences. The prescriptive nature of these models does not consider the effect of human behaviour on the execution of tasks however, thus reducing its analytical power. Despite its potential to offer analytical methods for analysis, the emphasis on modelling cognitive structures in task analysis products has waned in recent years. On the other hand, the domain oriented description of task has become increasingly more popular, modifying the hierarchical model and extending it with contextual components to enhance its prescriptive power.

In van Welie's ontology for task world models van Welie et al. (1998), a review of extended concepts in task models included temporal structures, user interface components, enhanced task units (allowing information passing and pre/post conditions for tasks) and organisation, agent and role definitions. In addition to the enhanced domain modelling, the 'Groupware Task Analysis' model van der Veer and van Welie (1999) encapsulates new semantics, including relationships between objects, tasks and users, events and triggers, and task constraints through definition of roles and responsibilities. The scope of the task model has also been extended to include the allocation of the roles of 'protagonists' (both users and system components) in an interactive scenario Filho and Liesenberg (1999). This high-level abstraction of task roles has been proposed to support unexpected changes in task context; a directed graph of nodes depicts protagonists' changes in intention between sets of tasks that make up the work scenario as a whole. Explicit inclusion of the concepts discussed above marks the clear strengthening of both contextual views of task analysis and also a significant step towards a unified model-based approach to user interface design. Indeed, Pribeanu et al. (2001) argue that an explicit contextual framework within such task models is essential for the design of the new wave of interactive systems. They suggest that the wide array of personal computing devices available to users means that a task will be situated within both the environment in which it is performed and the hardware/software solution used in its execution.

The inclusion of wider contexts found in contemporary task models suggest that the end of a task specification and the beginning of a domain or dialogue model is somewhat blurred. Forbrig (1999) suggests that, historically, the role of the task model was primarily to support the design process whilst

domain modelling supported actual design. He argues however, that the relationship between tasks, the user, the problem domain and the interactive system inevitably interact and co-evolve as the extant system is transformed into a new design. The impact that technology has on the task and domain model also impacts on design, and *vice versa*, making it difficult to understand one without the other. It seems likely that task and domain modelling will eventually merge, but Limbourg et al. (2001) identify a number of problems that must be resolved before such a synthesis can be addressed, summarised here:

- Lack of heterogeneity and understanding of task concepts
- Mapping of concepts between models and between toolkit software formats
- Reduced communication between project stakeholders through lack of development software integration
- Needless reproduction of research and development efforts

In their meta-task model, Limbourg *et al.* redefine ten task models as entity relationship diagrams and from this, a generalised model is created (see Figure 1) and used as a part of the DOLPHIN user interface design assistant.

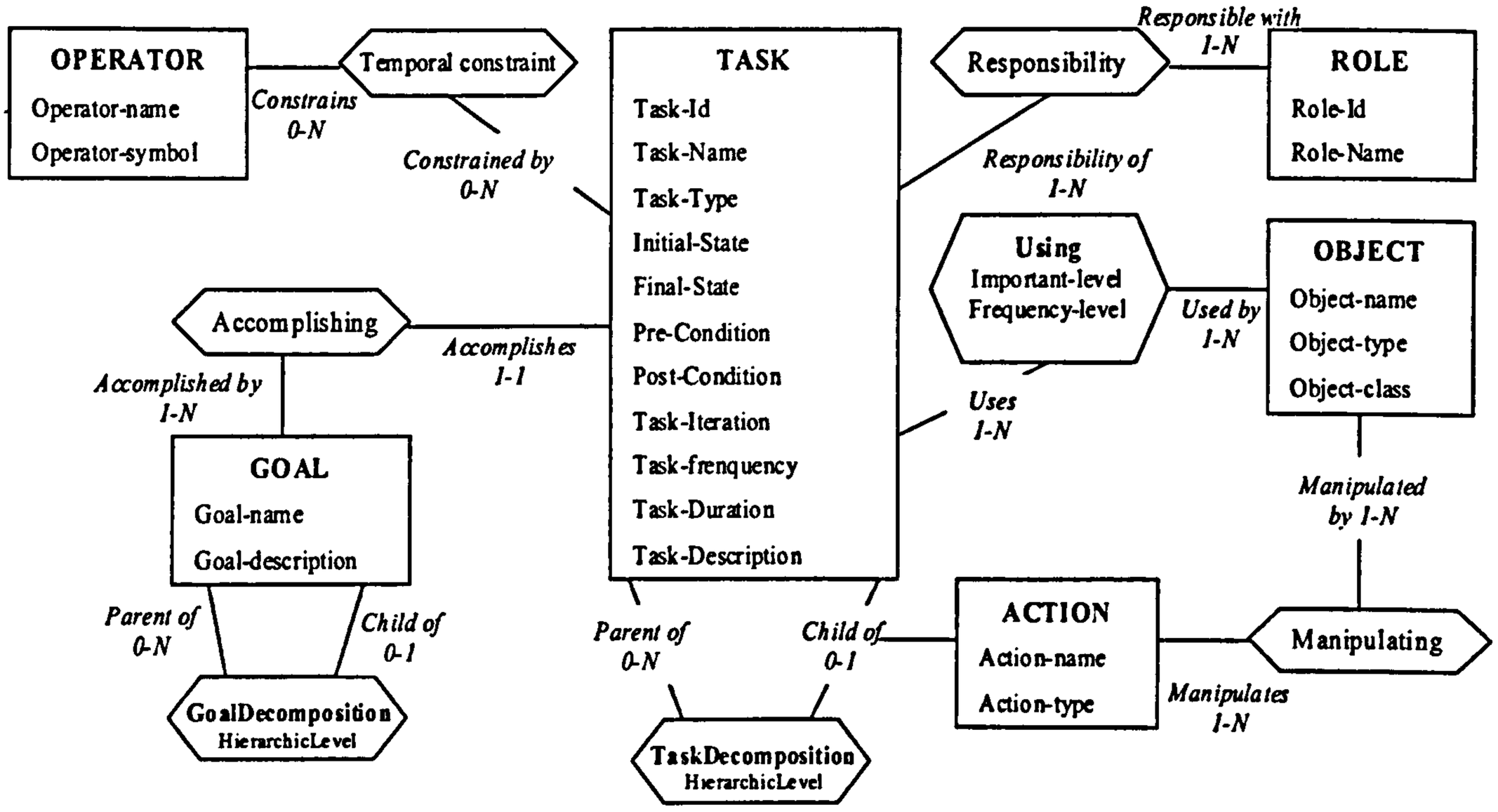


Figure 1 The meta-task model Limbourg et al. (2001)

The role of task analysis in user centred system design has become well established over the years and the application of cognitive modelling in the analysis of human-computer interaction and the prediction of work performance continues to inform the research community Diaper and Stanton (2003 - in press). Complex cognitive theories are not easily shared amongst all the stakeholders of an interface design project however. Recent discussions within the HCI research community, Carroll (2002); Diaper (2002), highlight the problems associated with the application of this approach to design. In an attempt to ameliorate this impasse, various notations and toolkits are being developed to partially integrate task-orientated views of design with other components of the envisaged solution. In the succeeding section on model-based user interface design, complementary design views and tools that have helped enhance user interface development are reviewed and their recent synthesis with task views discussed.

4. Model-based user interface design

The underlying design and functionality of a system are, at least in part, an expression of some model of the problem that the system has been designed to solve. Typically, this model (or parts of it) can be found in a number of different sources including a) the structures and functions in the language used to implement the program, b) software design documentation, c) user manuals and d) the user interface design. According to the design methodology and the types of notation used, this model may range from highly implicit (distributed within the source code) to highly explicit (expressed as data-flow diagrams; entity relationship models; object oriented models and so on). However, the problem of linking the underlying domain model and functionality of a system with the user interface is considered difficult

since the complexities of an interactive system become immediately apparent Patern and Mancini (1999). For this reason, system designers have sought means of abstraction that will allow them to proceed with resolving relatively high-level design issues without having to commit to a large number of low-level implementation details.

For such abstractions to be possible, frameworks were conceived that would allow the separation of the various activities of an interactive system managed during the course of interacting with the user and information processing. The Seehiem model Green (1983) proposes three high-level abstract components, in turn responsible for the device-level input and graphical presentation of output, a dialogue controller and an application model. Due to its inflexibility and lack of guidance on the integration of these levels within an application framework or design methodology, the Arch/Slinky Gram and Cockton (1996) model superseded the Seehiem model. In this framework, the relative bindings between each of the levels are more flexible, allowing for varying emphasis on the importance of each layer according to the context of the application being designed. Many of the model-based approaches use architectures that reflect and extend this basic separation between the graphical presentation and the application layers in an attempt to break down the design problem into partially de-coupled parts. According to da Silva (2000), such approaches provide three main advantages over traditional design models:

- They can provide a more abstract description of the UI than UI descriptions provided by other UI development tools

- They facilitate the creation of methods to design and implement the UI in a systematic way since they offer capabilities: (1) to model user interfaces using different levels of abstraction; (2) to incrementally refine the models; and (3) to re-use UI specifications

- They provide the infrastructure required to automate tasks related to the UI design and implementation processes

However, successful de-coupling of design abstractions necessarily implies the successful means of integrating them into a coherent whole, and it is this 'mapping problem' that has been the focus for the proponents of model based design Puerta and Eisenstein (1999).

Early model-based tools were designed primarily to automate the mapping between common user interface components or 'widgets' (such as buttons, lists, menus and so on) with the underlying application model. In these cases, the application model provides abstractions of the data that are

available for use by the user and a rule-based system infers design choices based on: a) the operations that can be made on the data; and b) the availability of appropriate GUI components. An early example of this is found in the 'DON' interface design assistant Kim and Foley (1990). Rules encapsulating the application domain (referred to as 'conceptual knowledge'), organisational and style templates, graphical theory and 'design knowledge' (mappings between the application abstract and widget presentation) are automatically combined to generate a user interface. A slightly different approach to this mapping is taken by HUMANOID Luo et al. (1993). In HUMANOID, the approach to reducing complexity is supported by machine management of 'design goals'; solutions to each goal are modelled as interactions between the user and the system. Similar to some task modelling approaches, goals can be broken into sub-goals; unlike task models however these goals focus on the mapping of only simple interactions to pre-defined application abstractions. Binding of task descriptions to application functionality is also featured in the BOSS environment Schreiber (1994); here, tasks are described as sets of hierarchic interaction graph templates or HITs. Each HIT encapsulates links to attributes, data flow structures, function calls and presentation components. BOSS extends traditional automatic generation of the presentation of the user interface by using a run-time engine capable of generating interfaces from data created by users. Rule-based generation of non-WIMP interfaces has also been developed, where there is a requirement for domain-specific graphical representations. The ADDI tool ElSaid et al. (1997) allows users to select aspects of the domain model they wish to examine; a presentation manager then selects from a number of knowledge bases to transform the data into an interactive, graphical display.

It became apparent that designers often prefer to 'get their hands dirty' with the mappings between the domain model and the interface which led to a shift in design support that model-based design environments might provide. The Interactive UIDE Frank and Foley (1993) is an early example of the change in perspective, providing both automatic support as well as an editable mapping notation, allowing expert designers to specify their own links between domain and widget models. At the same time, the gathering acceptance of, on the one hand, user-centred system design and on the other, object orientated design techniques, led to new opportunities for the synthesises of design views. Scenario-based design techniques Carroll (2002) and use-case descriptions Jacobson et al. (1992) are combined in an object-oriented support tool called the 'Point of View' (POV) Browser Rosson (1999). The POV tool allows the analyst to create objects that have functional responsibilities and 'point of view' that relates to the context of a specific scenario. Bound to each object is a set of textual descriptions of usability claims that can highlight positive and negative consequences of the use of an object within the context of the scenario. As such, this tool combines, in a very weak sense, user centred system design considerations.

An analytical approach to scenario-based design is outlined by Benyon and Macaulay (2002) in their description of the PACT framework in which data from scenarios is refined into a model of objects and

user actions. Other expansions of scenario-like descriptions include the extension of UML Fowler and Scott (2000) to provide facilities for user interface design, renamed UMLi da Silva and Paton (2000). In this extension, the user interface notation encapsulates high-level GUI concepts such as containers, input/output points, display parts and editing parts; these are subsequently mapped to a generic widget template called the Abstract Presentation Pattern. In addition, task descriptions are expressed using an extended version of activity diagrams; activities identified in use cases are linked using modified state chart notation and high level user interface abstractions.

The model-based approaches discussed so far represent strong system-orientated views of development; with the exception of UMLi, their task descriptions are constrained within the specific scope of the system's expected functionality rather than within the task domain of the user. However, use-case approaches in interaction design have also been criticised on the grounds that they combine both system and user variances, which may confound design decisions Markopoulos and Marijnissen (2000). The conceptual separation of task, presentation and dialogue from domain models helps to isolate user-orientated issues; a number of toolkits provide such distinctions. Currently, the model-based approach to design enjoys the inclusion of a number of additional user-centred constructs, although there is still no general consensus as to exactly which are appropriate or how they can be coherently integrated da Silva (2000). The synthesis of hierarchical task specifications and an extended entity relationship model (ERM) in the TRIDENT environment Bodart et al. (1994), is graphically integrated in an activity chaining graph to provide a dialogue model. Presentation units (PU) are defined for each task and encapsulate any number of entities from the ERM; six different contexts in which a PU is implemented are provided, depending on input and output requirements. In keeping with the automation maxim, a heuristic engine is also provided to offer automatic selection of interface components based on the PU interaction type.

Recent research has advanced the scope of model-based design, offering explicit structures and mappings that reflect high-level abstractions of interactive software such as the Slinky/Arch framework as well as binding task oriented models. The Model-Based Interface Designer or MOBI-D Puerta (1997; Puerta and Eisenstein (1999) is a development environment comprising a number of tools that support the specification of tasks, domains, user profiles and presentation and dialogue models. Similarly scoped work can be found in the Teallach environment Griffiths et al. (2001) which supports domain, task and presentation models with particular focus on the integration of object-oriented databases. In a review of the model-based paradigm, de Silva (2000) provides an overview of the primitive components of 14 tools, organised into application, task-dialogue, and abstract presentation and concrete presentation categories. A more detailed account of the underlying methods and technologies that support the abstraction of tasks,

dialogue and problem domains is given in chapter 3. A brief account of the emergence of the 'abstract' and 'concrete' specification of the user interface is now given for completeness.

In an attempt to reduce complexity and to reduce premature commitment to specific implementation decisions, the presentation component of the user interface is further refined by the introduction of abstract descriptions of interaction components. It is common parlance to use the word 'interactor' to refer to any component of the user interface that either displays graphics or receives user input or both. In fact, the term 'interactor' has a much stricter definition within formalist circles Duke and Harrison (1993) – a review of the variations on its formal structure and application is found in chapter 3. Formal definitions withstanding, the 'abstract interactor object' is frequently used in model-based design to encapsulate the basic characteristics of WIMP components without committing to a particular implementation, see Schreiber (1994); Bodart et al. (1994); Puerta (1996); da Silva et al. (2000) for examples. Concrete instances of the abstract are dependent on the technology that the tool supports: platform independent Java Swing conversions are becoming popular, see Luyten and Coninx (2001); Griffiths et al. (2001). However other platform specific conversions for Microsoft Windows Puerta et al. (1999) and Apple Macintosh Schneider and Cordy (2001) have also been developed.

In contrast to the task-oriented view, traditional model-based design has a much stronger emphasis on the mapping of system side abstractions to interface component technologies, rather than descriptions of the world of users and work environments. Its power lies in its ability to join high-level software engineering design concepts with user interface components; uncoupled descriptions of user tasks give no guidance for programmers in this matter. Recent work connecting task and domain oriented models has improved communication between HCI analysts and software engineers still further by incorporating task and domain modelling into one design environment.

5. Discussion

It is clear from the even limited range of methodological viewpoints in HCI surveyed here that the design of effective and efficient interactive systems is a non-trivial problem. Blandford and Duke (1997) argue that design models must make a trade off between the general applicability of their concepts and their power to explain how and why a particular design improves usability. In this chapter, the notations and tools are for the most part devoid of cognitive user models (excepting the GOMS tools); this is a significant trade-off for model-based design practitioners since user perception and behaviour is critical in the determination of task execution. As discussed above, the integration of cognitive models into the broader engineering of interactive systems is a formidable problem indeed and unlikely to be formally resolved soon.

Not unexpectedly then, current trends in tool-based support for task and model-based design have had to delimit the scope and complexity of user interface design concerns they address, in order to ensure that the realisation of these tools is tenable. In addition to the onerous academic endeavour of trying to transform and accommodate the multitude of methods into a rigorous engineering method, the HCI community is also faced with the problem that computing technology will not wait until some sort of order is finally resolved. Personal computing technology now offers graphical processing power capable of rendering interactive, cinematic quality virtual environments for a variety of problem domains Kirk (2003). Although not technologically of the same order of power, mobile computing devices also represent a major challenge to the HCI community since these devices demand new contextual considerations and implementation constraints Mueller et al. (2001).

With the prospect of increasingly 'rich' interactive systems becoming available to the public, the opportunities for novel user interface designs deployed across multiple hardware platforms grow. At present, the mappings between task or domain models to WIMP components can be guided by heuristics that have emerged as a result of many years of research (and industrial development) working with the ubiquitous desktop paradigm. The progression toward unity in this regard is threatened by technological change because MB-UID (model-based user-interface design) tools do not provide mappings to concrete solutions outside of a (often implementation specific) 'WIMP' environment. Notations such as UAN Hartson et al. (1990); Hix and Hartson (1993) map classic 'desktop metaphor' objects and actions to task structures. However, the fundamental concepts of this metaphor are implicit in the lexicon. Metaphors are important conceptual devices since they communicate the state of the system in meaningful and often creative ways to the user. Through metaphors, users are offered a means of translating their task intentions formulated in terms of operations in the real world into actions they can perform at the user interface Dix et al. (1998). The limited and implicit treatment of metaphor in MB-UID research therefore:

- Inhibits innovation and development with new user interface technologies

- Lacks the intermediate mapping that metaphors provide to aid users in the execution of their tasks.

In addressing this problem, it would be desirable to utilise the considerable progress the MB-UID community has made in unifying design views. A more detailed examination of the abstractions employed to this end must therefore be conducted in an attempt to identify mechanisms already in use that might be modified or extended in order to support metaphor modelling – see chapter 3.

6. Conclusion

In this chapter, model-based approaches to interface design have been introduced against the background of a broader, generalised HCI knowledge framework and task analysis. Model-based user interface design research within the HCI community continues to generate toolkits that support notations combining core user-centred system design concepts. For contemporary 'WIMP' based design solutions, these toolkits support the appropriate level of abstraction and may soon become standard parts of industrially accepted software development packages. However, personal computing technology is delivering entirely new forms of interaction that extend far beyond traditional WIMP solutions. The potential for innovation in design would be more effectively catalysed with an explicit user interface metaphor model.

CHAPTER 3 User Interface Design Architectures, notations and tools

1. Introduction

This chapter continues the review of model-based design by examining the architectures that support the expression of the principal abstractions of model-based user interface design: input/output devices, presentation component/dialogue control, domain abstractions and task models. For the most part, these architectures can be regarded as 'implementation independent' – i.e., they are levels of abstractions that have been realised using a variety of technologies from computer science, software engineering and computer graphics disciplines. To begin, an outline of these underlying technologies is given and put into the context of the development phases of an interface design project. This is followed by a review of model-based abstractions, observing their variations in conception and application within a broader, interactive system framework. In conclusion, the relative merits of these approaches are considered and an extended framework proposed for the inclusion of metaphor-oriented user interface design.

2. Supporting technologies for user interface modelling

During the development lifetime of an interactive system, numerous tools and formalisms are employed to support the various design stages including the elicitation of requirements, design, specification, prototyping, development and evaluation. The scope of this thesis is delimited to the consideration of only the specification and prototyping phases of the life cycle, although it is recognised that user interface design considerations proliferate throughout Faulkner and Culwin (2000). Even within this narrow development window however, a wide range of tools exists ranging from informal, craft-based support to highly abstract, formal methods and notations. The nature of these tools also reflects their application during the specification and requirements phases.

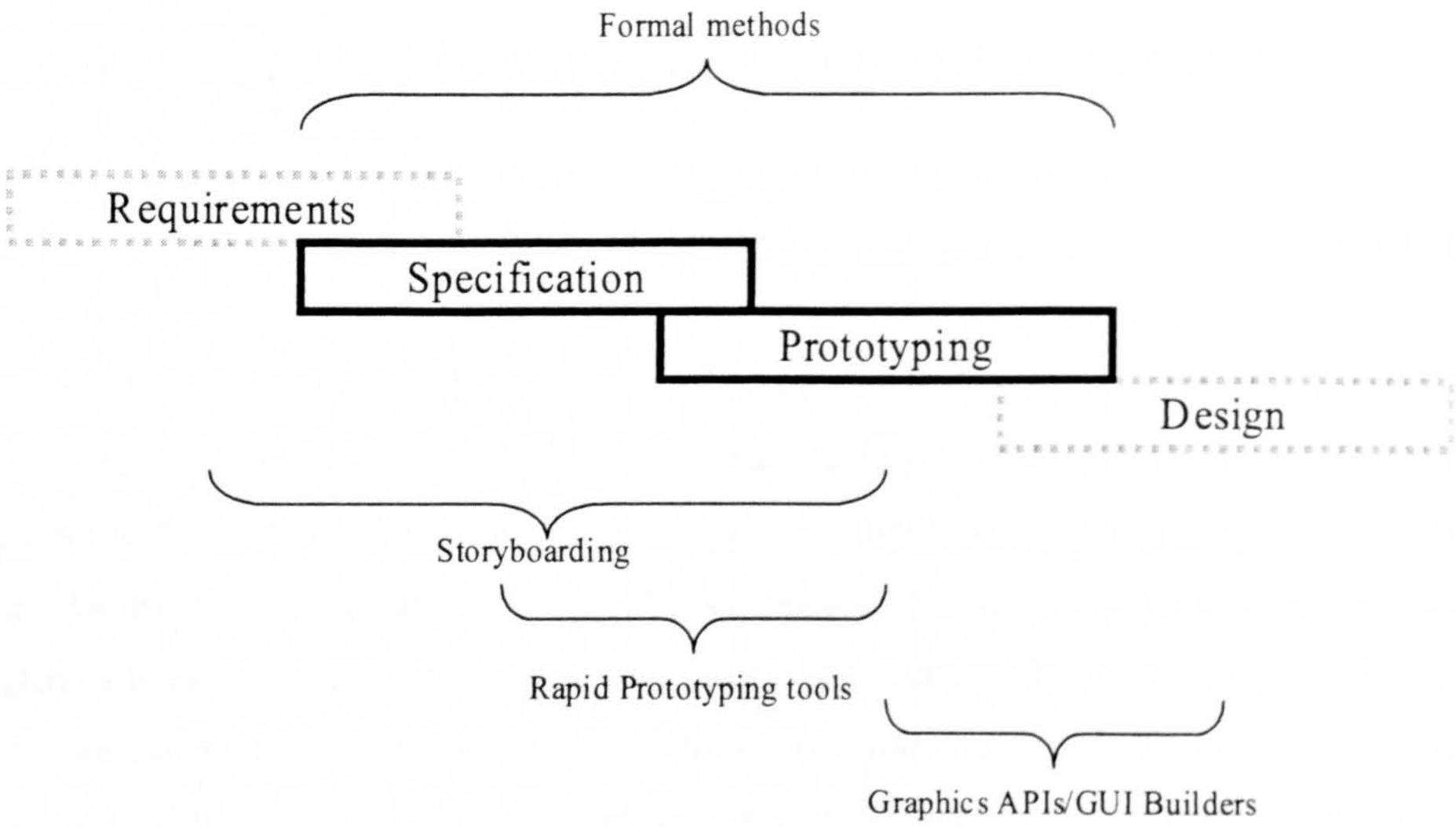


Figure 2 Methods and tools in user interface design phases

In figure Figure 2 a summary of these methods and technologies and their application to the life cycle is provided; in the following sections, each are discussed with respect to the life cycle and their support for model-based design, contrasting prototyping and implementation tools with formal methods and models.

2.1 Storyboarding

One of the simplest and most immediate ways of conveying some of the features of a user interface is through storyboarding. The use of storyboards allows collaborating end users of the system to quickly understand some of the designer's intentions for the interactive system and to contribute to the design process at a level that is comprehensible to them Preece et al. (1994). Like those used in film production, storyboards convey the appearance and some simple behaviours (of the system, in this case) through the use of a sequence of annotated drawings. An electronic extension of the storyboarding technique is found in the SILK prototyping tool Landay and Myers (2001); designers are able to sketch common user interface components (such as buttons and sliders) using a graphics tablet or mouse and SILK will convert them into executable prototypes. Simple dialogue control is supported through the depiction of arcs, drawn by the user, connecting buttons to the display of new windows. In themselves, storyboards provide little guidance to the software engineer with respect to design and implementation issues proper and so are unlikely to be of any great use after the early prototyping stages. With respect to user interface models, arguably storyboards only really support the 'abstract' specification of interface components and little else.

2.2 Rapid prototyping tools

The relative simplicity of the storyboarding technique means that expressing dynamic aspects of the user interface is very difficult. In addition, the user is unable to interact with the design (with the exception of SILK) and so will not have much of an idea as to how the final system will actually behave. To rectify this and at the same time maintain the desirable *rapid* production of prototypes, high-level prototyping tools have been developed including *Hypercard* Atkinson (1987), ICON Chung and Shih (1997) and *Director* Canter (1988). Tools such as *Director* have been used in the rapid production of interactive, multimedia prototypes Millard et al. (1998) but have also been used, in their own right, to develop commercial applications (particularly in the gaming and web-based markets). Prototyping tools such as these provide a graphical, direct manipulation toolkit to create, place and animate interactive elements at the user interface. User interactions via the keyboard and mouse can also be captured and a scripting language allows the simulation of system responses at run-time. As a result, designers can quickly mock up the appearance and to a limited extent, the behaviour of the system. An alternative to the procedural scripting approach can be found in the Penguims system Hudson (1994), in which a spreadsheet model is used to declaratively define and express the relationships (through equations) of the graphical components of the system.

The limitation of the behavioural modelling in prototyping tools typically appears at the point where the system requires semantic operations from the underlying domain model or functional core (static, 'dummy' data are often used in its place). To this extent, prototyping tools fall short of the capabilities of a fully-fledged development environment on the basis that:

- These tools primarily support only high level presentation and interaction characteristics
- Programming support is rather less powerful than traditional languages (ie., C/C++)
- Prototyping tools are 'closed' development environments

Due to these limitations, the products of rapid prototyping tools rarely extend into the design phase of a project. From a model-based point of view, prototyping tools such as *Director* provide support for 'concrete' component specification and a proprietary, high-level, input event-hierarchy. It could be argued that since a programming language is provided in Director, in a very weak sense, dialogue and domain abstractions are also supported. However, since these models would have to be explicitly coded it is argued here that this provision is negligible.

2.3 Graphics APIs/GUI builders

In contrast to the limitations discussed above, graphics application programmer's interfaces (APIs) and GUI builders are often integrated into 'industrial strength' software development tools such as *Visual C++* Microsoft (2001), *C++ Builder* Borland (2001), *UIM/X* VisualEdge (1997) and *CodeWarrior* Metrowerks (2003) . User interface builders provide the developer with a palette of standard WIMP components such as buttons, menus and windows. The apparent ease with which it is possible to 'draw' user interfaces with these tools is comparable with the prototyping tools already discussed. However, this apparent simplicity belies the underlying complexity and necessary computer programming skills required to implement non-trivial designs. In addition to the increased syntactic and semantic complexity that a more powerful and general purpose programming language (such as C/C++ or Java) entails, the developer must now concern him/herself with the particular details of retrieving input from and output to the user. Broadly speaking, the software engineer is presented with two possible options: a low-level, device rendering development path or an operating system dependent WIMP component management course.[3] Low-level device rendering means working with computer graphics APIs such as *OpenGL*, *DirectX* or *PHIGS* - these APIs provide low-level or 'direct' access to the user interface devices and rendering methods. The advantage of this approach is that the developer is not constrained to a limited range of interaction components; payment for this advantage is made through the extended effort required to implement a user interface environment from scratch. Some reduction in the work required for graphics rendering can be found through the use of functional graphics languages. 'Pictures' Finne and Jones (1995) is a device independent graphics language that supports the composition and translation of vector-based graphics primitives. The Haggis graphical framework Sage and Johnson (1997a) extends this by allowing many virtual, concurrent input/output devices to be managed simultaneously.

Alternatively, the developer may chose to use an existing, proprietary WIMP framework such as *Microsoft Foundation Classes* Petzold (1999) or *Motif* Brain (1992). To his/her advantage, the engineer can re-use previously built components and delegate much of their maintenance at run-time to the operating system. On the other hand, developers must be familiar with and work within the framework and constraints of the component set; this makes moving out of the prescribed rendering and event-capturing framework very difficult or impossible. In an attempt to reduce the programming complexity of such frameworks, Rajagopala et al. (1997) specify a higher level object oriented API for X/Motif programmers. Savidis et al. (1998) extend this idea using the 'PIM' meta-programming layer, allowing the specification of a variety of different interface APIs within a single framework. Despite this, the considerable increase in complexity and programming effort places these technologies firmly in the

design phase of a project cycle in all but trivial prototypes since it is undesirable to devote considerable software engineering resources to implement a trial interface that is likely to be discarded later on.

It is difficult to quantify to what extent modern programming environments support model-based design abstractions. The reasons are two-fold; a) the environment can potentially express all abstractions implicitly (similar to the prototyping argument above) and b) the modular nature of modern programming environments allows the inclusion of specialised abstractions. An example of the latter argument can be seen in Borland's '*TAction*' component – an object that abstracts an *action* (irrespective of how the action is physically performed) that has some effect on the functional core. The Borland GUI framework allows any number of concrete interface components to point to this abstract action on receipt of user input; in this sense it could be said that some support for the mapping between task and domain models exists. The extensive use of object-oriented programming concepts Yourdon (1994) allows the encapsulation, aggregation and specialisation of interface components, the effect of which is that dialogues are managed through method calling between 'super components'. Critically, whilst these abstractions are *possible* they are not (excepting a few rare cases) an explicit part of the user interface development environment in these tools. To put it another way, tools like *Visual C++* or *C++ Builder* demand that the programmer translate UI models into a concrete framework of programmable objects that can be compiled.

3. Formal methods

So far, the technologies presented have broadly fallen under the remit of 'implementation oriented' tools: software developed for the generation of user interfaces within a specific user interface technology (for a review of these conventional tools, see Myers (1995). In contrast to this approach, the models and notations used in formal methods are of a higher level of abstraction. They do not consider the 'mechanical' details of the system, instead the 'what' is being specified, rather than 'how' it will be implemented; formal methods are synonymous with the specification phase of the project life cycle Hall (1990). It is *not* the intention of this thesis to construct or extend a detailed account of the mechanics underpinning formal methods. Rather, a brief and high level description of these approaches is given so that the relationship between model-based abstractions and the mechanics that are used to reason about them can be demonstrated.

The languages available to formal methods practitioners include mathematical notations such as Z Spivey (1989), VDM Jones (1986), CSP Hoare (1985) and LOTOS Bolognesi and Brinksma (1987). In addition, a number of computational tools also exist to electronically model system behaviours, including parsers

---

[3] In fact, the engineer may not have a choice, depending on the requirements of the software project.

for context free grammars Levine et al. (1992), State Charts Harel (1987) and Petri nets Peterson (1981). Disposed with these languages and tools, formalists are able to rigorously describe abstract entities, their properties and operations. A popular distinction between the 'design, build and test' prototyping cycle and formal specification is that whilst testing can reveal the existence of 'bugs' within a system, only formal specification can demonstrate the absence of them Hall (1990). The application of these tools is desirable since it provides a mechanism for unambiguously specifying and reasoning about potential designs before committing resources to implementing them in code.

Formal mathematical languages are used to specify interactive systems at different levels of abstraction according to the domain of interest (see Brun and Beaudouin-Lafon (1995) and Campos and Harrison (1997) for overviews). At a relatively high level of abstraction, informal usability properties of a system such as 'what you see is what you get' (WYSIWYG) can be expressed in a rigorous manner: Dearden and Harrison (1997) demonstrate this formally using the RED-PiE model Dix (1991). Here, an interactive system is modelled as a set of user commands, *C*. Sequences of these commands input to the system is termed as a program, *P*, and a set of effects that represent output to the user, as *E*. Programs are mapped to effects via an interpretation function *i*. The effects *E* are mapped to a set of results *R*, reflecting the states of the objects being manipulated by the user, and a set of displays *D*, representing the information presented to the user at any point in the interaction.

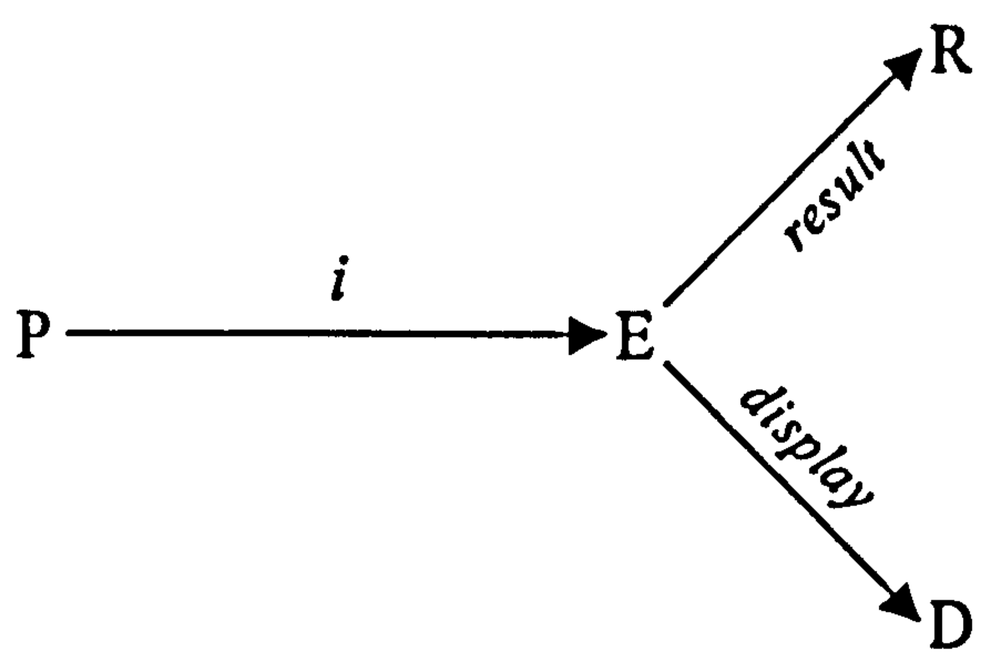


Figure 3 The RED-PiE model Dearden and Harrison (1997)

In their example, Dearden and Harrison use an example of a word processor in which result $r \in R$ specifies the current state of the document if it were printed and $d \in D$ represents a whole or partial representation of the document displayed to the user on the screen.

From this, the notion of WYSIWYG as a principle of *observability* such that what the user sees accurately reflects the states of the objects being modelled by the system is given by reasoning about two input sequences p and q to the system:

$$\forall p, q : P \mid \; display(i(p)) = display(i(q)) \cdot result(i(p)) = result(i(q))$$

Dearden and Harrison (1997)

Informally, this specification reads: for all programs p and q whose displays after interpretations are equivalent, by implication, the states of the modelled objects are also the same. The authors note, however, that it requires a skilled designer to apply the appropriate levels of abstraction; for example, the above formalism only holds for documents that can fit within the display capacity of the screen and so further extensions to the equation are required for the realistic modelling of a word processor. Extensions of display-oriented formal reasoning can be found in Doherty and Harrison (1997) on the transformation of logical operations required to perform a task into their perceptual equivalents.

Finer grained formal abstractions of the interactive system emerged with the concept of 'interaction objects' Duke and Harrison (1993), sometimes referred to as 'agents' Coutaz et al. (1995) or simply 'interactors' Hussey and Carrington (1999). The concept of an interactor is not itself an explicit part of the formal specification languages outlined here (indeed, a number of variations of interactor architectures exist, see section 5.3). Informally and at a high level, interactors can be considered as objects that privately hold state information (referred to as the *abstract* or *model* part) and maintain communications with de-coupled *display* (or *presentation*) and *controller* parts. Whilst the exact roles of the display and controller parts vary according to interactor architecture, it can be argued that between them they manage the communications between the user, other connected interactors, and the functional core.

Formal models of interactor architectures have been expressed in logic-based specification languages such as Z to reason about interactor data, relations and functions, see Hussey (2000). Other formal approaches include algebraic specifications such as LOTOS Palanque et al. (1996) and GRALPLA Torres et al. (1996) which have been used to specify communications and event passing between interactors. Two well established interactor variants, the MVC Krasner and Pope (1988) and PAC Coutaz et al. (1995) frameworks are contrasted by Hussey and Carrington (1997) using the Object-Z language Duke et al. (1995), an object-oriented extension of Z. Similar work by Hussey and Carrington (1998) extends

formal reasoning using Object-Z to the platform independent specification of common WIMP components such as buttons and menus; the specific behaviours of a particular widget class (such as the Tk component classes) are derived from the abstract classes.

The temporal ordering of operations within interactor models is refined by Markopoulos (1997) in the specification of the ADC framework using LOTOS – a communicating process algebra that incorporates a data typing language, ACT-ONE. The application of LOTOS to interactor specification allows a distinction to be made between abstract and display part operations of the interactor and the temporal ordering over their execution. By specifying the temporal sequencing of operations, LOTOS allows formalists to reason about 'when' behaviours occur either synchronously or asynchronously within a system and so model the dialogue between the user and the interactive system. The translation from a LOTOS specification to a prototype has been demonstrated by Sage and Johnson (1997b). In this demonstration, the Haggis tool Finne and Jones (1995) was used to implement a simple interactive game; a larger scale case study by Sage and Johnson (1998), an interactor-based prototype of a multi-user system created using the Clockworks tool Graham and Urnes (1996) was converted to LOTOS for formal model checking.

Whilst formal methods of specifying interactive systems are attractive because they may lead to the proof of specific behaviours in a system, they do not in themselves generate system designs but must instead be incrementally converted, in small steps, toward a final system Hall (1990). Formal specifications have also been criticised for being difficult to use, requiring substantial training on the part of the writer Carr (1996), Jambon et al. (1999). These drawbacks reduce the communicability of potential system solutions between designer and user, so some compromise is desirable in which the high-level, low commitment abstractions of formal models combine with the enhanced immediacy and accessibility of prototyping tools.

4. Computable models

Rather than adopting a purely analytical approach to formal specification, some model-based approaches make use of existing computer science technologies to specify, compile and run simulations of the prototype user interface. Although this approach does not wield the same analytical power as those used in formal methods, the underlying concepts used to specify the interface are at a similar level of abstraction such that they can be converted for formal analysis, as in Sage and Johnson (1998). Three computable models commonly used in model-based interface design are examined here: *context free grammars*, *state models* and *Petri nets*.

4.1 Context free grammars

Originally used to specify programming language syntax for compilers, context-free grammars (and associated parsers) have been re-used to cover a wide range of user interface design structures including task models Payne (1984), VR systems Jacob et al. (1999) and multiple hardware target specifications Mueller et al. (2001). The Backus-Naur Form or BNF Naur (1984) system for expressing formal grammars characterises the general mechanism for describing an arbitrary, but well-formed grammar.

```
Picnic ::=    SandwichSelection
            | SandwichSelection Drinks ;

SandwichSelection ::= SandwichSelection Sandwiches
                    | Sandwiches ;

Sandwiches ::= CheeseAndPickle | BeefAndMustard ;

Drinks ::=    Cola | Orange | Tea ;
```

Figure 4 Sample BNF grammar

Well-formed grammars specified using BNF[4] describe a grammatical tree in which 'leaf nodes' are rewritten as higher order branch nodes, and so on, until the root node is reached. In Figure 4, a simple selection of sandwiches and drinks are legal components of a picnic; whilst drinks remain optional, sandwiches are a mandatory element. Tools such as Lex and Yacc Levine et al. (1992) automatically generate code based on a lexical specification that identifies legal alpha-numeric symbol sequences which are then passed as tokens for the parser to assemble as a well formed grammatical tree.

Jambon et al. (1999) use a formal grammar to specify both task structures based on the MAD formalism Scapin and Pierret-Golbreich (1989) as well as dialogue sequences to describe interactions with a CAD system. The syntax for a functional language supporting the generation of WIMP components described by Schneider and Cordy (2001) is also expressed as a formal grammar (similar work using XML can be found in Mueller et al. (2001); Luyten and Coninx (2001). The highly structured nature of formal grammars makes them attractive candidates for specifying conventional WIMP interfaces since components such as menus or forms have a hierarchical or aggregate composition.

[4] Newer formalisms include SGML and XML

4.2 State models

The use of state models to simulate computer system behaviour is widely practised Sommerville (2001). Within the user interface community, the same formalism can be found in early work describing menu-based interactions Wasserman (1985), direct manipulation systems Jacob (1985) and has latterly been applied to more contemporary GUI design Horrocks (1999); Carr (1997) as well as 'virtual reality' (VR) environments Jacob et al. (1999).

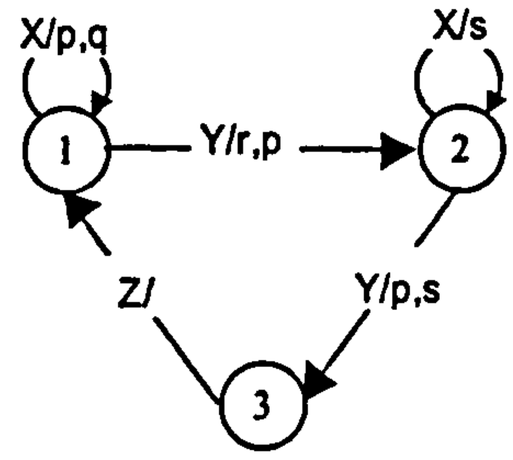


Figure 5 A simple state model Horrocks (1999)

State modelling plays an important role in the description of semantics in many aspects of model-based design (this is particularly obvious in specification of the behaviour of input devices and abstract components, see section 5.2). The mechanics employed for describing the 'state' of a user interface (or part of it) are varied, however the most commonly methods are by *attribute* or *Statechart* or *Petri Net*. Informally, the state of a particular part of a system is marked by the value of some attribute; in the UAN notation for example, the states have an informal representation such as 'selected = file' Hartson et al. (1990). Object-oriented methods extend the attribute-based marking of states by encapsulating one or many of such markings in a single object. The Teallach task model Griffiths et al. (2001) uses just such a mechanism in its task model.

Explicit markings of states and transitions between them are found in the form of statecharts Harel (1987) and, in a more sophisticated form, petri nets Peterson (1981). For the sake of brevity both are described informally here. Basic state models are directed graphs of nodes (representing individual states) connected by arcs (representing transitions between states), see Figure 5. When a state is entered, some operation on data internal to the system may occur. Transitions connected to the state contain conditions or rules (such as '*value* > 10') relating to the internal system, such that when satisfied, the transition's action is said to 'fire' (possibly some modification of the internal system) and a new state is entered. In Figure 5, three states are depicted connected by transitions *X*, *Y* and *Z* that fire zero or more actions *r*, *p*, *q* and *s*. This basic mechanism serves as the basis for many variations applied to suit the domain being modelled. Harel (1987) extends this by providing a graphical formalism for embedded states (states

within states) and concurrent state modelling. In Trætteberg's direct manipulation model Trætteberg (1998), transition rules include rules that relate to UI component events and conditions. Jacob et al. (1999) use discrete transitions between states to control continuous (but transitory) transitions that describe relationships between input devices and VR components.

4.3 Petri nets

A more expressive form of state modelling is found in Petri nets, which are capable of describing concurrent states of a system through the 'marking' of tokens within a network. The Petri net consists of one or more 'places' (similar to nodes, described above) which may hold zero or more tokens (these may carry values that are specific to the domain being modelled). Transitions represent controlled pathways through which tokens may be consumed and generated between one place to another; places and transitions are connected via arcs. Each arc may specify a 'weight' - in the case of input arcs this means that a transition cannot fire until the number of tokens from an input place matches the weight across the arc. The number of tokens output from a transition to a place is determined by the weight of the arc connecting it, by default this is one. Finally, an arc may also be inhibitory, inverting the logic of a normal arc such that an empty place connected to a transition via an inhibitor causes it to fire.

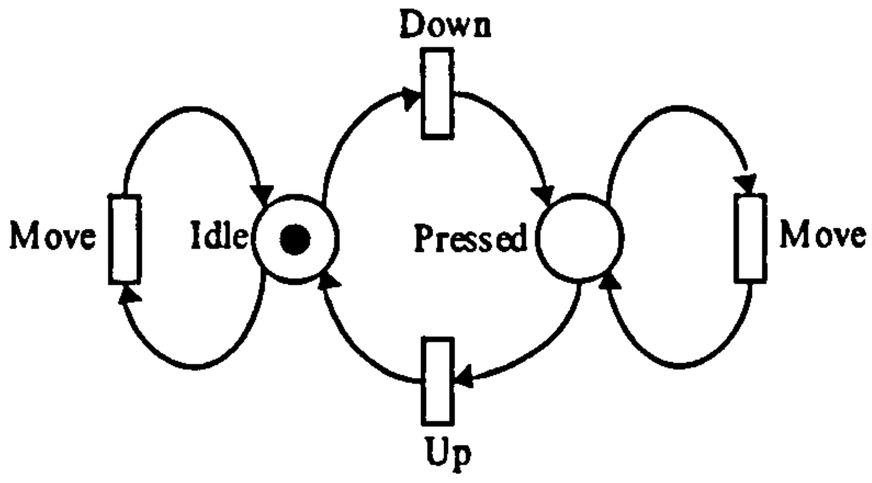


Figure 6 A simple mouse model Accot et al. (1996)

Further refinements to this basic model have been applied to suit model-based user interface abstractions as required. For example, the ICO formalism Bastide et al. (1998) adds a precondition on transition firing in the form of a logical expression operating on the passed tokens – the Boolean result determines the execution of transition. In addition, tokens may represent input from the user originating in special 'event places' (on the other hand, any place in the ICO Petri net may use tokens to render information to the display). Other refinements of the transition model can be found in Accot et al. (1996) in their mapping of physical device actions to 'interaction level' operations (for example 'mouse button down' to 'click' actions) by adding device events to transition rules; in Figure 6 a petri-net describes basic mouse behaviour. Massink et al. (1999) refine both place and transition concepts by quantifying and typing valid tokens for each place and specialising discrete and continuous transition types. Discrete transitions have a fire condition, a delay time, a firing action (a function that produces the output token type and

value). Continuous transitions accept only tokens of type *real*, the firing action of the transition specifies differential equations linking the values of both input and output tokens on a continuous basis.

5. Computable objects and prototyping tools

Formally modelling states and message passing allows designers to simulate and reason about the *high-level* abstractions of a user interface solution without either having to a) 'mould' their design ideas into a particular GUI framework; or b) commit substantial software engineering resources in the process of doing so. For these reasons, these computational tools represent a 'half way house' between the implementation specific prototyping techniques and formal methods; when used to underpin model-based user interface design tools they inherit both the advantages of formal abstractions and the communicability of prototyping strategies. Few of the computable objects discussed here are used alone and in their original form; instead they are extended and integrated into a larger toolkit to provide support for a range of model-based user interface abstractions.

5.1 User interface abstractions

Having examined the different approaches to the engineering of user interfaces, some explicitly built on models (some not), a review of the specification of the five high-level models supported by these methods follows. In each of the abstractions, the specification strategies are inspected with regard to the particular interaction paradigm they support and the tools used to model them.

Approaches \ *Abstraction level*	I/O Devices	Components	Dialogue	Domain	Tasks
Storyboarding/ Prototyping tools		✓	✓		
SWEng APIs/GUI tools		✓	✓	✓	
Computational models/tools	✓	✓	✓	✓	✓
Formal methods		✓	✓	✓	✓

Table 1 Design approaches and abstractions

A summary of the model categories and their support is illustrated in table Table 1. Computational models and formal methods span furthest across the abstractions, with computational models providing the most support since formal methods tend to 'hide' the structural and procedural elements in a design. It is important to stress here that these categories have been chosen as a general outline for discussion and should not be considered as rigid delineation between model-based technologies since there are often cases where abstractions over-lap.

5.2 Device modelling

It is not surprising to find that the granularity at which physical input and output devices are modelled in the literature varies in accordance with the impact that these devices have on the overall interaction design. The range extends from highly detailed simulations of input device data transformations Massink et al. (1999); Jacob et al. (1999) to ostensibly defer them in favour of a higher level of application abstraction Markopoulos et al. (1999). This thesis only considers a limited range of input and output models (the mouse, keyboard and graphical display) although it is recognised that many other modes of interaction including audio, haptics and gesture based interfaces exist Pentland (2000).

Sub-component specification of graphical output to the user is expressed in a language that can be translated to the particular function calls of the target API. An example of this can be found in the conversion of the 'display part' of the *abstract user interface* (AUI) in Schneider and Cordy (2001) in which a generic library of graphical functions provides an abstraction for the specific API calls used in the 'concrete' implementation. An informal specification of abstract graphics functions can also be found in Du and England (2001), an extension of the UAN language.

The handling of input data from physical devices such as the mouse and keyboard falls into two categories: discrete, event-based and continuous, data-flow models. This division highlights the demands that new interface technologies, such as virtual reality systems, make on the interface designer. Such environments often require the interpretation of parallel and continuous modes of interaction Jacob et al. (1999). In both discrete and continuous models, the data 'piped' from the device into the logical system are transformed into a format that is compatible with the interaction model. For example, a stream of delta values x and y are transformed into a parameterised 'mouse move' event on a virtual pointer displayed on the screen. Whilst continuous interaction models make explicit how these transformations occur, the discrete models most frequently use a direct and automatic translation, assumed to be provided by the services of the underlying operating system.

Discrete event notations, such as those used by the UAN Hartson et al. (1990) specify generic 'desktop' interactions:

(1) ~ [file_icon] M

(2) ~ [x,y] * ~ [x',y']

(3) M

Figure 7 UAN example Hartson et al. (1990)

Here, the ~ symbol specifies movement of the mouse pointer into the context of an object, in this case a file icon [file_icon], see Figure 7. A button down action ($\vee$) on the mouse (M) initiates, in this example, a drag sequence. This sequence is described as zero or many (*) arbitrary changes in the x and y position of the mouse, ending in new positions x' and y'. The 'move file icon' action concludes with the mouse button being released (M$\wedge$). A statechart-based specification at this level of abstraction can be found in Trætteberg (1998); the direct mapping of device inputs to an interaction model using Petri nets is demonstrated in Bastide and Palanque (1999). There is no clear distinction between physical actions enacted upon the input device and events represented at the user interface in this type of event model. To illustrate this point, consider a direct manipulation system that uses input from a mouse device to direct the motion of a graphical pointer on the screen. The pushing and releasing actions on a mouse button can be considered a 'device level event'. A *double-click* action is a symbolic, device-independent or 'logical interaction level' event focused at the position of the pointer object on the screen. Accot et al. (1996) use an extended Petri net notation to express the transformation of device events into interaction level events:

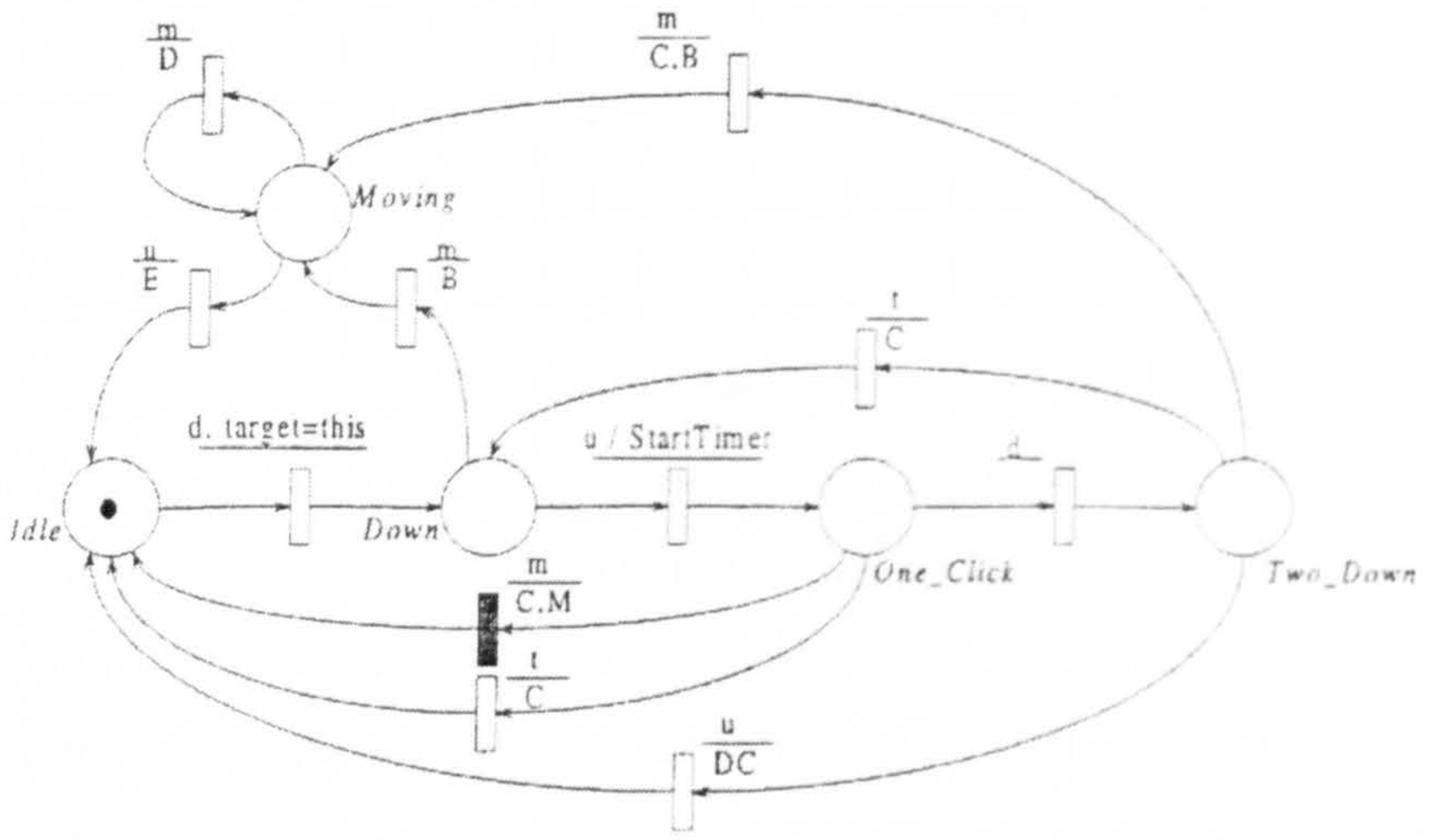


Figure 8 Device and interaction-level event modelling Accot et al. (1996)

In this example (see Figure 8), a 'pointer drag' event can be described as two routes through the network depending on the timing that distinguishes a *drag* operation from a 'wobbly' *double-click*. Places in the network depict the state of the device whilst transitions are labelled with the device-level tokens that are consumed and the high-level interaction events are produced as a result. Device level events are expressed in lower case (d = mouse down, u = mouse up, m = mouse move, and a special system service, t = timeout). Logical level interaction events are specified in upper case; C = 'click', M = 'pointer movement', B = 'begin drag', D = 'drag', E = 'end drag' and DC = 'double click'. This work is continued in the modelling of the keyboard actions in Accot et al. (1998).

The notion of an input device 'stream' is expressed as processes in Sage and Johnson (1997a) LOTOS simulation of a simple game. Similarly, 'queues' are used to post device data polled from the system by the CUI services in Schneider and Cordy (2001) to the higher level AUI model. For virtual environment modellers, it is useful to regard input devices (even the keyboard) as a temporary source of continuous data[5] Jacob et al. (1999). This allows relationships between continuous values, such as the velocity of the mouse movement and the motion of an object in virtual space, to be expressed. In-coming device data in Jacob's model flow through a state chart in which transitions contain mathematical functions that relate the mouse movement to interaction objects displayed on the screen.

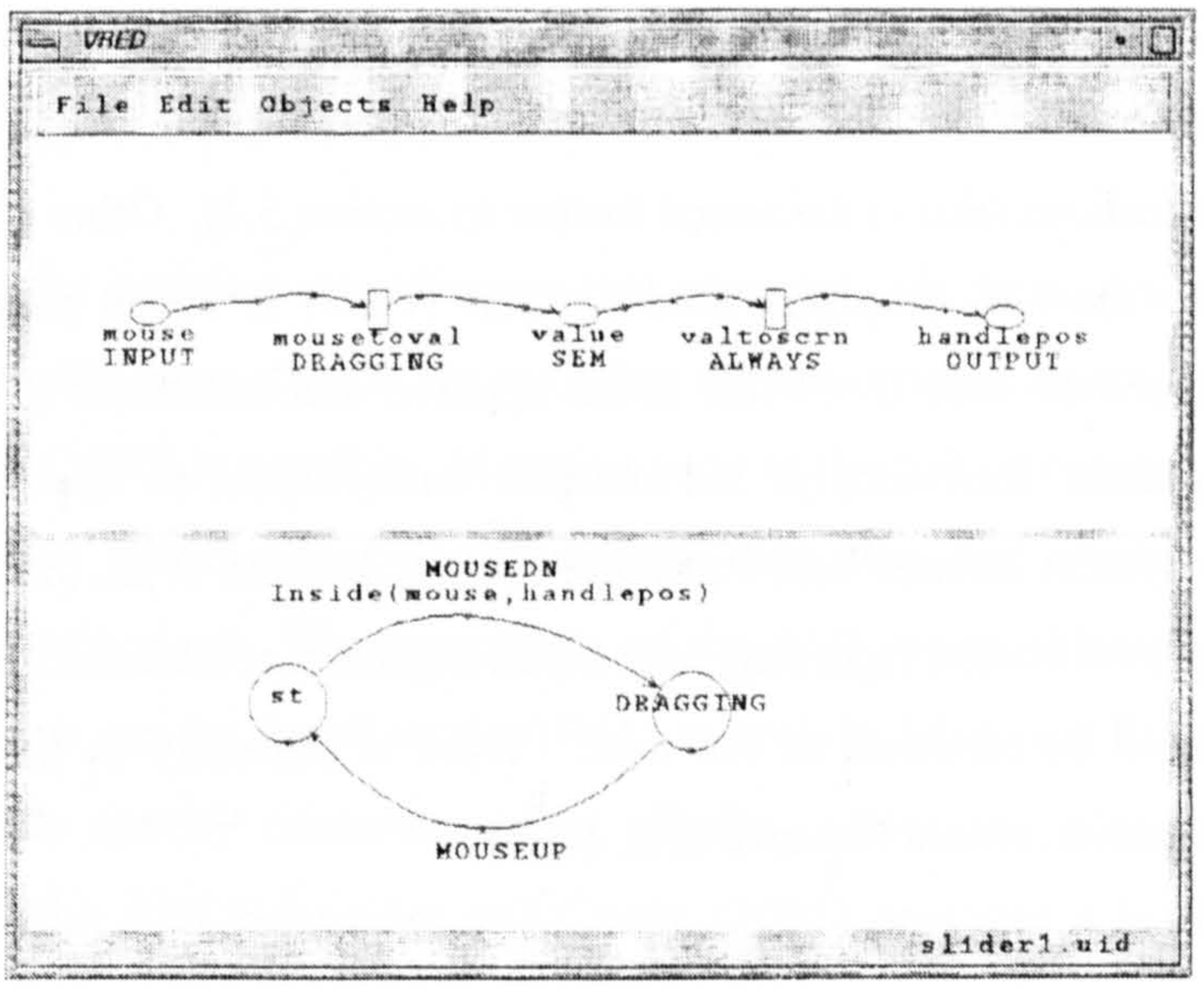


Figure 9 Mouse and scroll bar interaction Jacob et al. (1999)

Figure 9 Jacob et al. (1999) shows a simple relationship between mouse movement data, the position of the pointer and a scroll bar. Similar approaches can be found in the *HyNet* system Massink et al. (1999). Here, a modified Petri Net model is applied to map mouse movement to relative movement in a virtual environment based on the continuous relationship between the pointer position and a 'zero motion' target (a two dimensional square) centred in the middle of the display.

In comparison with other model-based dimensions (such as component presentation or task specification), low-level user input and output models receive relatively little attention. With the exception of the treatment given to virtual reality interactions, it would seem that much of the details at this level are deferred to either the prototyping tool or programmer.

---

[5] Although it is recognised that in implementation, this is of course digitised into discrete packets of data

5.3 Display component and dialogue modelling

In this section, a review of the architectures that describe the domain adapter, dialogue and logical interaction components of the Slinky model is given. The dialogue control is the intermediary that connects parts of the task or domain model with the presentation of GUI components. In some cases, this communication is a direct mapping, in others it is actualised via some abstracted proxy.

Direct styles of mapping between dialogue and component presentation are described in the UAN notation Hartson et al. (1990) and its derivatives Gray et al. (1994); Du and England (2001) . Here, user actions are mapped with feedback from the interface, the states of interface objects and remarks regarding functional core operation within a task-based framework. Temporal operators qualify relationships within the task and action specifications (this is discussed further in section 5.4). Other direct mappings between task and dialogue include the IOG graphical notation Carr (1997) in which pictorial representations of GUI components are combined with UAN-like event notation and statechart structures. Petri nets are used to describe the dynamic behaviour of an interface with respect to user actions on presentation components in the ICO system Bastide and Palanque (1999); Bastide et al. (1998). Tokens travelling around the network carry information regarding user input, objects that transmit and receive data from the functional core and data to be rendered to the user. Work is currently in progress to combine task descriptions with the ICO environment Navarre et al. (2001).

A special exception to this type of mapping between the domain and the presentation via a dialogue model can be found in CAD and VR environments. Such environments typically circumvent any dialogue intermediary with a direct call to the system core based primitive gestures using the mouse and keyboard. Jambon et al. (1999) present a modified version of the Arch model (called H4) that allows the domain adapter to directly render 2D/3D data structures modelled in the functional core to the user interface. A similar approach is taken in Jacob et al. (1999) in which input received from the mouse is transformed along a data path to provide appropriate data for 3D transformation matrices.

The use of a proxy as an intermediate step between the domain model and the user interface is used both at *design-time* where implementation details are deferred da Silva and Paton (2000); Markopoulos et al. (1999); Bodart et al. (1994)) as well as within *run-time* systems da Silva et al. (2000); Schneider and Cordy (2001); Schreiber (1994). These proxies provide potential solutions to the problem of mapping domain models to GUI components Puerta and Eisenstein (1999) by:

- Differentiating the 'abstract user interface' (AUI) from the 'concrete user interface' (CUI)
- Exposing some of the domain model at the interaction level

By inserting an abstract user interface specification that provides only basic descriptions of typical GUI components, high level design-time support for mapping either software object models Jaaksi (1995) or task models Johnson et al. (1995) is possible. Mapping parts of the task or domain model to properties of the abstract interaction layer allows designers to explicitly flag how, and where, changes in the system state affected by the user take place. The complexity of the AUI varies; relatively simple and narrowly defined abstractions such as can be found in the UMLi model da Silva and Paton (2000) and Teallach Griffiths et al. (2001) only specify *container*, *input*, *display*, *editor* and *chooser* generic interactions. Other tools go further – the AUI Schneider and Cordy (2001) provides a lexicon of generic WIMP components, graphic rendering and input capture methods (these are subsequently translated into C++ code). The MIM interface model Puerta (1996) allows abstract component definition based on declaring attributes for each object.

Proxy-based mapping discussed so far has adopted a 'layered' view in which relations between the domain or task model are progressively transformed between independent, but interacting layers (mirroring to a greater or lesser extent the Arch model). However, the object-oriented paradigm offers an alternate framework in which interactors (introduced in chapter 2) capture these principles but within a network of communicating objects. Edmonds (1992) gives an account of the early evolution from the linear Seeheim abstraction to an 'interactor' based framework. Precursory models such as those proposed by Williams' 'communicating objects with surfaces' Williams (1992) and Took (1992) UMA model are some of the early examples of object-oriented architectures that separate the user interface from the function core. Contemporary interactor architectures reflect both this general trend towards separation and also the adoption of object-orientated software engineering concepts. Duke and Harrison (1993) give a broad definition of an interactor object:

> "The notion of an interaction object is one of an independent entity with a local state that can engage in events within its environment, possibly resulting in changes to the state. In this respect an interactor is much like a state-based process … or the notion of object that underpins the object-oriented metaphor."
> (page 2)

Hussey and Carrington (1997) introduce the notion of interactor as a mediator between the user and the system:

> "An interactor has a presentation (lexical) aspect which reflects the internal state of the application (syntax and semantics), and which mediates between the underlying application and the user." (page 2)

Interactors are arranged in communicating networks that move user input toward the functional core and return the state of the application in the opposite direction to be displayed to the user; an overview is provided by Markopoulos (2001), see Figure 10.

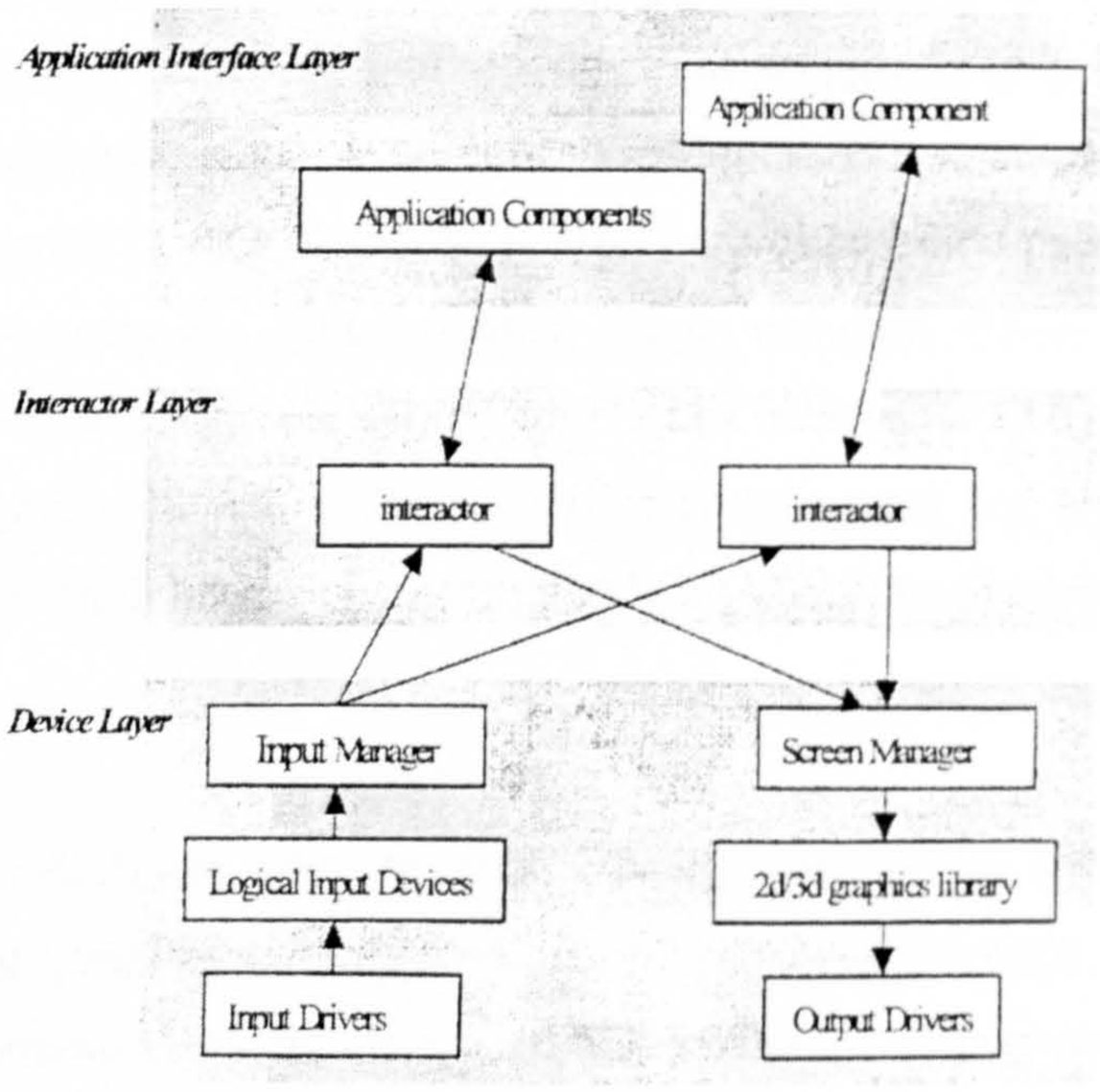
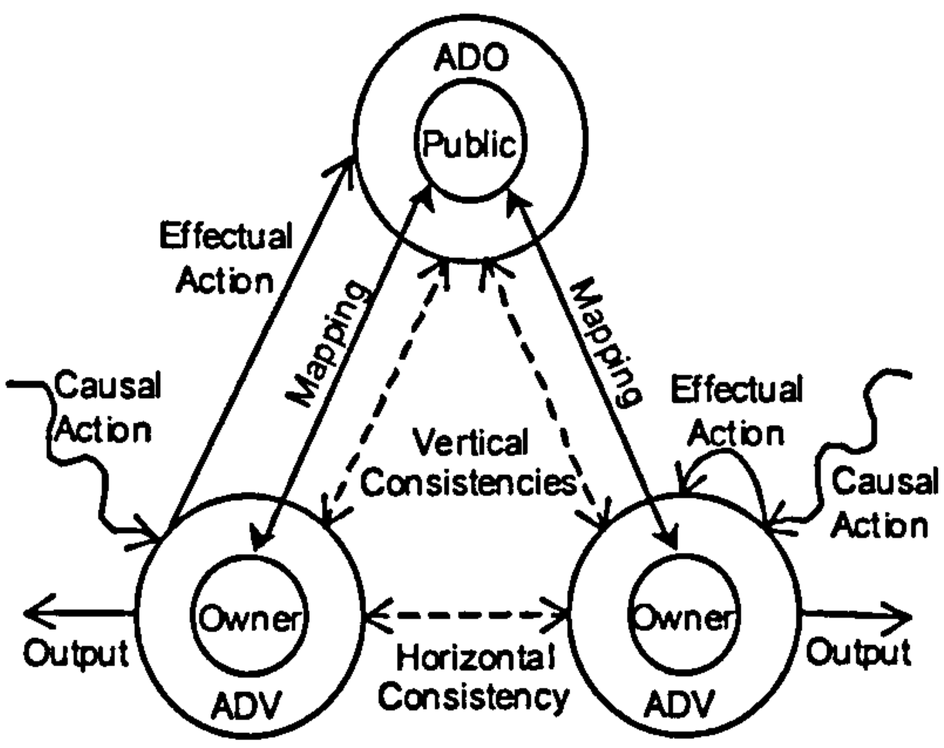
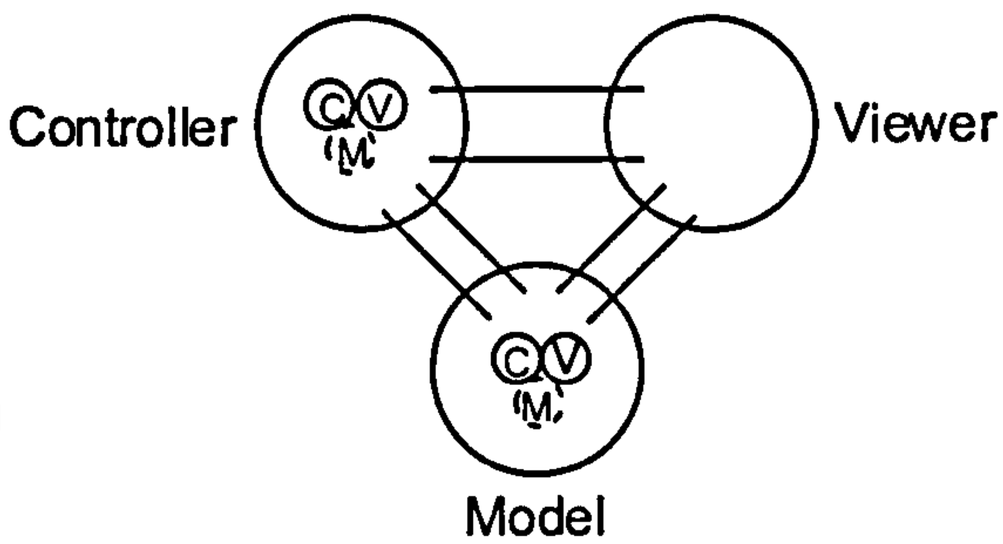


Figure 10 Interactor overview Markopoulos (2001)

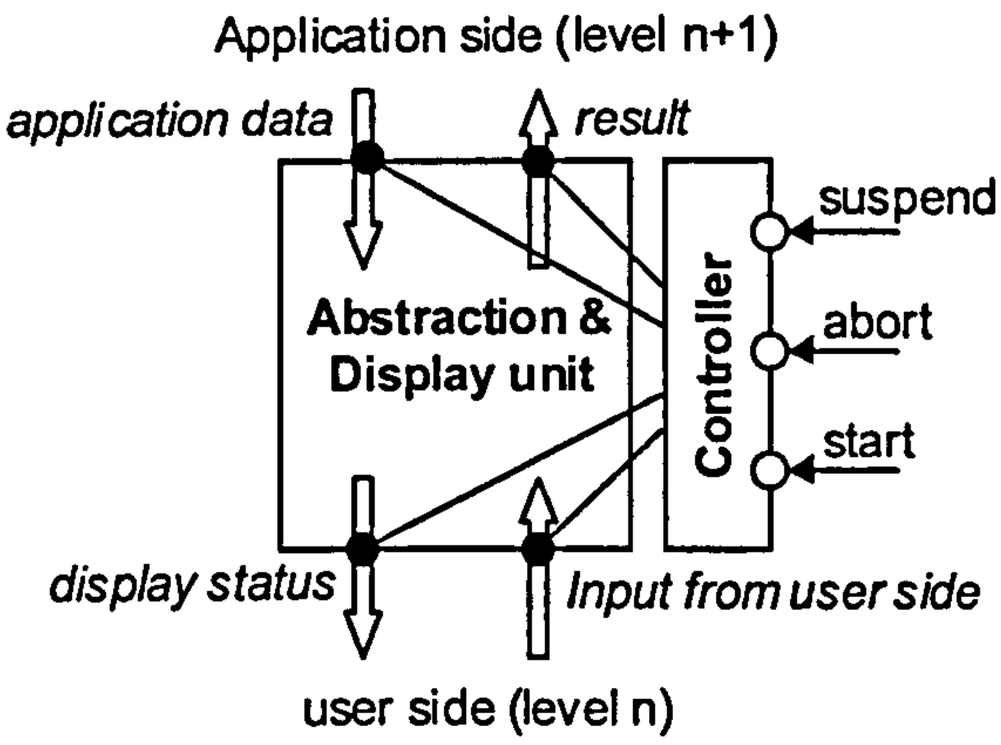
Whilst these general properties hold for all interactor architectures, variations in their conception can be found in the literature. To illustrate this point, four models are briefly examined here: MVC Adams (1988), PAC Coutaz et al. (1995), ADV/ADO Alencar et al. (1995) and ADC Markopoulos (1995), see Figure 11.



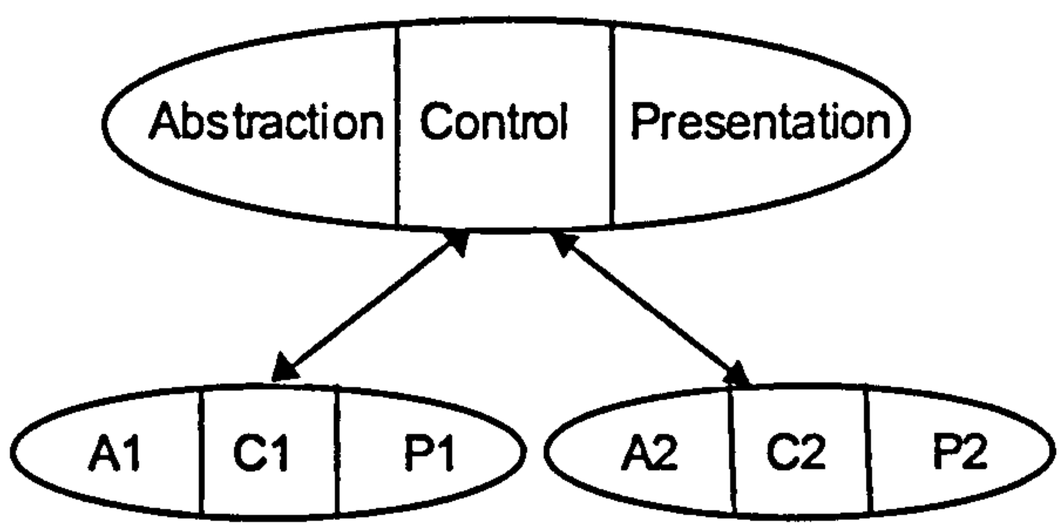
ADV/ADO model (Alencar, 1995)



MVC model (Hussey and Carrington, 1996)



ADC model (Markopoulos, 1995)



PAC model (Hussey and Carrington, 1996)

Figure 11 Interactor abstractions

Common to all formats is the central *abstraction* that maintains a description of some part of the system's state. In the MVC, this is referred to as the 'model', the 'abstract data object' makes up this part in the ADV/ADO framework whilst PAC and ADC simply refer to this as the 'abstraction'. Within the MVC and ADV/ADO frameworks, abstractions may contain child abstractions or aggregates, whilst PAC and ADC maintain a hierarchical organisation[6]. The management of input and output local to an interactor is the responsibility of the *controller* in the case of MVC, PAC and ADC and 'morphisms' in ADV/ADO. The primary role of the controller in MVC is to manage input (which may include data translation appropriate for the abstraction). PAC controllers facilitate both input and output data from the presentation to the abstraction as well as communicating changes to other, connected interactors. Morphisms or 'mappings' and temporal rules embedded in ADV/ADO interactors express constraints and

[6] Although ADC interactors can be combined, see Markopoulos (1997).

relationships between states and actions made accessible by each interactor. The ADC abstraction continuously processes input and output, mediated by its associated controller that can enable, suspend, resume or disable gates.

User input enters the abstraction via the controller parts in MVC, whilst it is combined with the presentation (or view) part in PAC (although this must be passed through the controller before changes in the abstraction may take place). 'Causal actions' are the equivalent in ADV/ADO[7] in which an explicit distinction between user and system input (referred to as 'effectual actions') is made. Similarly, the ADC model provides user input 'gates' on its display side whilst other interactor or system related input is channelled through gates on the abstraction side. System-side input affecting the state of the interactor in both the MVC and PAC models are assumed as method calls from within the abstraction.

Changes of the system's state are made visible to the user via mappings between the abstraction and the view parts. Output to the user is displayed in the *view* part of the MVC, PAC, ADV/ADO and ADC architectures described as *view*, *presentation*, *ADV* and *dout* respectively. In MVC, each view (and its associated controller) is dependent on an abstraction that broadcasts changes of its internal state to all associated views and controllers. Similar changes are effected in the PAC model through the controller which maps abstraction to presentation as well as propagating appropriate events to its parent controller (if it has one) and any dependent children. Mappings between the viewable ADV object and an associated ADO object, expressed in the *interconnection* section of the ADV specification, provide a similar service in the ADV/ADO model.

There are, in fact, many other variations to be found in the literature which are either based on existing frameworks (such as MVC) or declare an interactor-like basis for design. These include the formal specification of widgets Hussey and Carrington (1998) and framework modifications, such as 'PAC-star' for networks Calvary et al. (1997). A number of windowing environments and GUI toolkits have also been developed based on interactor frameworks including *SmallTalk* Adams (1988), *Teallach* da Silva et al. (2000), *GRALPLA* Cabrera et al. (1999), *Vista* Brown et al. (1998), *Clockworks* Graham et al. (1996), and SIRIUS Windsor (1990).

5.4 Domain and task modelling

Two regularly modelled entities that 'drive' the observable performance at the human-computer interface are: 1) domain models representing some aspect of the functional core and; 2) some description of human

behaviour, such as a task model expressing actions that the user takes to achieve some goal. The scope of this discussion must be limited in both areas since a review of all software engineering methods and psychological models that could potentially contribute to this area is far beyond the remit of this work. For this reason, the literature reviewed here concentrates on the predominant works conducted within the model-based user interface design community: data-oriented domains and task modelling. It is worth noting that the level of abstraction found in 'domain models' varies within the literature, the continuum of which ranges from simple, high-level entity enumeration to complex software engineering data models Griffiths et al. (2001). Both task and domain models may be usefully considered side by side since they can be viewed as two co-operating systems engaged in a shared activity.

Forbrig (1999) describes the relationship between tasks, the problem domain and an interaction model in Figure 12. These models, in Forbrig's view, have considerable overlap, sharing common concepts (particularly artefacts or objects, which are commonly referred to by each). Here, the transformation from an extant interactive work model to an envisaged new system is a process in which task objects are reassigned with new tools and agents (either human or computer) engaging in roles that manipulate the state of the shared problem space.

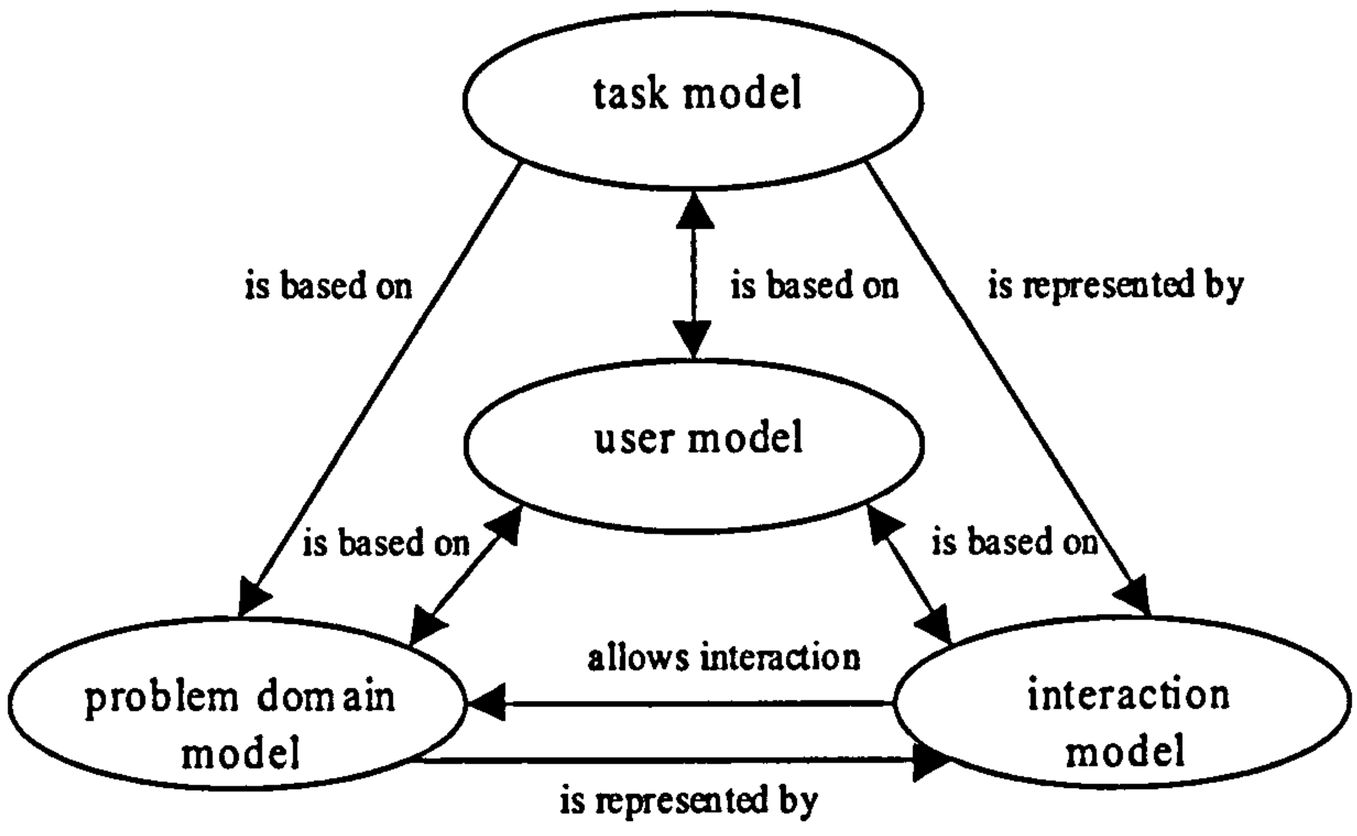


Figure 12 Forbrig's model

The user model supplies an axis of refinement to the task, problem domain and interaction model in that it qualifies a user's engagement with the tasks relating to the problem domain, their access to data and particular interaction styles and modalities. This view is useful when considering the work of organisations in which problems are solved within groups, where individuals have particular roles and must work co-operatively. Tools such as Euterpe van der Veer and van Welie (1999) make this

[7] The ADV object is an extension of the basic ADO in the ADV/ADO model

distinction explicit by declaring *agents* who play *roles* that are responsible for *tasks*. Similar effects are achieved in Teallach Griffiths et al. (2001) and MOBI-D Puerta and Eisenstein (1999) by assigning users to 'user types' which can then be associated with particular tasks. This style of user modelling is relatively simple – a more sophisticated research community exists in its own right, see Kay (1999). Whilst the simple types of user model found in the MB-UID certainly enhance the expressiveness of these tools, the attention given to them is frequently subordinate to the focus on the problem and task domains and so are not considered further here.

The hierarchical structuring of tasks is evident in some form in many of the model-based toolkits, including ADEPT Johnson et al. (1995), EUTERPE van der Veer and van Welie (1999) and CTTE Breedvelt-Schouten et al. (1997). A number of task models include state conditions of both the task nodes themselves and also those of associated problem domain objects. The utility of this is the means of determining whether a task can be executed as well as its expected state after task completion, see IMAD-star Rodriguez and Scapin (1997) and Teallach, Griffiths et al. (2001). Temporal relationships between task units include serialisation, concurrency and iteration, examples can be found in UAN Hartson et al. (1990) and CTTE Breedvelt-Schouten et al. (1997).

Mappings between the task and problem domains have been demonstrated using various relational models. TADEUS Stary et al. (1997) uses Object Relation Diagrams (ORDs) to describe relationships between task hierarchies and data classes. An ORD relationship may be an informal descriptor such as "request a ticket", a temporal constraint, or the specialisation and aggregation of classes. The MIMIC modelling language which underlies the MECANO and MOBI-D toolkits Puerta et al. (1994) Puerta (1996) allows the specification of arbitrary *relations* (such as an *is-a* relation), *attributes* and *conditions* (a tuple of initial, pre and post-conditional states) for each domain object. The BOSS tool Schreiber (1994) uses conventional software engineering constructs (such as records, trees, tables and lists) to specify the application programmer's interface to the functional core. Functions within the API are defined in terms of predefined data types for input and output and may include optional pre-conditional constraints on the input types before execution can occur. Tasks are described through the recursive instantiation of HIT objects, which can occur in parallel, by selection, or in sequence. Each HIT can potentially act as an input/output portal for the user and contain domain data and functional calls pertinent to the task.

The TRIDENT tool Bodart et al. (1994) uses an entity relationship attribute model (ERA) to specify database schemas and an activity chaining graph (ACG) to describe the functional data processing requirements for terminal nodes of a task model. Data input (from the user) flows to functional calls to the system, which may result in either error messages, or the retrieval of the appropriate ERA schema entities for further processing. Considerable support for application and data oriented domain modelling

is provided by Teallach Griffiths et al. (2001) in which standard database data types and functionality and Java API classes are modelled using the ODMG database standard Cattell et al. (1997). The task model in Teallach includes the specification of task state objects containing attributes derived from queries made to the database schema in the domain model or auxiliary method calls to a specified Java object.

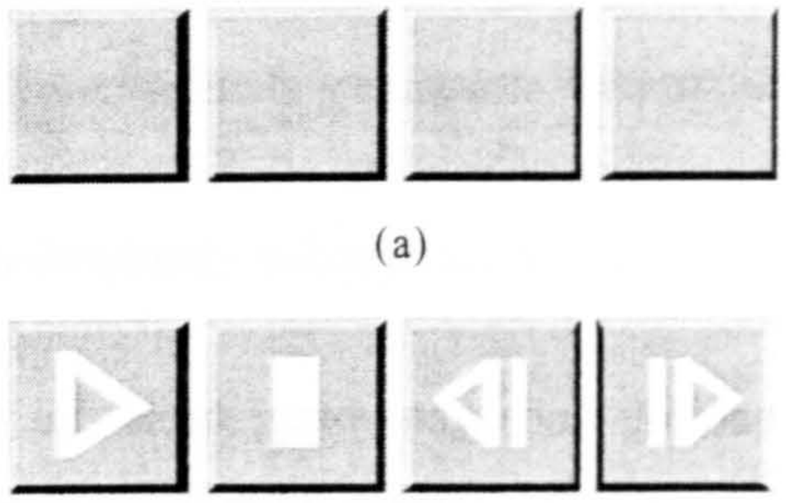
6. Discussion

In contrast to traditional storyboarding and prototyping tools, the expanding range of model-based notations and toolkits explicitly expresses significantly more detail regarding not just the appearance and visible behaviour of an interactive system but also other, related design views that must be integrated in the development of the software product. Formal methods practitioners have rigorously defined and reasoned about the behaviour of widely used interactive components such as buttons, lists and scrollbars. Computational objects allow designers to simulate, or with the appropriate tools, rapidly prototype user interfaces without recourse to committing to any particular implementation.

Model-based software support in these areas potentially enhances interface design through combining (some) aspects of formal abstraction and computational simulation into an integrated model building, simulation and prototype generation environment. As this field of research matures, an agreement on the interrelationships between the five components of the Arch model is beginning to emerge. Arguably, some of the most integrated and broadly based toolkits in this respect are *MOBI-D* and *Teallach*. In both cases, an underlying meta-language (MIMIC and ODMG respectively) provides a uniform means of integrating task, user, problem domain and interaction models within their respective scopes. These substantial achievements are not unqualified, universal solutions to all possible interactive system designs however; both are restricted to the ubiquitous desktop based interaction style and *Teallach* is predominantly designed for database applications. Narrowly focused frameworks are inevitable in this endeavour since integrating the Arch model alone is a non-trivial problem without adding the additional complexity of user and task modelling.

In addition to the existing complexity of this problem, the model-based interface design community must also face up to new interface technologies. The impact of new forms of computational devices, interfaces and human-computer contexts extends beyond the need to formally reason and simulate the behaviour of the technology itself. At present, it is useful to be able to map tasks *x*,*y* and *z* to button *a*, text field *b* and window *c* since it appears to describe (in a superficial way) a mapping between the user's formulation of her world of work and the computer's internal model of the objects that are used to achieve her goals. However, it is important to recognise that windows, icons, menus and pointers are not of themselves

directly a part of the user's or the computer's representation of the problem domain. To illustrate this point, consider the following arrangement of buttons:



(a)

(b)

Figure 13 Button example

Considering (a) we have no clue as to what function these buttons represent, but the affordance of pushing to indicate some required action is universally understood[8]. Adding symbols (familiar to at least those living in western society) to the buttons enhances the understanding of their function. There are two things to be noted here: firstly, buttons in Figure 13 are *images* of buttons rather than actual buttons and secondly, the symbols in (b) engage in the viewer a conceptual model that may be applied to understanding of multiple problem domains. Buttons may take on many different appearances both at the human-computer interface and on other machines; their physical operation and feedback to the user vary as well – some may be physically depressed whilst others light up or generate a sound to indicate operation. So in this respect, the concept of a button is not entirely dependent on its appearance and operation. In addition, the composition and symbolism used in (b) indicates a conceptual model that goes beyond the idea of four buttons arranged horizontally. A shared cultural understanding employed by both the designers of the device and its end user suggests that the functional operation of this device includes starting a process, stopping it, and jumping backwards or forwards according to pre-defined units of measurement.

Of course, controls like those in Figure 13 are frequently used on a variety of media players including cassette, CD, DVD and MP3 players. In each of these cases, the underlying principle of moving through a process (a serialised stream of media) by 'playing' it or navigating between sections of it remains constant whilst the task and the domain models may change. Users may wish to skip through 'tracks' on a cassette, CD or MP3 list whilst the nature of DVD denotes chapters in a story. A 'track' is an entirely abstract entity for the user since a 'track' has no meaningful, physically discernible characteristics on modern media objects such as a CD or computer hard disk. From a device designer's point of view however, a 'track' ultimately translates to a logical address space on the CD media or a file pointer.

In this sense then, the design of a user interface is more than simply the sum of its components – it includes a shared model that serves to partially describe the characteristics of the problem domain and the user's understanding of the world – a metaphor. At present, the model-based community has largely ignored this mediating mapping because of its reliance on the 'implicitly' understood and well-established desktop GUI Crowle and Hole (2001). Vanderdonckt and Berquin (1999) present a 'metaphorical structurer' that parameterises the presentation of familiar graphical objects such that they can be mapped to system data. This mapping allows data to be rendered in a variety of ways (in one of their examples, various presentations of a clock is shown). Whilst useful, this approach is limited since it only describes one kind of mapping: domain to presentation and does not consider other aspects of a metaphor such as object actions and behaviours. The modelling of virtual actions, such as the 'double click' found in Bastide and Palanque (1999) and Trætteberg (1998) referent model are two of the few that make explicit distinctions between the underlying concepts of the desktop metaphor and the execution of device actions. However, these are views limited to existing, WIMP-based components descriptions of interaction.

[8] This is true for the vast majority of users familiar with modern computers.

7. Mandate for a new framework

Five abstractions for model-based user interface design have been identified in this chapter along with examples of the application of a variety of tools and notations to each layer. A summary of the findings of this chapter is provided below:

Methods		Advantages	Dis-advantages	Abstraction support
	Storyboarding Prototyping tools	· UI designs communicated easily to end users · Parts of prototype testable by users	·'Throw-away' product - has little use beyond very early design stages	CP DG
	SWEng API GUI tools	· Development with target UI · Some rapid GUI tools available	·High development costs ·Early commitment to technology	CP DG DM
	Computational models and tools	· Abstract and concrete design views · Solutions communicable and testable with users	· Many design views; not all integrated · Tools limited to WIMP solutions	IO CP DG DM TK
	Formal methods	· High level of abstraction/reasoning for solutions · No technology commitment	· Difficult to use · Low communicability of design ideas to other project stake-holders	CP DG DM TK

IO = Input/Output devices CP = UI Components DG = Dialogue control

DM = Domain model TK = Task model

Figure 14 Abstraction summary

Whilst all the methods summarised above (see Figure 14) have at least one merit that is desirable as a potential feature for a UI design framework that includes a metaphor model, it is the undesirable elements that lead to a conclusion regarding the approach to adopt. The commitment to both significant development time and a particular platform also makes the software API approach undesirable and, in a sense, adopting this approach more or less ignores the MB-UID argument altogether. At opposite ends of the scale, storyboarding/prototyping and formal methods will be for the most part not considered further, for similar reasons: the former lacks engineering 'power' whilst the latter's expressive medium (mathematics) has low communicability to non-formalists. Some qualification is required here however: the interactor abstractions discussed in section 5 are echoed in other approaches (such as *Teallach*, *Clockworks* and *UMLi* – see section 3) and so influences from the formalists cannot and should not be totally disregarded. Computational models and tools have been shown to support a number of different design views, although at present this is still an emerging method that lacks coherency. This not

withstanding, computational models reviewed here provide the richest set of solutions across a range of abstractions. In the proceeding chapter, a synthesis of ideas from this approach is used to support an explicit metaphor model and is presented in the introduction of the 'Interface Specification Meta-Language' (ISML).

8. Conclusion

In this chapter, contemporary model-based design abstractions and the technologies used to support them have been examined. Models for input/output devices, presentation, dialogue, task and domain models were presented and the various mappings between each described. Current tools allow a limited degree of integration between the usability engineering and software design community through the mapping of task and problem domain models via the abstraction of common GUI objects. However, little work has been done in explicitly specifying the underlying conceptual metaphors these objects are frequently used to represent. An examination of current model-based user interface design methods suggests that computational models offer potential 'building blocks' with which to address this problem.

BLANK IN ORIGINAL

CHAPTER 4 The Interface Specification Meta-Language

1. Introduction

In this chapter the ISML framework and specification language is presented. A rationale for the ISML design is followed by a more detailed discussion of its constituent parts using a small-scale example for illustration. Specification issues arising from the example are then discussed.

2. ISML rationale

The concept underpinning the application of metaphor to user interface design in an attempt to aid understanding and interaction with the underlying system has already been outlined in chapter 1. ISML has been designed on the basis that the metaphor is an independent and partial mapping between a model of tasks understood by the user and the computational operations on the application domain by the underlying system. Arguably, the metaphor mechanism that acts as a bridge between the system and the user's world of work has only partial correspondences with each domain. Further, this mechanism in itself has no absolute manifestation with respect to its implemented appearance and operation at the user interface - the wide variation of the ubiquitous user interface desktop illustrates this point. The principal advantage of this 'de-coupling' of metaphor from other design concerns is that designers can consider the metaphor view explicitly and without the potential constraints imposed by implementation details.

An explicit metaphor model alone has limited benefits for the usability specialists and is of almost no use to other stakeholders of the interface design project. For such a model to become useful, a framework must be developed that provides a developmental pathway connecting both user-oriented models (such as task descriptions) and software architecture (interactor definitions) concerns. Additionally, it is desirable to maintain a level of abstraction that does not commit the metaphor model, or interactor design based on it, to a specific interface technology. The computational models reviewed in chapter 3 already suggest a number of useful modelling strategies including:

- Communicating objects
- State and constraint modelling
- Abstract-to-concrete mappings
- Event modelling

In developing the ISML framework it would be desirable to re-use these techniques since their application to various UI abstractions has already been demonstrated, see chapter 3. It has been argued above that the metaphor mechanism in a user interface may exist independently of implementation. For these reasons, a reasonable approach to introducing a metaphor model would be to create an additional abstraction of objects, behaviours and interactions that can then be mapped to extant MB-UID design views.

Importantly, to insure against untenable complexity, a limited range (and depth) of such design views must also be set. With this rationale in mind, the ISML framework was developed to integrate some of the existing model-based concepts discussed in chapters 2 and 3 and explicitly specify a user interface metaphor.

3. Framework overview

The ISML framework is composed of five parts, Figure 15 depicts the high-level relationships between each.

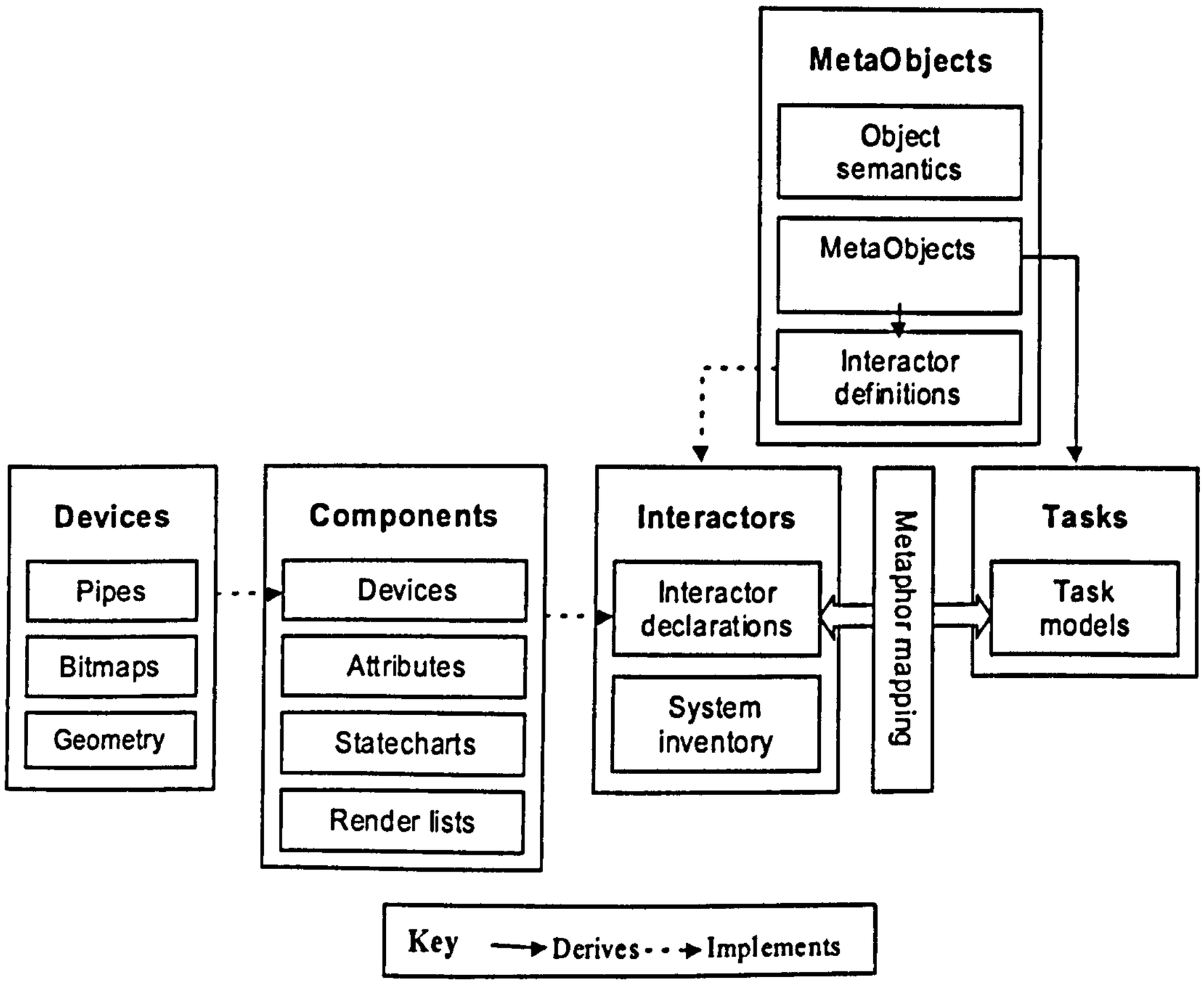


Figure 15 ISML overview

Many of the parts of the ISML framework can be found in other model-based specifications, but their particular abstraction and arrangement is unique to ISML. Notably, a common basis for specifying objects, their properties, actions and rules regarding behaviour is supported using the 'meta-object' layer (see Figure 15 ISML overview) and exploited in both the description of interactive solutions and task models. Since both the interactive solution and task model are derived from this base, it is possible to associate the execution of tasks with the manipulation of metaphorical objects, enacted using a specific interactor design. Below, a brief high-level overview of the five parts of ISML framework is explained; a high-level overview of the ISML framework is given by Crowle and Hole (2003).

Devices are simple abstractions of user interface input/output hardware used to model entities such as the mouse, keyboard and graphics adapter (see section 6.1). Logical abstractions of user input and output objects are specified as *components* (see section 6.2) which refer to previously declared devices for implementing their function (for example, a desktop mouse pointer is likely to refer to at least two devices for its change in position and display).

Meta-objects play a pivotal role in the definition of both interaction and task domains – objects declared here have attributes, states, constraints and communication mechanisms that serve both domains (see section 6.3). The definition of 'meta-object layer' forms the basis for the specification of the metaphor abstraction layer, its implementation (as 'interactors', see below) at the user interface, and the task model.

Interactor definitions use meta-objects as a basis for a specific design solution whilst *tasks* refer to them to describe how goals are achieved. Further refinement of interactors from their basic meta-object description is accomplished through a) the mapping of components to interactor abstractions and b) communication with the functional core. The intersection of meta-objects in use in both interactor and task models is described in the metaphor mapping sub-section. Finally, the system inventory specifies the starting state of the system in terms of instantiated interactors.

The following sections in this chapter deal with the five parts in turn, but begin with a general introduction to the technical nature of the ISML language. The graphical notation developed by Altova's XML Spy toolkit is used to specify the structure and grammar of ISML (specified in XML) for ease of reading.

4. Notations

ISML uses a Backus-Naur Form based grammar to specify the user interface, presented here in XML Bradley (1998). Briefly, the eXtensible Mark-up Language is a mark-up language[9] that specifies the well-formed description of data structures and their relationship with each other. Abstract data types may be declared which may contain attributes (of basic data types such as integers or strings) and other data types. These type definitions, along with the expected structure of the document defined using these types (which may be arbitrary) are held in a schema[10]. XML documents using this schema are said to be valid if the data within obey the syntactic rules of the schema.

An extension of the example provided in chapter 3 is provided here for simple illustration, but is not intended to serve as a complete tutorial to XML; interested readers should see Bradley (1998).

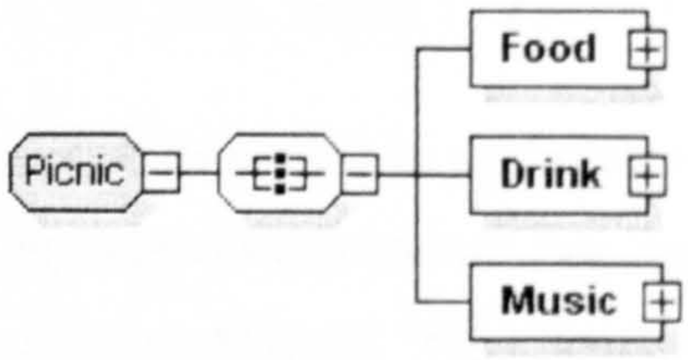


Figure 16 XML Example 1

In Figure 16, a *Picnic* (a group element type) must contain *all* (represented by the symbol) of its expected elements *Food*, *Drink* and *Music* which are of types **SandwichSelection**, **DrinkSelection** and **MusicSelection** respectively.

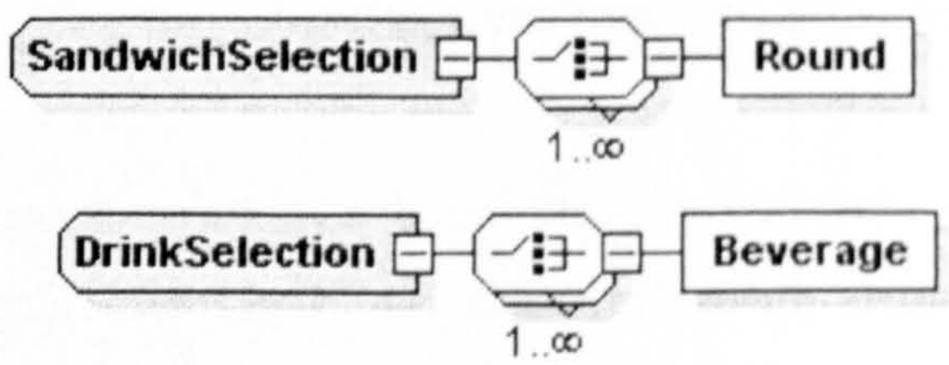


Figure 17 XML Example 2

---

[9] A subset of SGML (ISO, 1986. 8879:1986 Information Processing -- Text and Office Systems -- Standard Generalized Markup Language (SGML). International Organisation for Standardization)

[10] In fact there are a number of schema formats used in XML, here the XML Schema Definition (XSD) is used

Figure 17 describes complex types **SandwichSelection** and **DrinkSelection** – both of which may have a *choice*[11] of 1 or more elements (*rounds* and *beverages*) of types **Sandwich** and **Drink** respectively.

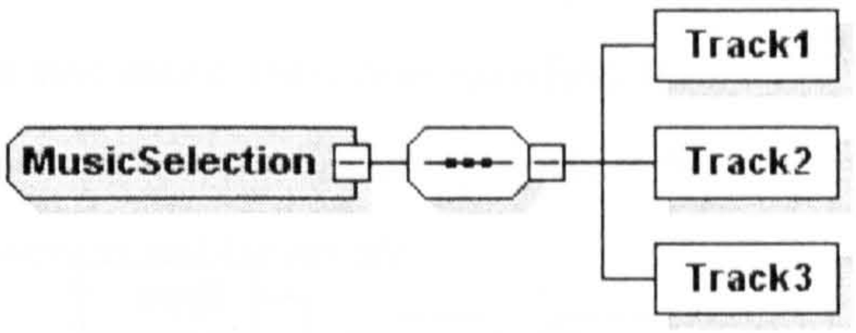


Figure 18 XML Example 3

Complex type **MusicSelection**, see Figure 18, must have one sequence of track elements track1, track2 and track3, which are of type **track**. The **sandwich**, **drink** and **track** types each contain attributes (not shown graphically here).

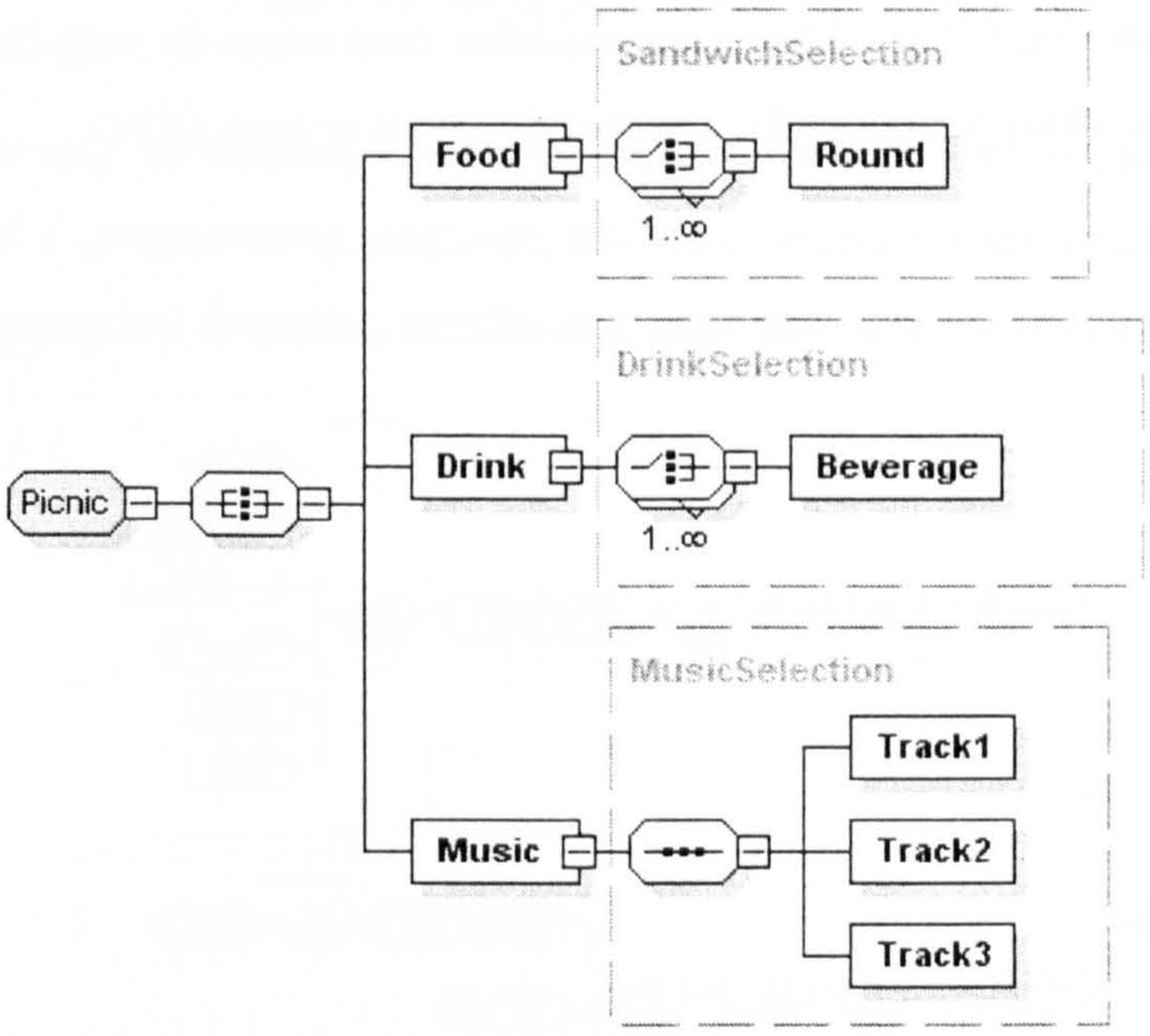


Figure 19 XML Example 4

The overall structure of this example is graphically represented in Figure 19, elements enclosed by a dashed border indicate complex types. In the following sections, ISML is described in more detail and for the sake of brevity, sections of the language once covered but which reappear in other parts of the framework will be omitted.

---

[11] This is graphically depicted by the connecting symbol between the boxed elements

5. ISML Basics

Within ISML a number of basic types are used throughout the framework – these include attribute, state model and procedural code sections.

5.1 Attributes

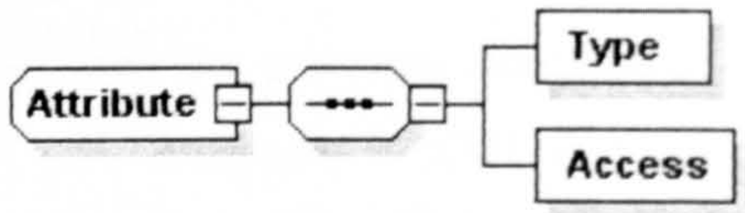


Figure 20 ISML attributes

Attributes have a required name, type and access. Basic types[12] of ISML attribute include common programming data types of bool, integer, float, string; attributes may also be of type *set*, referring to a special form of array (see section 6.3.5). The access qualifier determines its visibility to other objects in the environment and may be either readable (RO), writeable (WO) or both (RW).

5.2 State models

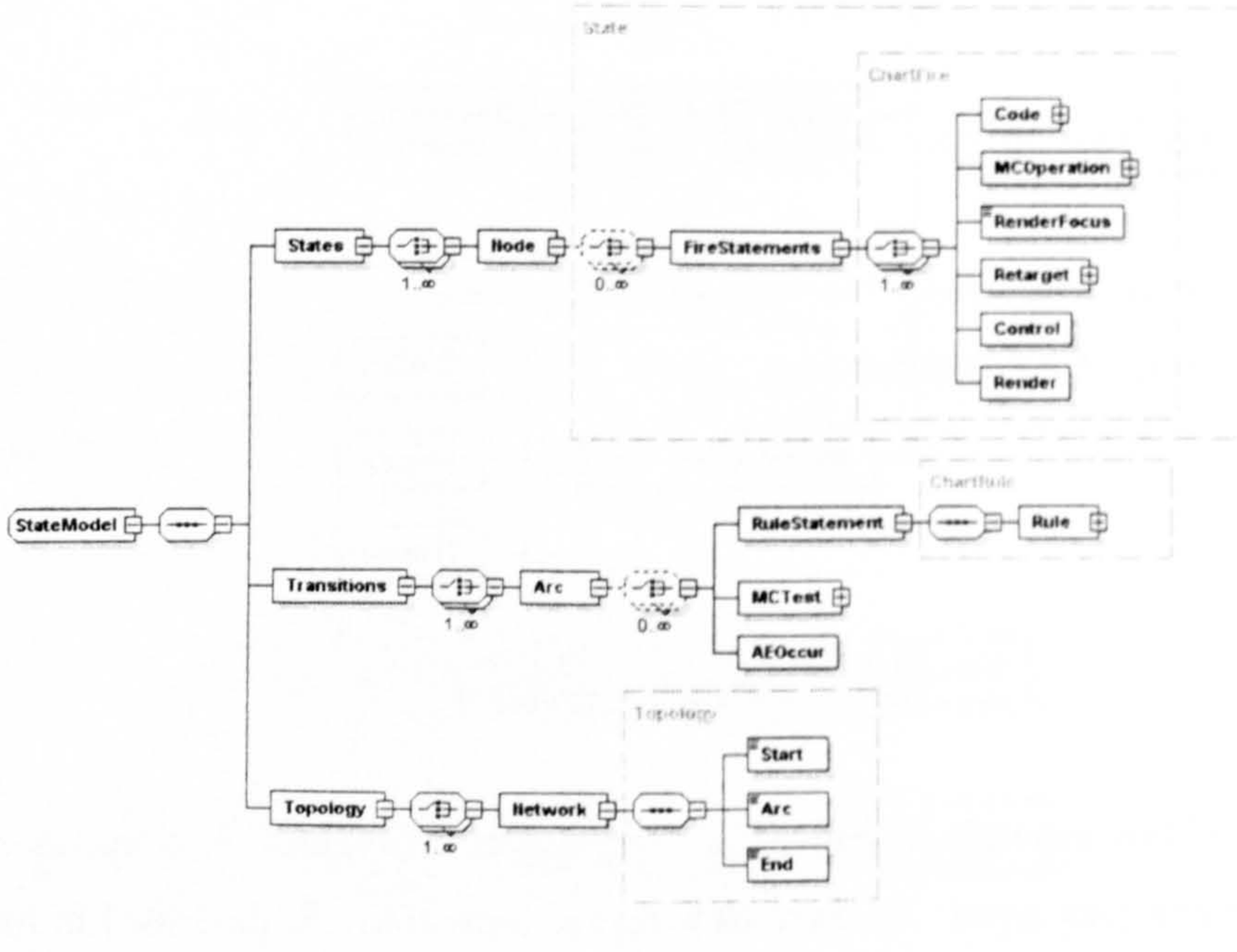


Figure 21 ISML state model

A basic, non-recursive state model is supported in ISML in which nodes and transitions are connected together by the topology. Each state may have one or many fire statements, executed when the model

enters the state and may be either procedural code, a render focus statement (see section 6.2), a 'mapping-constraint' operation (see section 6.3.5), a re-targeting expression (see section 6.5.2), a controller expression (see section 6.4.1) or a render function (see section 6.5.1). Every transition may have zero or more rules or MC tests (see section 6.3.6) which, when specified, must be satisfied for the transition to occur. In the case where a node has more than one satisfied transition available, priority is implicitly implied in the order in which they occur in the topology section for that node. The topology is simply a list of named start and end nodes connected by an arc.

5.3 Procedural code

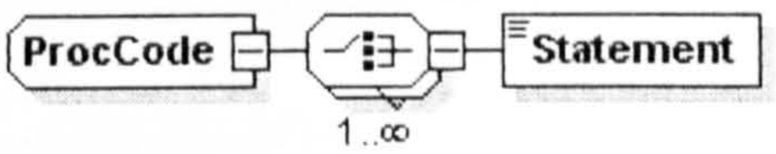


Figure 22 ISML procedural statements

Procedural expressions may be inserted at various points within an ISML specification; it is important to stress that ISML is not a programming language, but may contain programming language fragments for the expression of mathematical formulae, conditional logic tests and the execution of Action-Events (see section 6.3.2).

[12] At present, complex type support does not exist

6. ISML Parts

6.1 Devices part

Input and output devices in ISML are specified as an abstraction of their basic attributes and low-level software related functions. Devices are not abstractions of computer hardware, but instead provide hooks for low-level APIs such as Microsoft's DirectX and encapsulate I/O operations such as polling for input or the direct rendering of graphical primitives.

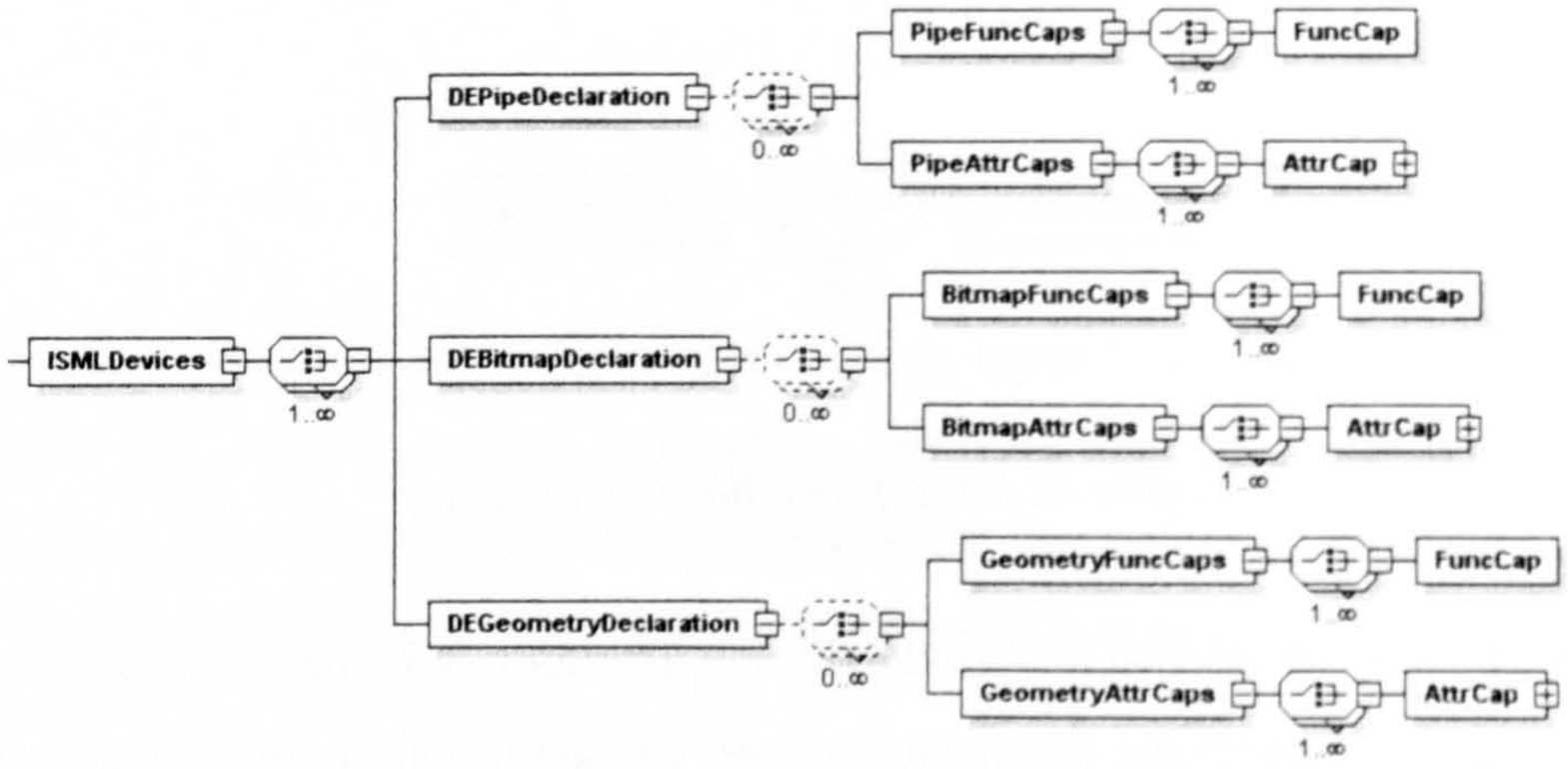


Figure 23 ISML Devices part

The framework for devices includes DEPipes, DEBitmaps, and DEGeometry which abstract device input streams, generic bitmap and geometry support[13] respectively. Presently, these definitions only provide a rudimentary lexicon for input/output devices but in principle could be extended in the future. Each device's definition is an expression of capability in terms of either the data it provides or the functions it exposes to the interactive system (similar to Microsoft's DirectX HAL). Available functional capabilities of each type of device (FuncCap) are chosen from those provided by the ISML lexicon and given a name: for example, the ability of a device to render a bitmap might be expressed as:

```
<BitmapFuncCaps>
        <FuncCap Caps="COPY_FROM_BITMAP" FuncName="Copy"/>
</BitmapFuncCaps>
```

In the above example, the generic function capable of copying bitmap data from a source bitmap object to a target (COPY_FROM_BITMAP from the ISML lexicon, see appendix F) is defined as Copy, although it could be a reference to a target API call, such as the Microsoft DirectX8 call 'IDirect3DDevice8::CopyRects(…)'. Attributes declared in a device are used as data holders for in-

[13] Such as the rendering of triangles in three dimensional space

coming and out-going data required by the functions and are used in the rendering of components (see section 6.2).

In the specification segment below, a simple mouse and bitmapped display device is defined:

```
<ISMLDevices>
        <DEPipeDeclaration Name="mouse">
                <PipeAttrCaps>
                        <AttrCap Name="button">
                                <Type Type="INTEGER"/>
                                <Access Type="RO"/>
                        </AttrCap>
                        <AttrCap Name="xChange">
                                <Type Type="INTEGER"/>
                                <Access Type="RO"/>
                        </AttrCap>
                        <AttrCap Name="yChange">
                                <Type Type="INTEGER"/>
                                <Access Type="RO"/>
                        </AttrCap>
                </PipeAttrCaps>
                <PipeFuncCaps>
                        <FuncCap Caps="PIPE_MOUSE" FuncName="GetMouseInfo"/>
                </PipeFuncCaps>
        </DEPipeDeclaration>

<DEBitmapDeclaration Name="displayDevice">
                <BitmapAttrCaps>
                        <AttrCap Name="width" Caps="WIDTH">
                                <Type Type="INTEGER"/>
                                <Access Type="RO"/>
                        </AttrCap>
                        <AttrCap Name="height" Caps="HEIGHT">
                                <Type Type="INTEGER"/>
                                <Access Type="RO"/>
                        </AttrCap>
                </BitmapAttrCaps>
                <BitmapFuncCaps>
                        <FuncCap Caps="LOAD_BITMAP" FuncName="Load"/>
                        <FuncCap Caps="COPY_FROM_BITMAP" FuncName="Copy"/>
                        <FuncCap Caps="RENDER_TEXT" FuncName="Text"/>
                </BitmapFuncCaps>
        </DEBitmapDeclaration>
</ISMLDevices>
```

The 'mouse' pipe defines an input stream that has three integer attributes describing the state of the buttons as an integer 'button'[14] and the last known relative change in 'x' and 'y' directions (these attributes are updated using the 'GetMouseInfo' function, see section 6.1). A 'displayDevice' provides output for the system, having 'width' and 'height' attributes and the capability of loading bitmap data from a file and rendering bitmap and rasterised text to a target.

6.2 Components part

Once supporting devices have been defined, components may then be specified that use one or more device classes as means of communicating with the user. A component definition shares some of the

[14] Here, the integer value (a 32-bit value) describes the on/off state of up to 32 switches

features of a 'concrete' interface widget (discussed in chapter 3) - it may contain attributes such as 'height' or 'width' or 'font name' or any arbitrarily defined property that is in some way meaningful to its design. Zero or many state models may also be declared within a component, each model operating independently[15]. State models in components may only refer to locally scoped attributes and render lists. Input or output are continuously updated through the execution of previously defined device functions within a render list. Any number of render lists may be declared, but only one list has 'focus' at any one time – this may be changed from within the state model, using the 'render focus' declaration. In this way, the appearance of a component or the manner in which it handles device input can be modified according to its state.

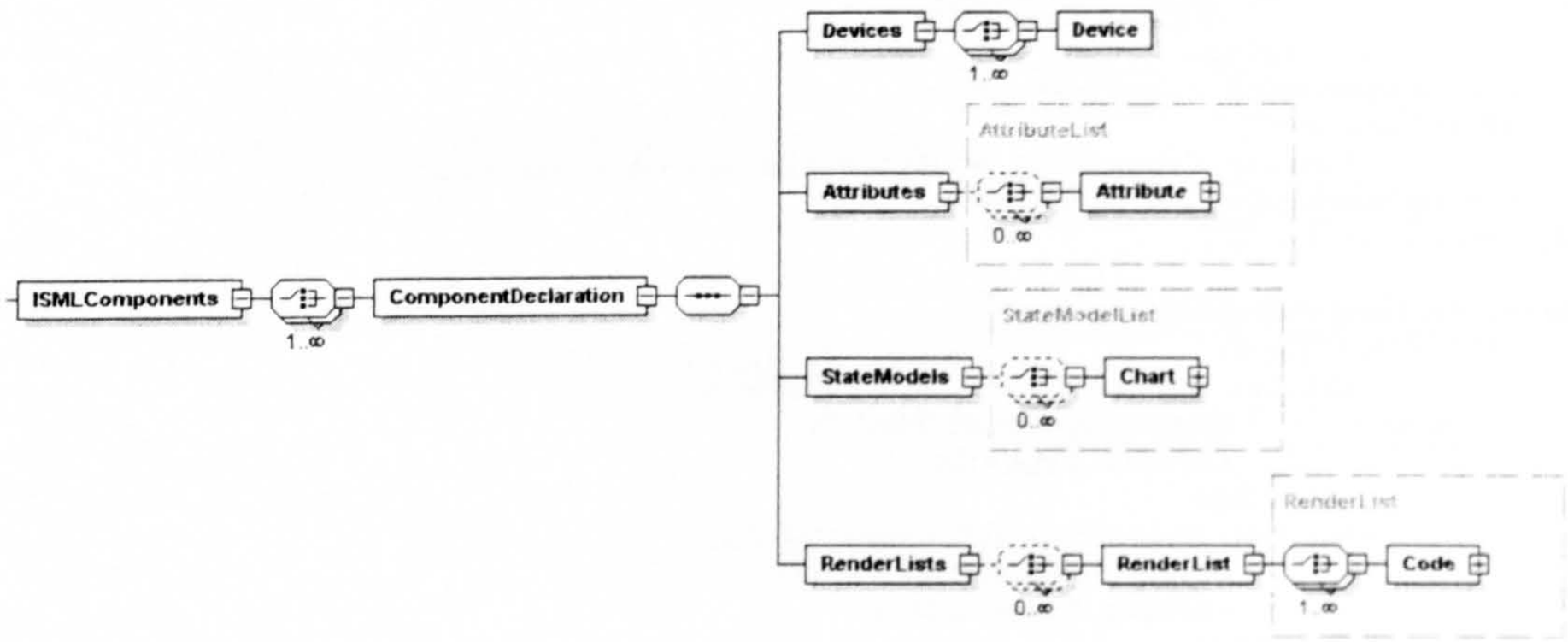


Figure 24 ISML Components part

[15] At present, synchronicity between models is not supported – operations and transitions are evaluated in the order that the state models are specified.

In the following example, a 'desktop mouse' is defined:

```
<ComponentDeclaration Name="desktopMouse">

        <Devices>
                <Device Name="deskMouse" Implements="mouse"/>
        </Devices>
```

In the 'devices' section, the abstracted mouse device defined earlier is instantiated as 'deskMouse'.

```
<Attributes>
                <Attribute Name="xChange">
                        <Type Type="INTEGER"/>
                        <Access Type="RO"/>
                </Attribute>
                <Attribute Name="yChange">
                        <Type Type="INTEGER"/>
                        <Access Type="RO"/>
                </Attribute>
                <Attribute Name="dmButton">
                        <Type Type="INTEGER"/>
                        <Access Type="RO"/>
                </Attribute>
        </Attributes>
```

The attributes of the desktop mouse are very similar to the shared device, but in this logical abstraction of the mouse, only one 'virtual button' is modelled – in this case it may be 'unarmed' or 'armed', see the state model below:

```
<StateModels>
        <Chart Name="dmButtonStates">
                <States>
                        <Node Name="dmUnArmed"/>
                        <Node Name="dmArmed"/>
                </States>
                <Transitions>
                        <Arc Name="dmMouseDown">
                                <RuleStatement>
                                        <Rule>
                                                <Statement>(dmButton == 1)</Statement>
                                        </Rule>
                                </RuleStatement>
                        </Arc>
                        <Arc Name="dmMouseUp">
                                <RuleStatement>
                                        <Rule>
                                                <Statement>(dmButton == 0)</Statement>
                                        </Rule>
                                </RuleStatement>
                        </Arc>
                </Transitions>
                <Topology>
                        <Network>
                                <Start>dmUnArmed</Start>
                                <Arc>dmMouseDown</Arc>
                                <End>dmArmed</End>
                        </Network>
                        <Network>
                                <Start>dmArmed</Start>
                                <Arc>dmMouseUp</Arc>
                                <End>dmUnArmed</End>
                        </Network>
                </Topology>
        </Chart>
</StateModels>
```

Rules enabling transitions between the unarmed and armed states refer to the device model for the current state of the buttons; in this case only the first switch is tested. Retrieval of this information is gathered by the render list 'dmInput'.

```
<RenderLists>
                <RenderList Name="dmInput">
                        <Code>
                                <Statement>
                                deskMouse->getMouseInfo(xChange,yChange,dmButton);
                                </Statement>
                        </Code>
                </RenderList>
        </RenderLists>
</ComponentDeclaration>
```

In fact, this definition of a 'desktop mouse' is incomplete since no graphical description of this logical device has been specified. This is quite deliberate since it may be desirable to define a number of subtly different mouse behaviours whilst binding them to a constant appearance. For example, in a system in which 2D and 3D interaction contexts co-exist, it may be useful to use a '3D Mouse' input component which accepts motion from the abstract mouse device and modifiers from an abstract keyboard to express six degrees of freedom in motion. For this reason, the presentation part is separately defined:

```
<ComponentDeclaration Name="desktopMouseImage">
        <Devices>
                <Device Name="dmiBitmap" Implements="displayDevice"/>

                <Device Name="dmiTarget" Implements="displayDevice" Target="AFFECT"/>
        </Devices>
```

In this device section, an optional 'retargetable' attribute is specified for the second device instance. Retargetable devices are useful when it is desirable for a component to use the device context of another. Any device used in a component may be either re-targeted (AFFECT) to another device or be declared as a potential target itself (EFFECT). Only one of each type may be declared in any one component, and no declared device may be both.

```
<Attributes>
        <Attribute Name="xPosition">
                <Type Type="INTEGER"/>
                <Access Type="RW"/>
        </Attribute>
        <Attribute Name="yPosition">
                <Type Type="INTEGER"/>
                <Access Type="RW"/>
        </Attribute>
</Attributes>

<StateModels>
        <Chart Name="dmiStates">
                <States>
                        <Node Name="Init">
                                <FireStatements>
                                        <Code>
                                                <Statement>
                                                dmiBitmap->Load("bitmap.bmp");
                                                </Statement>
                                        </Code>
                                </FireStatements>
                        </Node>
                        <Node Name="Running"/>
                </States>
                <Transitions>
                        <Arc Name="start"/>
                </Transitions>
                <Topology>
                        <Network>
                                <Start>Init</Start>
                                <Arc>Start</Arc>
                                <End>Running</End>
                        </Network>
                </Topology>
        </Chart>
</StateModels>
```

The position of the desktop mouse image is both readable and writable; a simple state model executes a call to the bitmap device to access an image to display before entering an endless loop in which the render list 'dmiRender' continuously renders the mouse image to its target (see section 6.5.1).

```
<RenderLists>
        <RenderList Name="dmiRender">
                <Code>
                        <Statement>
                        dmiTarget->CopyFrombitmap(dmiBitmap,xPosition,yPosition,0);
                        </Statement>
                </Code>
        </RenderList>
</RenderLists>
</ComponentDeclaration>
```

6.3 ISML Meta-objects part

Central to the ISML framework is the meta-object part in which the syntactic and semantic definitions that underpin the metaphorical aspects of a user interface are specified. In fact, the meta-object section can be split into two parts: (i) the metaphor objects and (ii) the interactor architectures derived from them.

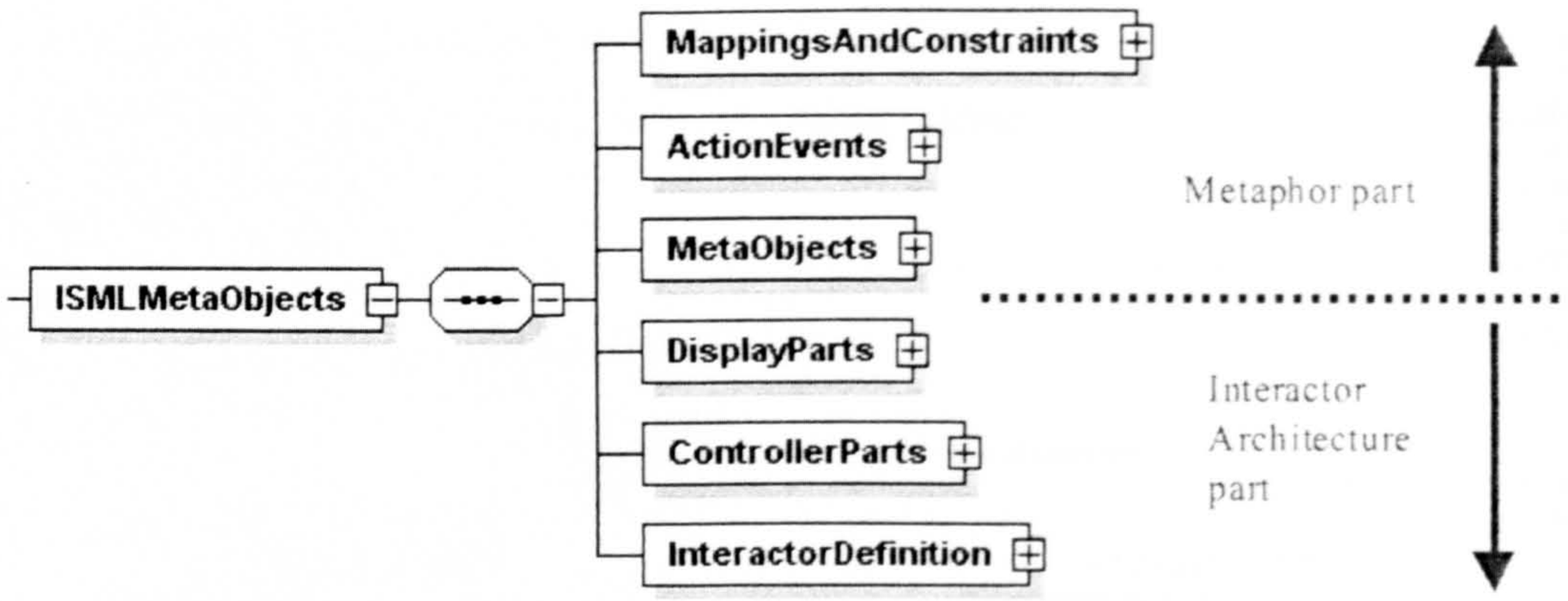


Figure 25 ISML Meta-objects part

6.3.1 MAPPINGS AND CONSTRAINTS

Within the metaphor part, mappings and constraints definitions are used to specify potential, transient relationships between subsequently defined meta-objects. Whilst both mappings and constraints, as well as Action-Events (see below), are defined outside of the scope of any particular meta-object they are not globally applicable.

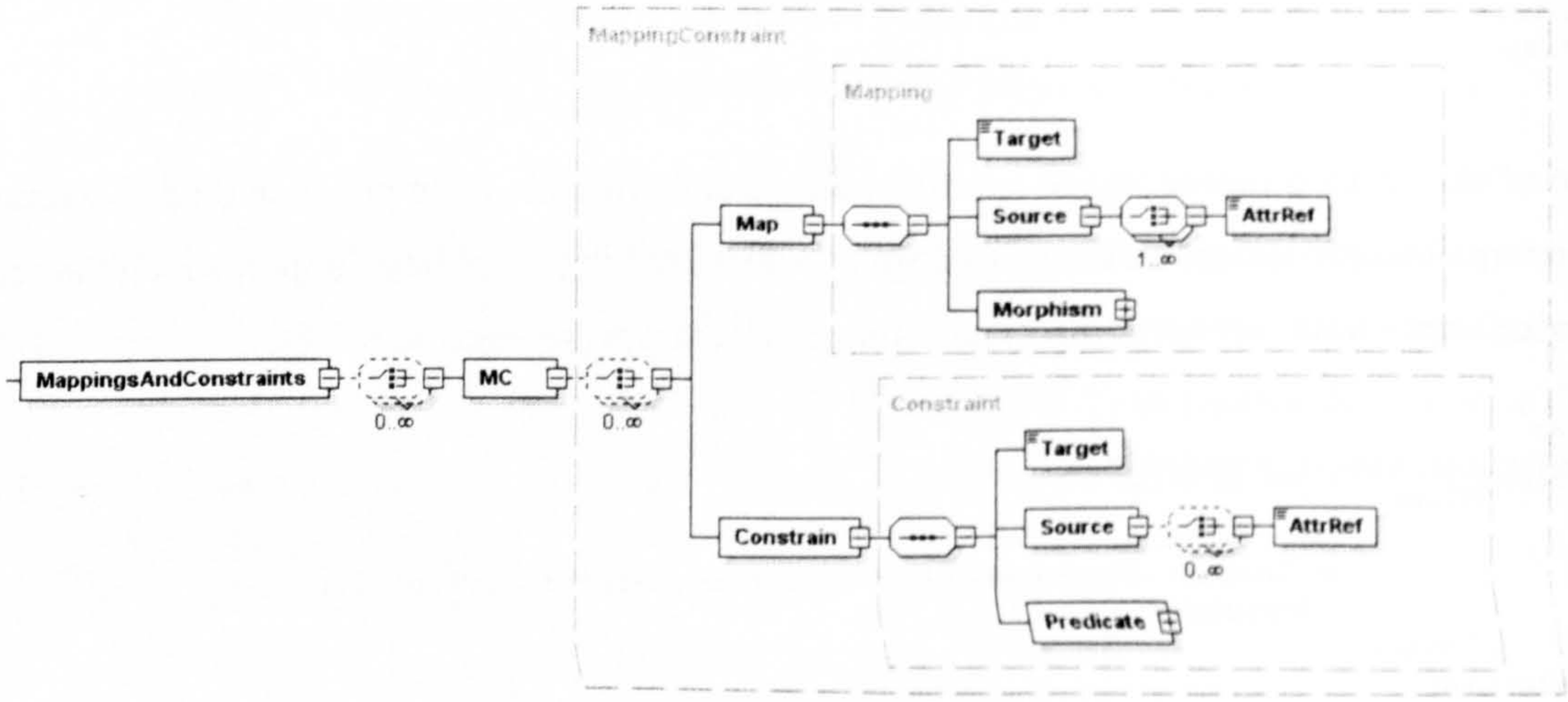


Figure 26 ISML Mapping and components

The specification syntax for both mappings and constraints are very similar – either element (*map* or *constrain*) declares a target upon which either the morphism or predicate will operate. Morphisms are mathematical relationships between named sources and the target, such as 'target = x + y' whilst predicates must be logical statements that evaluate to either true or false such as '(target > x)'. A mapping must have at least one source attribute as the basis for the mapping expression, whilst for a constraint this is optional. In defining a mapping or constraint, not only is a relationship specified but also an associated, ordered *set* object which may contain zero or many references to meta-objects.

Adding meta-objects to a particular mapping-constraint (or 'MC' – mappings and constraints are contained in this single unit) automatically applies mappings (if possible) and allows the testing of constraints (see section 6.3.6).

In the following example, a very simple direct manipulation environment is imagined in which objects, such as the cube, may be 'picked up' from some surface by an entity (depicted by the arrow), moved, and then 'dropped'. In addition, it may be desirable that the objects being manipulated cannot be dropped outside of the surface.

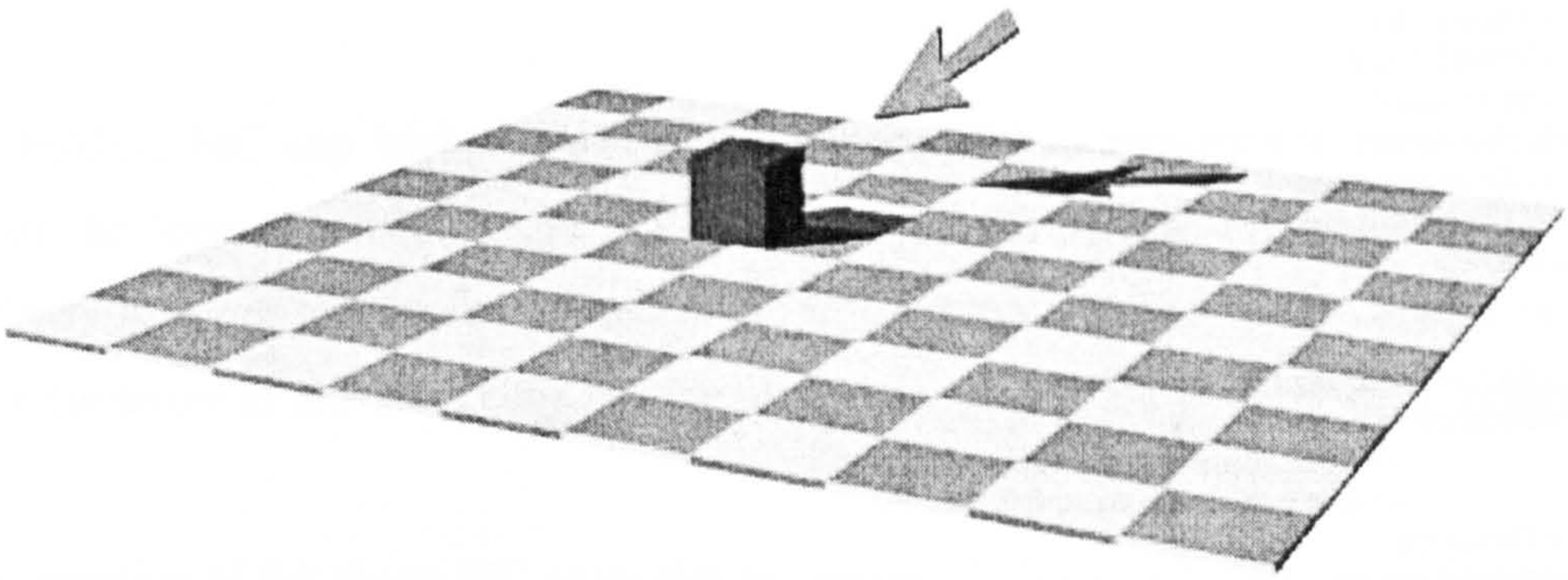


Figure 27 Direct manipulation scenario

In imagining this environment in an abstract form, a number of simple concepts are useful – that of *ownership*, *containment* and *holding*. Figure 27 is presented as a three dimensional model – this is for the illustration of each object's context in the example, the MCs given here do not model 3D operations, although they could be extended to do so.

The mappings and constraints for these three concepts are given below.

```
<MC Name="owns"/>
```

An empty MC, 'owns' is the simplest type of mapping-constraint that will be used to denote exclusive ownership of one object by another. This basic MC allows the model to maintain an association with the cubes resting on its surface, as opposed to cubes that may be elsewhere in the environment. 'Contains' is an MC with two constraints on target attributes *x* and *y* – for ease of reading, normally illegal characters in an XML document have been left non-escaped.

```
<MC Name="contains">
        <Constrain>
                <Target>xPosition</Target>
                <Source>
                        <AttrRef>xPosition</AttrRef>
                        <AttrRef>width</AttrRef>
                </Source>
                <Predicate>
                <Statement>
                ( ( target > xPosition ) && (target < xPosition + width) )
                </Statement>
                </Predicate>
        </Constrain>

<Constrain>
                <Target>yPosition</Target>
                <Source>
                        <AttrRef>yPosition</AttrRef>
                        <AttrRef>height</AttrRef>
                </Source>
                <Predicate>
                <Statement>
                ( ( target > yPosition)  && (target < yPosition +height) )
                </Statement>
                </Predicate>
        </Constrain>
</MC>
```

In this simple two dimensional containment relation, a point in 2D space is considered 'contained' within another object if that object has a boundary defined by xPosition, yPosition, width and height attributes. It may be useful to test the 'contains' constraint to verify whether an object being dropped falls within the boundary of the surface. The final MC is an morphism named 'holds' that maps the xPosition and yPosition attributes of the source object to the same attributes of the target. When the holding object, in this case represented by an arrow, holds a cube the position of that arrow affects the position of the cube.

```
<MC Name="holds">
        <Map>
                <Target>xPosition</Target>
                <Source>
                        <AttrRef>xPosition</AttrRef>
                </Source>
                <Morphism>
                        <Statement>target = xPosition</Statement>
                </Morphism>
        </Map>
        <Map>
                <Target>yPosition</Target>
                <Source>
                        <AttrRef>yPosition</AttrRef>
                </Source>
                <Morphism>
                        <Statement>target = yPosition</Statement>
                </Morphism>
        </Map>
</MC>
```

Before the holding MC can be enforced, it would be useful to remove the cube reference from the influences of the 'owns' and 'contains' MCs affected by the surface on the cube. This serves two purposes: firstly it serves to release the association of the cube with the surface and secondly, it may be desirable for the arrow to move the cube outside of the boundary of the surface whilst manipulating it.

It is *not* the intention of the above MC examples to capture all the possible semantic features of terms like 'contains' or 'holds' but rather just to express useful relationships associated with them. In Figure 27, the objects discussed here are represented in 3D not because the MCs used here express all the features necessary to model a 3D environment, but to emphasise some of the elementary semantics associated with these manipulations.

6.3.2 ACTION-EVENTS

All communication of the actions of one object on another is defined within the Action-Event (AE) section. Similar qualifiers must be declared for each meta-object definition regarding how the AE is used (see section 6.3.3).

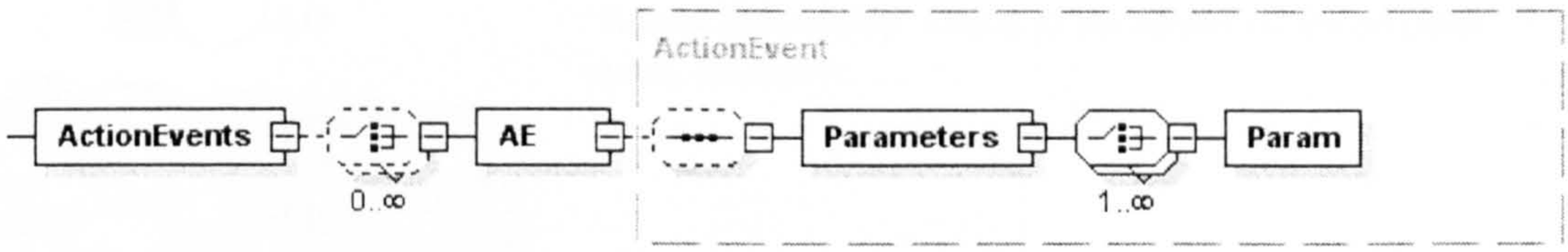
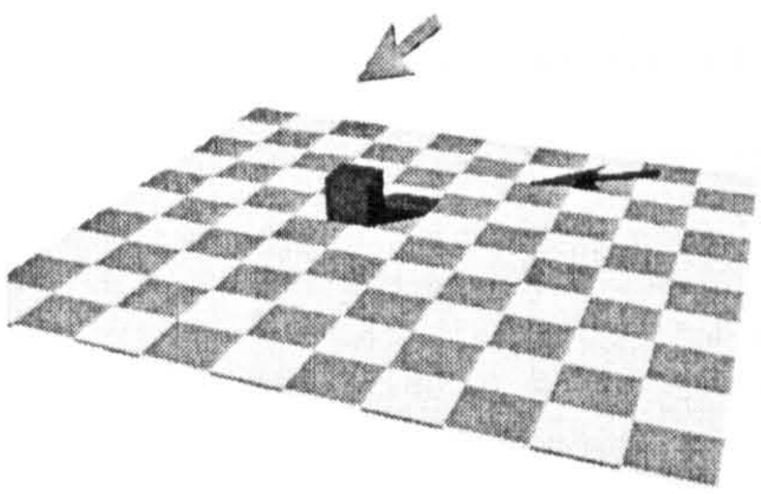


Figure 28 ISML Action-events

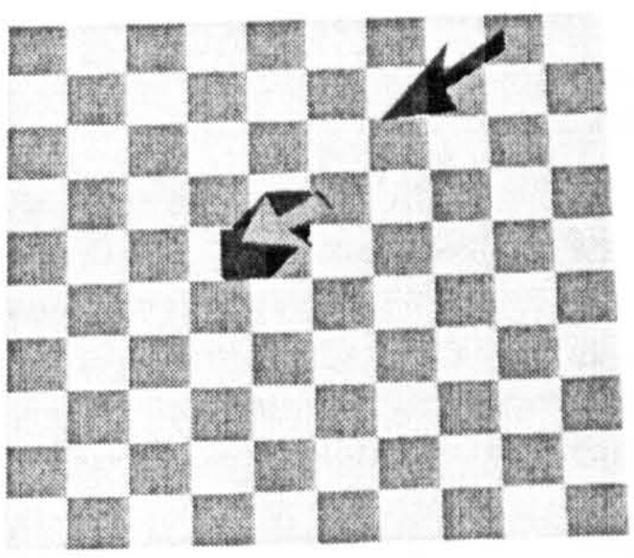
Each AE must be named and may also carry with it zero or many parameters that are specified as basic ISML types. Not all action-events defined here are necessarily used in the meta-object definitions; it may

be useful to define AEs that are used for a specific design solution in the interactor part (more on this in section 6.5.2).

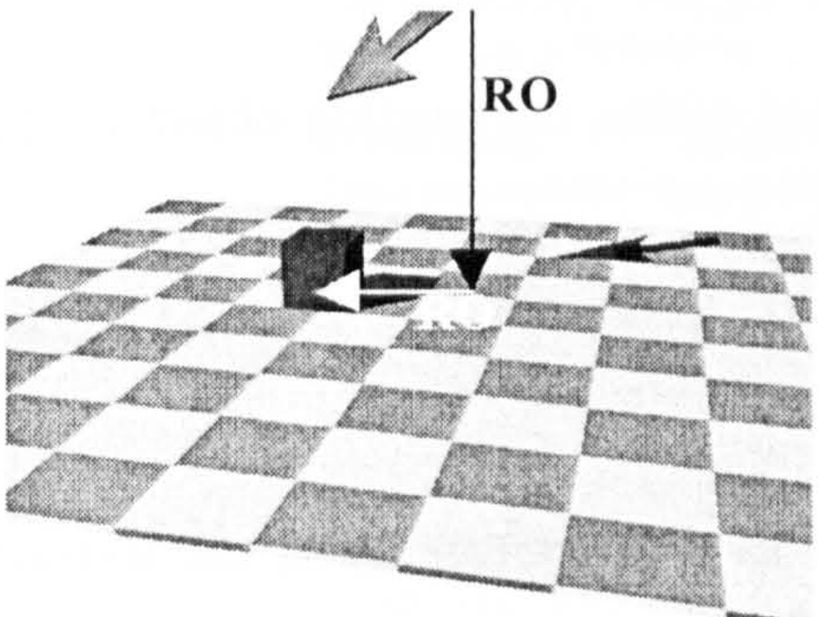
Following on from the simple direct-manipulation environment example begun in section 6.3.1, it is now possible to imagine a communication mechanism that would support the simple picking and dropping of the cube (see Figure 29).



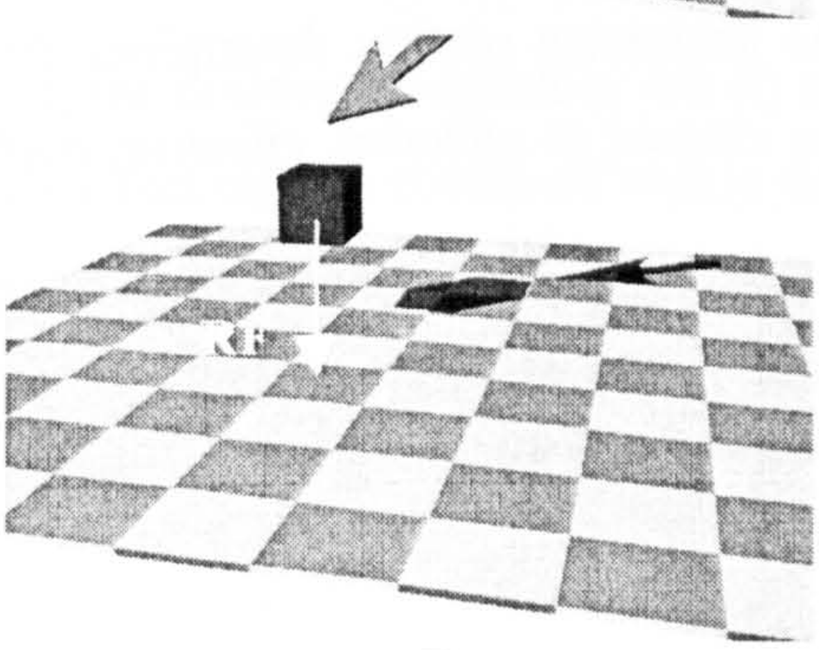
Initial state. The cube is a member of the surface's 'owns' MC.



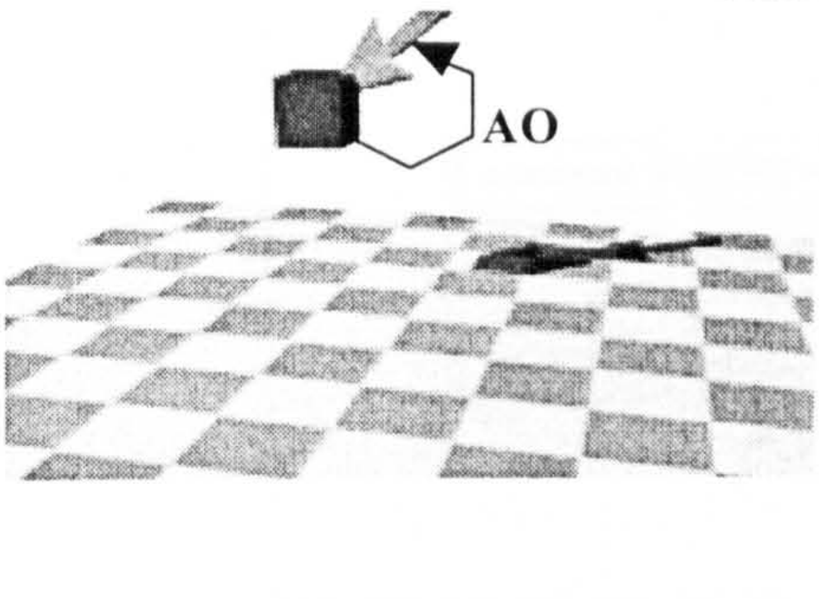
Begin pick up. The arrow is moved to indicate the focus of the cube object (i.e, their positions intersect in some manner).



Request Ownership (RO). The arrow sends a 'Request Ownership' action to the surface, supplying surface position information. The same action is then echoed from the surface to all the objects owned by it (in this example, only one).



Request Freedom (RF). Having received a Request Ownership action from the arrow, the cube tests the arrow for boundary containment using the 'contains' MC. If true, the cube sends a 'Request Freedom' action to its parent (the surface).



Accept Ownership (AO). Finally, the cube then sends an 'Accept Ownership' action to the sender of the original event, the arrow.

Figure 29 Direct manipulation Action-Event sequence

The syntax for the actions described in Figure 29 is relatively simple:

```
<AE Name="RequestOwnershipAction">
        <Parameters>
                <Param Name="eventSender" Type="SET"/>
                <Param Name="x" Type="INTEGER"/>
                <Param Name="y" Type="INTEGER"/>
        </Parameters>
</AE>

<AE Name="RequestFreedomAction">
        <Parameters>
                <Param Name="eventSender" Type="SET"/>
        </Parameters>
</AE>

<AE Name="AcceptOwnershipAction">
        <Parameters>
                <Param Name="objects" Type="SET"/>
                <Param Name="x" Type="INTEGER"/>
                <Param Name="y" Type="INTEGER"/>
        </Parameters>
</AE>
```

In all cases, a set object is used to refer to either an MC contained within the sending object or the sending object itself, signified as 'ME'.

6.3.3 META-OBJECTS

Metaphorical objects specified in the ISML framework are defined as meta-objects *types*, the abstract parts of which are comprised of attributes and state models. The semantics section determines the object's use of previously defined mapping-constraints which may be classed as *affective*, *effective*, *both affective and effective* or *exclusively affects*.

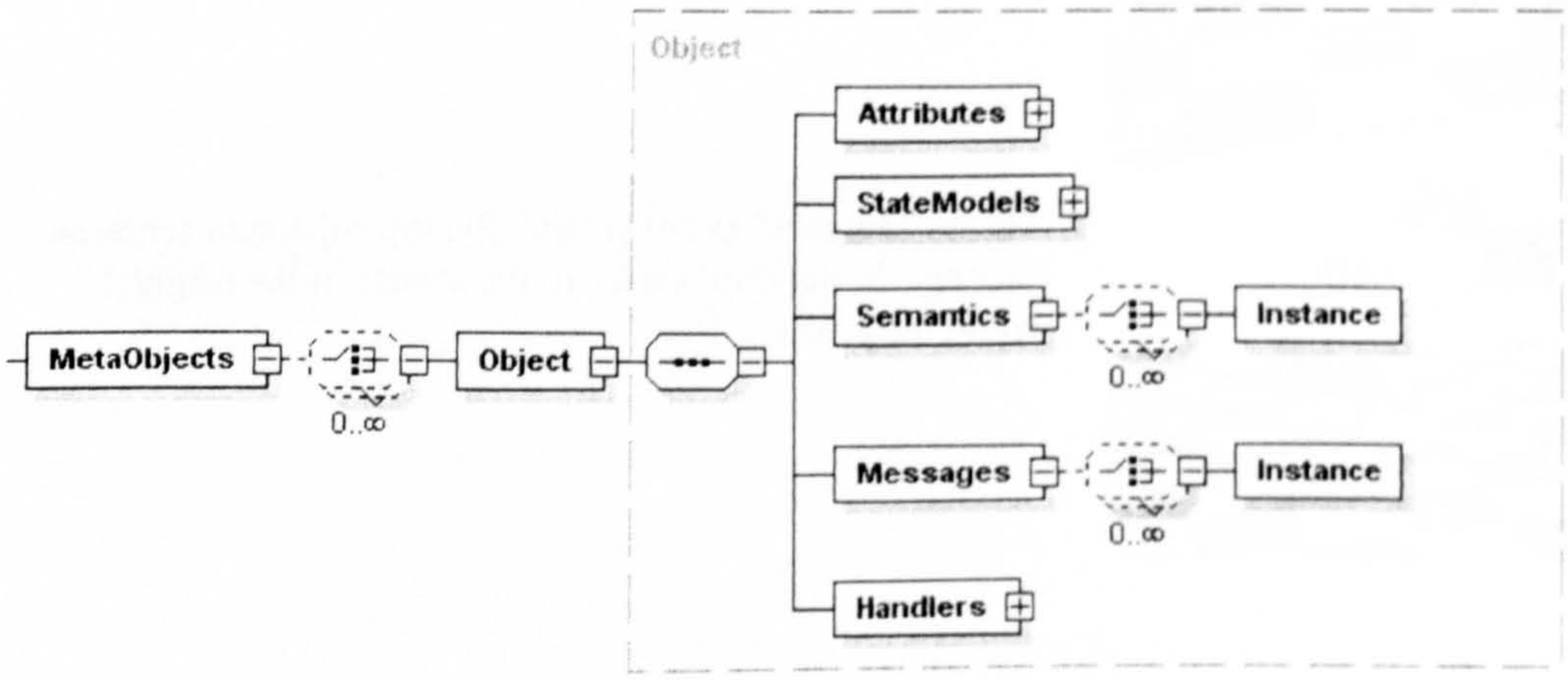


Figure 30 ISML Meta-object

One instance of an MC or AE class may be instantiated in the definition. In the partially complete example below, a 'desktop' object is specified:

```
<Object Name="desktop">
        <Attributes> … </Attributes>
        <StateModels> … </StateModels>

<Semantics>
                <Instance       Name="desktopOwns"
                                ImplementsMC="owns" Qualifier="XAFFECTS"/>

                <Instance       Name="desktopContains"
                                ImplementsMC="contains" Qualifier="AFFECTS"/>
                …
</Semantics>

<Messages>
                <Instance       Name="desktopROAction"
                ImplementsAE="RequestOwnershipAction"
                Qualifier="EFFECTS"/>

                <Instance       Name="desktopRFAction"
                ImplementsAE="RequestFreedomAction"
                Qualifier="EFFECTS"/>

                <Instance       Name="desktopAOAction"
                ImplementsAE="AcceptOwnershipAction"
                Qualifier="EFFECTS"/>
…
</Messages>

…

</Object>
```

Here, the desktop's semantics are (i) exclusive, affective ownership and (ii) affective containment. Both MCs affect objects contained within their sets[16] but since no effective relationship has been declared other objects cannot own the desktop. All three messages (described above) are effective to the desktop, which is to say that the desktop object will respond to these specific actions by other objects through its handlers.

[16] To qualify: objects in these sets must be legal 'effectees' of this MC, (specified in their semantics part).

6.3.4 HANDLER

An object reacts to action-events communicated to it through the execution of zero or many MC set operations or tests or any number of procedurally specified mathematical operations.

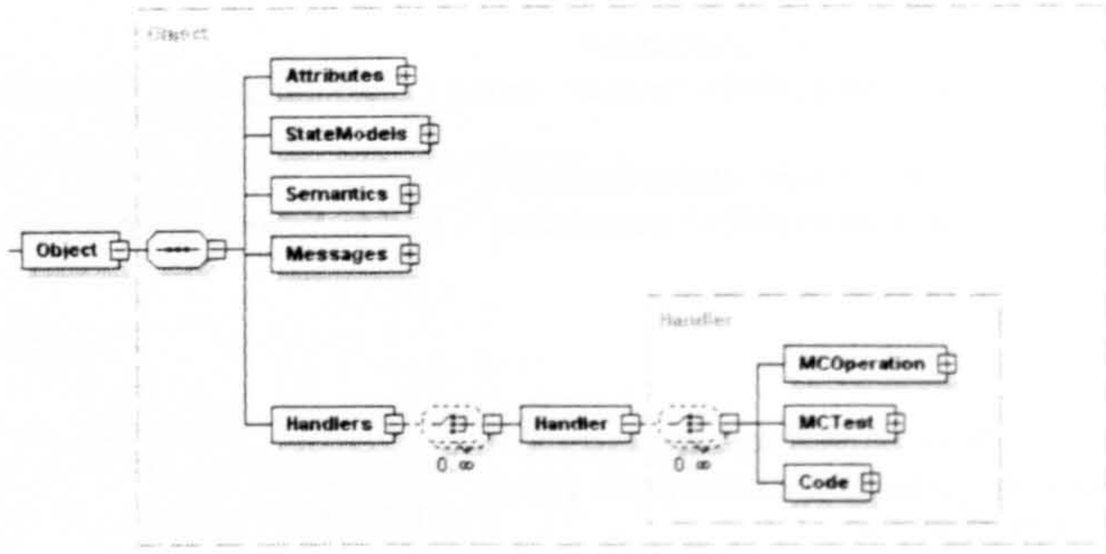


Figure 31 ISML Handler

6.3.5 MC SET OPERATORS

A small number of basic operations on MC sets are available to assist in the modelling of dynamic metaphor behaviour – these include emptying, adding to, subtracting from and calling an AE class of members within a set. Child elements named 'target' or 'source' are of the type *ObjectSet* (expanded only once in Figure 32 for brevity).

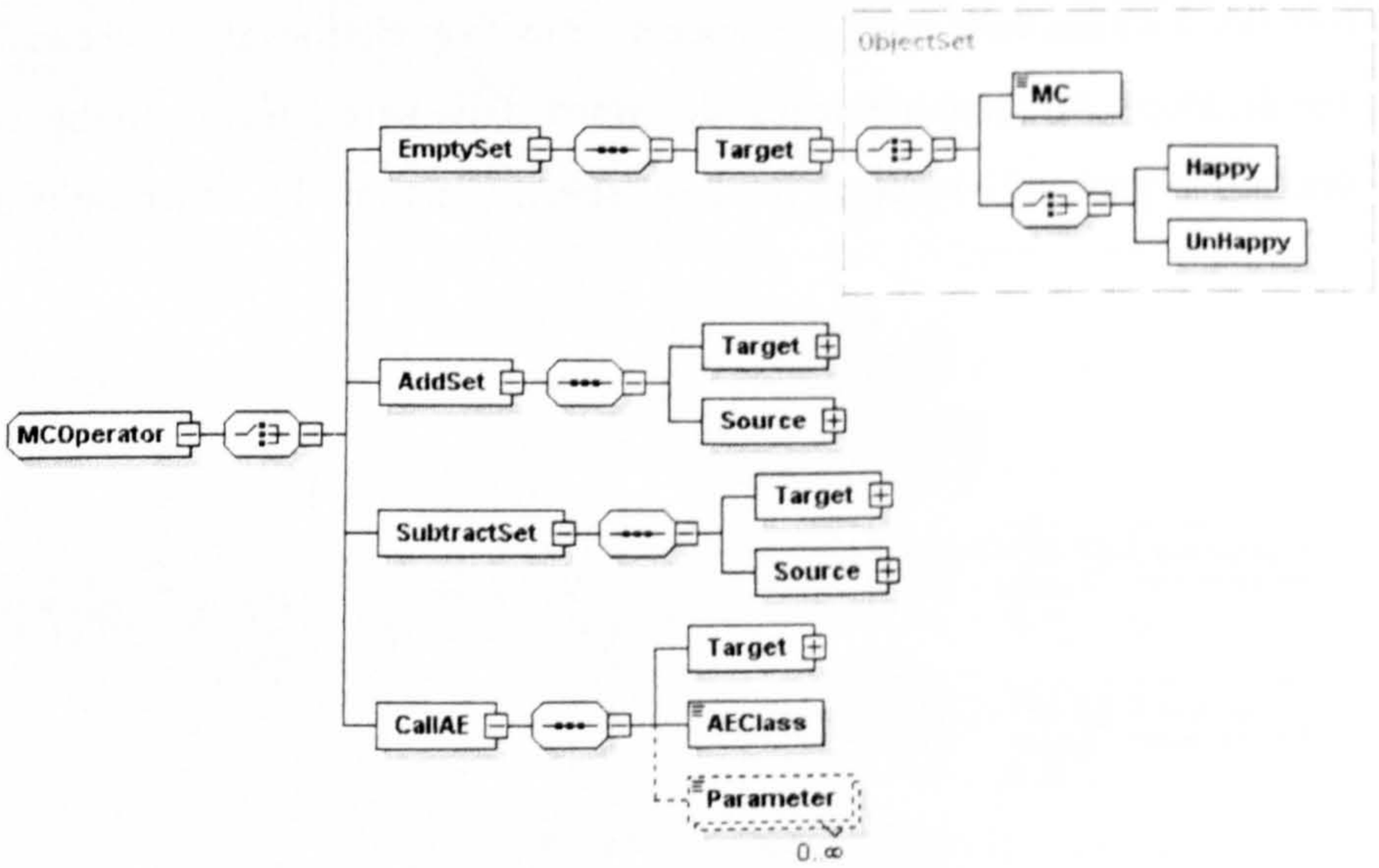


Figure 32 ISML Mapping-constraint operations

The focus of the set operation may be either a named set within the scope of the object the handler resides within or in temporary *result sets* 'happy' and 'unhappy'. A 'happy' set is filled with references to those

objects that evaluate true after an MC test expression (see section 6.3.6); the 'unhappy' set is filled with those objects that failed (every time an MC test expression is evaluated, the previous objects occupying the result sets are removed). Action-events may also be issued to any member of locally scoped MC sets although only objects that are legally able to accept them will receive notification of the action.

6.3.6 MC TEST EXPRESSIONS

Evaluating the condition of an MC set and the objects in it are possible through seven types of test, each of which evaluates to either true or false and possibly generates references to objects in either the happy or unhappy result sets. Objects within a set may be tested for a particular 'state focus', specified as a state name (string type) and the name of the state model (if the object has no such state model, the test is considered a failure).

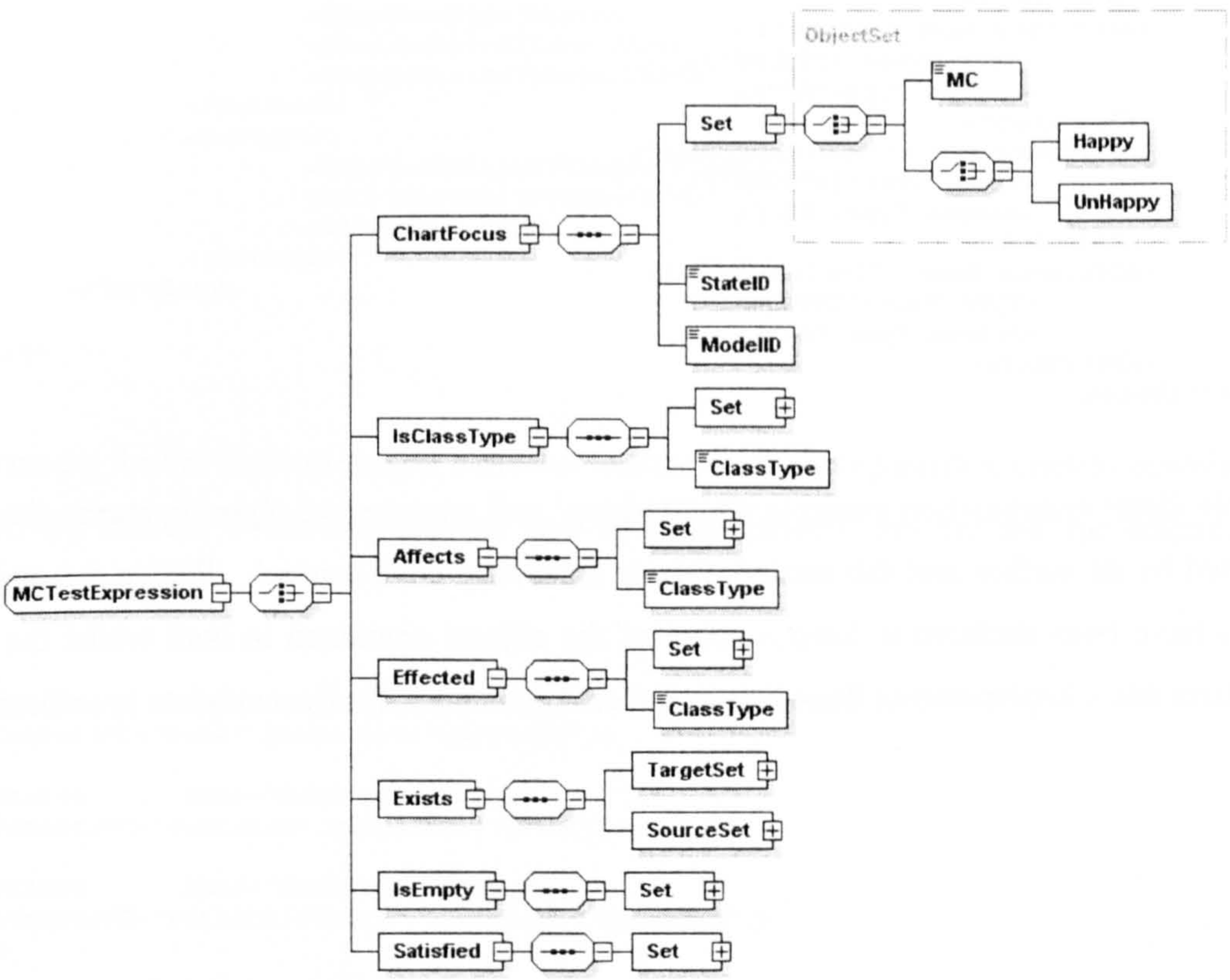


Figure 33 ISML Mapping-constraint test expressions

Class types may also be queried for a set such that those objects that are of that class are considered successes. Each object may also be evaluated with respect to whether or not it is capable of affecting a particular AE or being effected by it. Specific instances (as opposed to classes) of an object may be evaluated using the 'exists' test in which the object(s) of the source are sought in the target (the result is only considered a success when all objects in the source are found). A test for emptiness of a particular set is also provided, failure resulting in existing objects being placed in an 'unhappy' result. Finally, the

satisfaction of any existing constraints within an MC may be tested; only objects satisfying all constraints are placed in the 'happy' result set.

Now that the potential properties of a handler have been reviewed, the definition of the 'desktop' begun earlier can be completed:

```
<Object Name="desktop">
        <Attributes>
                <Attribute Name="xPosition">
                        <Type Type="INTEGER"/>
                        <Access Type="RO"/>
                </Attribute>
                <Attribute Name="yPosition">
                        <Type Type="INTEGER"/>
                        <Access Type="RO"/>
                </Attribute>
                <Attribute Name="width">
                        <Type Type="INTEGER"/>
                        <Access Type="RO"/>
                </Attribute>
                <Attribute Name="height">
                        <Type Type="INTEGER"/>
                        <Access Type="RO"/>
                </Attribute>
                <Attribute Name="containedItems">
                        <Type Type="INTEGER"/>
                        <Access Type="RO"/>
                </Attribute>
                <Attribute Name="floatingItems">
                        <Type Type="INTEGER"/>
                        <Access Type="RO"/>
                </Attribute>
</Attributes>
```

In this simple direct manipulation example, the 'desktop' will manage two object contexts, the first those objects owned by its surface and the second objects inhabiting its 'airspace'. Within the attribute part, two integers have been declared to keep a count of the objects contained in both whilst the semantics section declares MCs implementing these contexts (see Appendix G for the complete specification of this example).

```
<StateModels>
<Chart Name="desktopStates">
        <States>
                <Node Name="empty"/>
                <Node Name="containsThings"/>
        </States>

<Transitions>
                <Arc Name="recievesItems">
                <AEOccur AEClass="AcceptOwnershipAction" Direction="EFFECTS"/>
                </Arc>

<Arc Name="loosesAllItems">
                <MCTest>
                        <IsEmpty>
                                <Set>
                                        <MC>desktopOwns</MC>
                                </Set>
                        </IsEmpty>
                </MCTest>
                </Arc>

</Transitions>
                <Topology>
                        <Network>
                                <Start>empty</Start>
                                <Arc>receivesItems</Arc>
                                <End>containsThings</End>
                        </Network>
                        <Network>
                                <Start>containsThings</Start>
                                <Arc>loosesAllItems</Arc>
                                <End>empty</End>
                        </Network>
                </Topology>
</Chart>
</StateModels>
```

The bi-state model for the desktop simply indicates whether the desktop surface context contains items or not based on the Accept Ownership actions sent by other objects and the test for emptiness of the desktop's ownership MC.

```
<Semantics>
        <Instance       Name="desktopOwns"
        ImplementsMC="owns" Qualifier="XAFFECTS"/>

        <Instance       Name="desktopContains"
        ImplementsMC="contains" Qualifier="AFFECTS"/>

        <Instance       Name="desktopAirSpace"
        ImplementsMC="withinAirSpace" Qualifier="XAFFECTS"/>
</Semantics>
```

In addition to the two MCs 'desktop Owns' and 'desktop Contains' this desktop also now maintains objects within an 'air space' – space above the surface of the desktop, objects contained within which, are rendered last in the implementation of the system (see section 6.5.1).

```
<Messages>
        <Instance       Name="desktopROAction"
                        ImplementsAE="RequestOwnershipAction"
                        Qualifier="EFFECTS"/>

        <Instance       Name="desktopRFAction"
                        ImplementsAE="RequestFreedomAction"
                        Qualifier="EFFECTS"/>

        <Instance       Name="desktopAOAction"
                        ImplementsAE="AcceptOwnershipAction"
                        Qualifier="EFFECTS"/>

        <Instance       Name="desktopREAction"
                        ImplementsAE="RenderAction"
                        Qualifier="AFFECTS"/>

        <Instance       Name="desktopEASAction"
                        ImplementsAE="EnterAirSpaceAction"
                        Qualifier="EFFECTS"/>
</Messages>
```

Two additional AEs appear in the messages part in the form of 'Render Action' (this is used later in the Interactor definition rather than a part of the metaphor model) and the 'Enter Air Space' action (an important part of the metaphor model, but actually applied at instantiation), see section 6.5.1.

```
<Handlers>
        <Handler AEImplementation="desktopROAction">
                <MCOperation>
                        <CallAE>
                                <Target>
                                        <MC>desktopOwns</MC>
                                </Target>
                                <AEClass>RequestOwnershipAction</AEClass>
                                <Parameter>eventSender</Parameter>
                                <Parameter>x</Parameter>
                                <Parameter>y</Parameter>
                        </CallAE>
                </MCOperation>
        </Handler>

        <Handler AEImplementation="desktopRFAction">
                <MCOperation>
                        <SubtractSet>
                                <Target>
                                        <MC>desktopOwns</MC>
                                </Target>
                                <Source>
                                        <MC>eventSender</MC>
                                </Source>
                        </SubtractSet>
                </MCOperation>
                <Code>
                        <Statement>ownedItems--;</Statement>
                </Code>
        </Handler>

        <Handler AEImplementation="desktopAOAction">
                <MCOperation>
                        <AddSet>
                                <Target>
                                        <MC>desktopOwns</MC>
                                </Target>
                                <Source>
                                        <MC>objects</MC>
                                </Source>
                        </AddSet>
                </MCOperation>
                <Code>
                        <Statement>ownedItems++;</Statement>
                </Code>
        </Handler>

        <Handler AEImplementation="desktopEASAction">
                <MCOperation>
                        <AddSet>
                                <Target>
                                        <MC>desktopAirSpace</MC>
                                </Target>
                                <Source>
                                        <MC>eventSender</MC>
                                </Source>
                        </AddSet>
                </MCOperation>
                <Code>
                        <Statement>floatingItems++;</Statement>
                </Code>
        </Handler>
</Handlers>
</Object>
```

Four handlers collectively specify the behaviour of the simple manipulation of objects into and out of the desktop ownership context. Requests for object ownership are passed to the objects owned by the desktop - the desktop cannot be owned itself. An object requesting freedom from the desktop is removed from the ownership MC, whilst those requesting ownership are added. Objects entering the airspace are added in the same way.

To complete the simple direct manipulation model, a generic desktop object 'deskObject' and a pointing object 'pointerObject' must also be defined. Instead of fully listing the specification here, the parts salient to the overall model in each object are described in brief (see Appendix G for the complete specification).

6.3.7 THE GENERIC DESKTOP OBJECT

This object may be owned, contained and held by others but cannot do so itself. It may receive requests for ownership and in doing so request freedom from its current owner, as specified in the handler below:

```
<Handlers>
        <Handler AEImplementation="deskObjectROAction">
                <MCOperation>
                        <CallAE>
                                <Target>
                                        <MC Affector="true">deskObjectOwned</MC>
                                </Target>
                                <AEClass>RequestFreedomAction</AEClass>
                                <Parameter>ME</Parameter>
                        </CallAE>
                </MCOperation>
                <MCOperation>
                        <CallAE>
                                <Target>
                                        <MC>eventSender</MC>
                                </Target>
                                <AEClass>AcceptOwnershipAction</AEClass>
                                <Parameter>ME</Parameter>
                        </CallAE>
                </MCOperation>
        </Handler>
</Handlers>
```

A qualifier "Affector = true" determines that the AE being called is directed to the object(s) exerting the MC on this object.

6.3.8 THE POINTING OBJECT

Key behaviour of the pointing object is to pick up an object and then release it and is primarily specified in the state model:

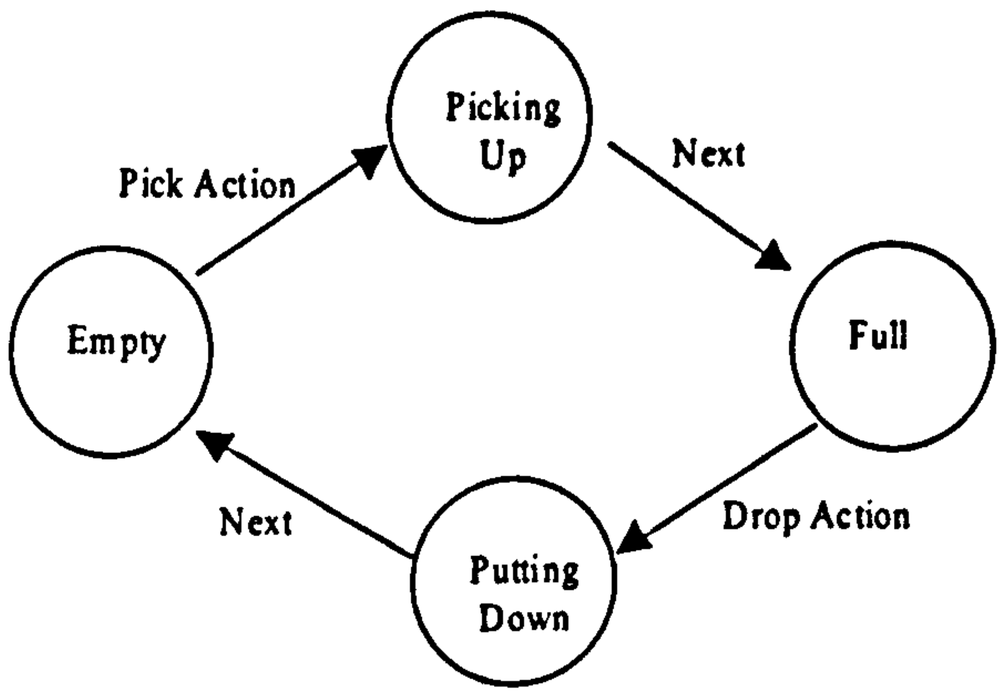


Figure 34 Pointing object state model

Transitions between the states are either transient ('Next' has no rule) or based on the value of the pointer's action attribute and the condition of the 'ownership' MC:

```
<Arc Name="pickAction">
        <RuleStatement>
                <Rule>
                        <Statement>(pointerAction == 1)</Statement>
                </Rule>
        </RuleStatement>
        <MCTest Logic="AND">
                <IsEmpty>
                        <Set>
                                <MC>pointerObjectOwns</MC>
                        </Set>
                </IsEmpty>
        </MCTest>
</Arc>

<Arc Name="dropAction">
        <RuleStatement>
                <Rule>
                        <Statement>(pointerAction == 0)</Statement>
                </Rule>
        </RuleStatement>
        <MCTest Logic="AND" Negate="NOT">
                <IsEmpty>
                        <Set>
                                <MC>pointerObjectOwns</MC>
                        </Set>
                </IsEmpty>
        </MCTest>
</Arc>
```

Rules in transitions may be logically concatenated and negated, evaluating pairs from the start of the list. The 'Pick Action' transition will fire if the pointer action is 1 and the ownership MC set is currently empty. Conversely, if the pointer action attribute is zero and the ownership set is not empty, the 'Drop Action' transition will fire. On entering the state of 'Picking Up', the pointer sends a request for ownership to the object whose 'airspace' it currently resides in – the desktop (the pointer set to belong to this 'airspace' later in the system set-up, see section 6.5.3).

```
<Node Name="pickingUp">
        <FireStatements>
                <MCOperation>
                        <CallAE>
                                <Target>
                                <MC Affector="true">pointerInAirSpace</MC>
                                </Target>

                                <AEClass>RequestOwnershipAction</AEClass>
                                        <Parameter>ME</Parameter>
                                        <Parameter>x</Parameter>
                                        <Parameter>y</Parameter>
                        </CallAE>
                </MCOperation>
        </FireStatements>
</Node>

<Node Name="puttingDown">
        <FireStatements>
                <MCOperation>
                        <CallAE>
                                <Target>
                                        <MC Affector="true">pointerInAirSpace</MC>
                                </Target>
                                <AEClass>AcceptOwnershipAction</AEClass>
                                        <Parameter>pointerObjectOwns</Parameter>
                        </CallAE>
                </MCOperation>
                <MCOperation>
                        <EmptySet>
                                <Target>
                                        <MC>pointerObjectOwns</MC>
                                </Target>
                        </EmptySet>
                </MCOperation>
                <MCOperation>
                        <EmptySet>
                                <Target>
                                        <MC>pointerObjectHolds</MC>
                                </Target>
                        </EmptySet>
                </MCOperation>
        </FireStatements>
</Node>
```

Finally, whilst dropping the currently held object, the pointer sends a request for acceptance of ownership of its held item and then empties references to that item from both its ownership and holding mapping constraint sets.

6.4 Meta-Interactor definition

The ISML Meta-object part concludes with definitions of interactor types based on the meta-objects already defined for use in the proceeding part of the specification. Interactors will actualise some or all of the properties of the metaphor model at the user interface through the inheritance of meta-object abstractions. This is achieved by defining display and controller types and binding them with a meta-object.

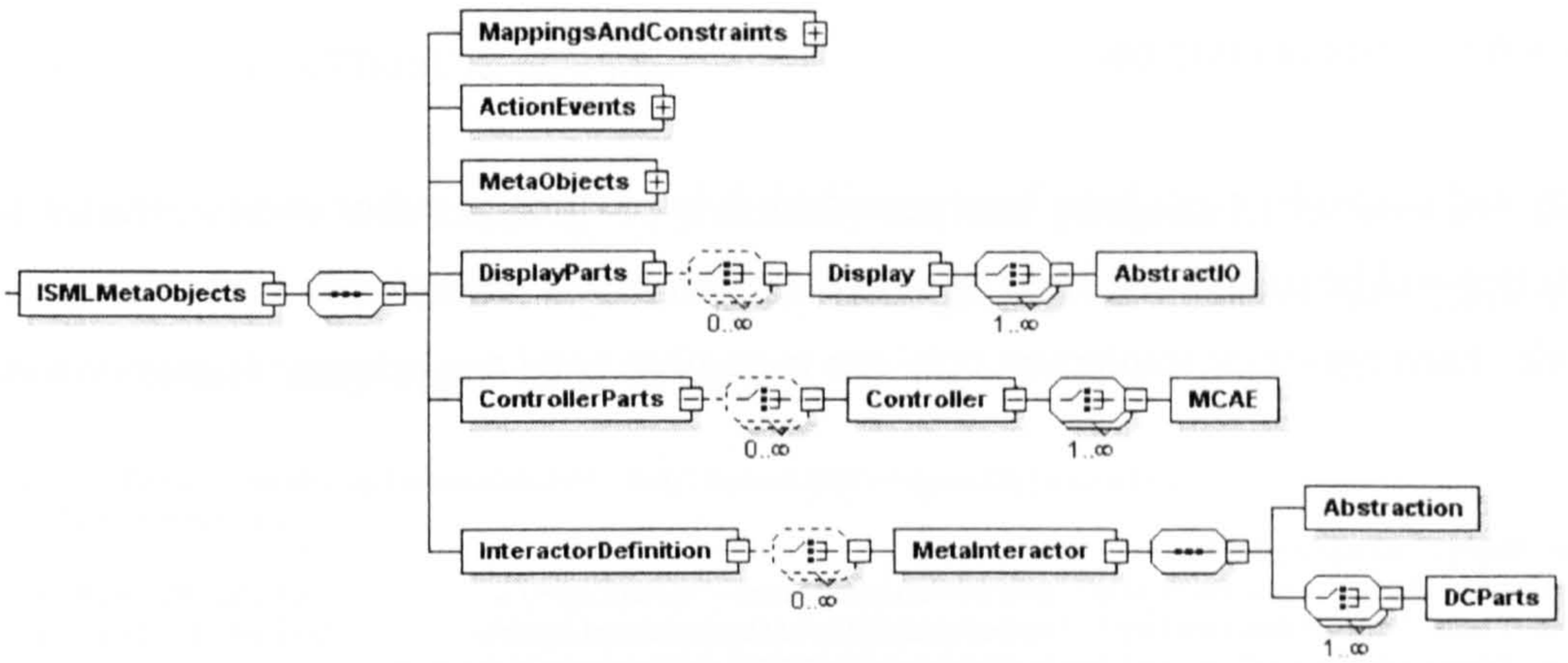


Figure 35 ISML Meta-Interactor definition

6.4.1 DISPLAY AND CONTROLLER PARTS

Each display part is simply a tuple of one or many named abstract input/output channels that will later be bound to a 'concrete' component (see section 6.5.1). Two display parts are named here, the second being more interesting as it will be used to bind the mouse device and image components together.

```
<Display Name="simpleDisplay">
        <AbstractIO Name="simplePart"/>
</Display>

<Display Name="compoundDisplay">
        <AbstractIO Name="inputPart"/>
        <AbstractIO Name="outputPart"/>
</Display>
```

Controllers are also simple tuples, collecting together one or many previously specified MC and AE definitions.

```
<Controller Name="globalController">
        <MCAE MCAERef="ownes"/>
        <MCAE MCAERef="contains"/>
        <MCAE MCAERef="holds"/>
<MCAE MCAERef="withinAirSpace"/>

<MCAE MCAERef="RequestOwnershipAction"/>
        <MCAE MCAERef="RequestFreedomAction"/>
        <MCAE MCAERef="AcceptOwnershipAction"/>
</Controller>
```

In the example given above, a global controller has been defined to manage all of the MCs and AEs already presented. This is a somewhat clumsy means of defining control for an interactor; it may be more desirable to specify a variety of controllers that exert particular types of management, such as for ownership and containment.

6.4.2 META-INTERACTOR DEFINITION

Once the display and controller parts have been specified, it is possible to define meta-interactor types – these are objects that will be used as the basis for interactor definitions proper in the Interactors part of the ISML framework. Each type must implement only one meta-object and one or many display or controller parts.

```
<MetaInteractor Name="desktopMI">
        <Abstraction Name="abstraction" ImplementsObject="desktop"/>
        <DCParts Name="desktopDisplay" ImplementsDC="simpleDisplay"/>
        <DCParts Name="desktopController" ImplementsDC="globalController"/>
</MetaInteractor>
```

Above, 'desktopMI' implements the desktop meta-object and uses the simple display and global controller[17]. Other meta-interactor definitions for the direct-manipulation example can be found in Appendix G.

6.5 Interactors

Specific details regarding the presentation of the metaphor and its links with a design solution for a particular problem are specified in the interactor part of ISML. An interactor definition must be based on a previously defined meta-interactor object and may be extended with problem specific abstractions using locally scoped attribute and state models.

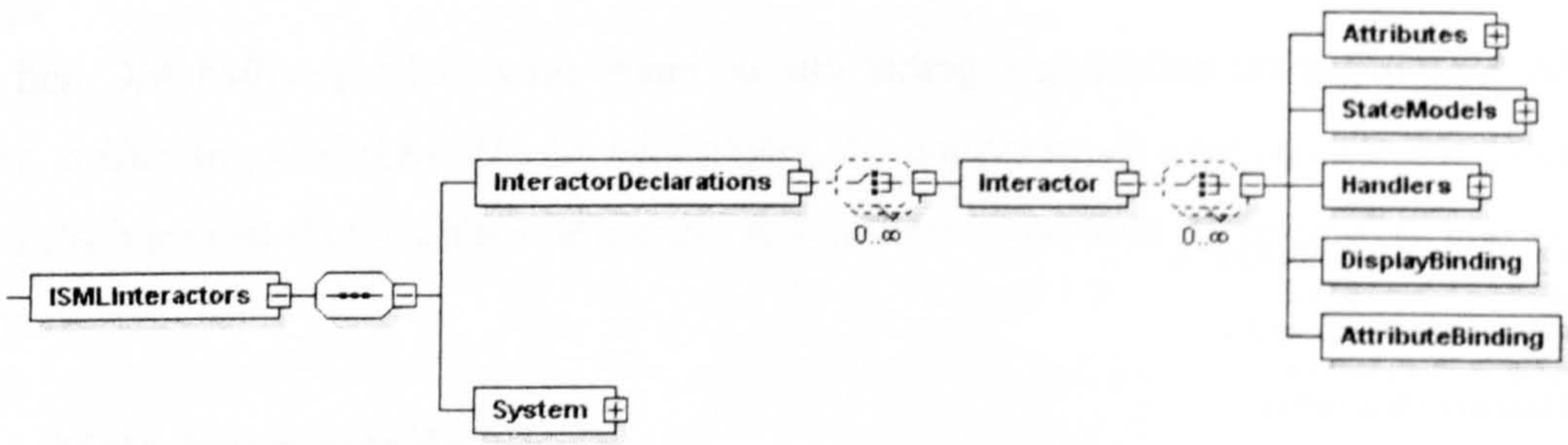


Figure 36 ISML Interactor FIGURE ISML14

[17] In fact, one of the members of the controller does not apply to the desktop making this a poor specification. However, for the sake of brevity and since the controller does not play an important role in this very simple example, this is noted but overlooked.

6.5.1 DISPLAY AND ATTRIBUTE BINDING

Display binding allows the mapping of previously defined components to abstract display parts (at present this is static). The desktop object uses the basic graphics component 'user display' as a concrete basis for the abstract 'simple part' of its display.

```
<Interactor Name="desktopInteractor" ImplementsMI="deskObjectMI">
        <StateModels>
                ...
        </StateModels>
        <DisplayBinding         DisplayPart="desktopDisplay"
                                DisplayAbstractIO="simplePart"
                                ImplementsComponent="userDisplay"/>

        <Handlers>
                ...
        </Handlers>
</Interactor>
```

In the partially specified 'pointer interactor' example below, the display parts are bound to input and output components respectively, effectively combining the two component behaviours into a cohesive object that behaves like a desktop mouse.

```
<Interactor Name="pointerInteractor" ImplementsMI="pointerMI">
        <StateModels>
                ...
        </StateModels>
        <DisplayBinding         DisplayAbstractIO="pointerIO"
                                DisplayPart="inputPart"
                                ImplementsComponent="desktopMouse"/>

        <DisplayBinding         DisplayAbstractIO="pointerIO"
                                DisplayPart="outputPart"
                                ImplementsComponent="desktopMouseImage"/>

        <Handlers>
                ...
        </Handlers>

</Interactor>
```

Once an association between the abstract display part and a component has been established, attribute bindings allow bi-directional mappings from the interactor abstraction to the concrete component. Now the generic 'deskObject' previously defined in the meta-object part is refined as a 'file' object for use in the desktop environment. Once again, the 'simple part' display is used but this time for a component called 'desktop file icon'. More interestingly, however, the attributes of this component are targets for the abstract co-ordinates of the *base meta-object* for this interactor class (xPosition and yPosition).

```
<Interactor Name="fileObjectInteractor" ImplementsMI="deskObjectMI">
        <Attributes>

        </Attributes>

        <DisplayBinding         DisplayPart="deskObjectDisplay"
                                DisplayAbstractIO="simplePart"
                                ImplementsComponent="desktopFileIcon"/>

        <AttributeBinding       Direction="AFFECTIVE"
                                InteractorAttribute="xPosition"
                                DisplayPart="deskObjectDisplay"
                                AbstractIO="simplePart"
                                AbstractIOAttribute="xPosition"/>

        <AttributeBinding       Direction="AFFECTIVE"
                                InteractorAttribute="yPosition"
                                DisplayPart="deskObjectDisplay"
                                AbstractIO="simplePart"
                                AbstractIOAttribute="yPosition"/>

        <Handlers>
                ...
        </Handlers>
</Interactor>
```

The effect of this mapping is the partial actualisation of parts of the metaphor model into a specific design representation. Component mapping suffices to describe the appearance of the interactive objects but additional logic must be introduced into the interactor design to complete the realisation. To achieve the effect of a conventional, two-dimensional representation of a simple direct manipulation environment, the order of rendering must be managed such that objects are drawn on top of the desktop surface and the pointing device on top of these. Action-events specified in the meta-object part fall into three implicit categories: those that are used solely in the metaphor model; those that are used in the metaphor and the interactor model; those that are exclusive to the interactor model.

In order to complete the simple environment, two additional AEs not used in the metaphor model, 'Render Action' and 'Give Me Pixels' are utilised by interactors (the impact of the use of AEs in this manner is discussed further in section 7).

```
<Node Name="alwaysOn">
        <FireStatements>

<MCOperation>
                <CallAE>
                        <Target><MC> ME </MC></Target>
                        <AEClass> RenderAction </AEClass>
                </CallAE>
        </MCOperation>

<MCOperation>
                <CallAE>
                        <Target><MC> desktopOwns </MC></Target>
                        <AEClass> RenderAction </AEClass>
                </CallAE>
        </MCOperation>

<MCOperation>
                <CallAE>
                        <Target><MC> desktopAirSpace</MC></Target>
                        <AEClass>RenderAction</AEClass>
                </CallAE>
        </MCOperation>

</FireStatements>
</Node>
```

Above, an excerpt from the single node state model within the interactor desktop object specifies 'Render Action' calls to three specific interactor groups. The first is a call to itself ('ME', the desktop), the second to all objects owned by the desktop and the third to objects within the desktop's airspace.

6.5.2 HANDLERS

All of the objects in this interactor model use the AE 'Render Action' to specify render statements that affect the mapped component rendering (determined by its current render focus – see section 6.2). The render action AE for the pointer object is shown below:

```
<Handle Name="pointerObjectREAction">
        <Render DisplayPart="pointerIO" AbstractIO="outputPart"/>
</Handle>
```

The remaining logic required for the interactor part is an example of dynamic re-targeting of abstract display parts (and consequently their associated components) to the context of another interactor's display.

```
<Handle Name="desktopGMPAction">
        <Retarget DisplayType="simpleDisplay">
                <Source MC="eventSender"/>
                <Target DisplayPart="desktopDisplay"/>
        </Retarget>
</Handle>
```

Re-targeting is only successful if all interactors in the source MC have a display part or parts of the type specified by 'display type' in the source element and the target element's 'display part' is also of the same type. Successful re-targeting maps all source interactor display parts to the single display part specified

by the target. The effect of this is that any components associated with source display parts with re-targetable devices then use the targeted device of the component associated with `<Target DisplayPart="desktopDisplay"/>`.

6.5.3 SYSTEM SET-UP

The interactor part of the ISML framework concludes with concrete instances of the interactive system to be instantiated within the 'system inventory' and any number of procedural set-up instructions that are executed at start-up.

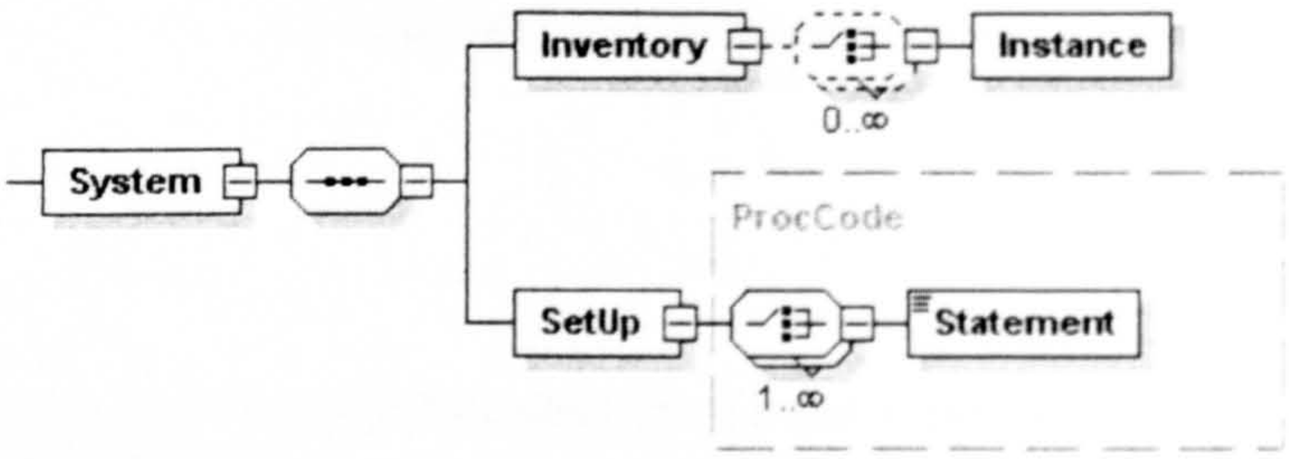


Figure 37 ISML System set up

```
<System>
        <Inventory>
        <Instance ImplementsInteractor="desktopInteractor" Name="myDesktop"/>
        <Instance ImplementsInteractor="pointerInteractor" Name="myPointer"/>

        <Instance ImplementsInteractor="fileObjectInteractor" Name="myFile"/>
        </Inventory>

        <SetUp>
                <Statement>myFile->xPosition = 50;</Statement>
                <Statement>myFile->yPosition = 50;</Statement>
                <Statement>myDesktop->desktopEASAction(myMouse);</Statement>
                <Statement>myDesktop->desktopAOAction(myFile);</Statement>
                <Statement>myDesktop->desktopGMPAction(myMouse);</Statement>
                <Statement>myDesktop->desktopGMPAction(myFile);</Statement>
        </SetUp>
</System>
```

An instance of each of the interactor definitions is created in the example above, followed by specifying some initial attributes. In set-up, action-events may also be called to initialise the expected conditions of both the metaphor and interactor models: the mouse enters desktop airspace, 'myFile' becomes owned by the desktop and finally, device context re-targeting is declared.

6.6 Tasks

Parts 'devices', 'components', 'meta-objects' and 'interactors' have so far encapsulated the appearance and behaviour of objects with respect to an interface design realisation[18] and potential metaphor behaviour that carries it, but so far without reference to the tasks that it might support. The ISML 'task world' re-uses the basic meta-object features to describe extant task related entities and their role within a hierarchical description of tasks.

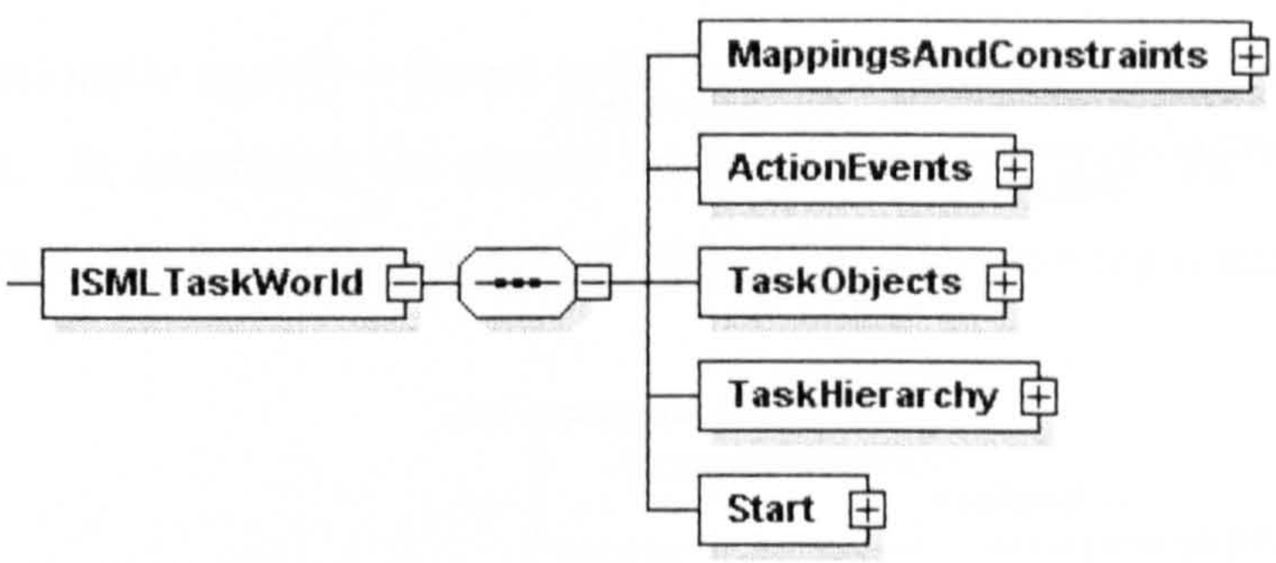


Figure 38 ISML Task world

Objects, MCs and AEs can maintain a high level of abstraction since the mapping from task to an actual interface design solution does not follow the same path as a derived interactor solution. Actions on objects are specified in the same way as described earlier (see section 6.3.2) but in addition, a 'user' object is specified:

[18] This term is used deliberately. A complete 'interactive system design' extends beyond that which the ISML framework provides.

```
<Object Name="taskUser">
        <Attributes>
                -
        </Attributes>
        <StateModels/>
        <Semantics>
                <Instance       Name="userHolds"
                ImplementsMC="taskHolds"
                Qualifier="AFFECTS"/>
        </Semantics>

        <Messages>
                <Instance       Name="userPickUp"
                ImplementsAE="taskPickUp" Qualifier="AFFECTS"/>

                <Instance       Name="userRelease"
                ImplementsAE="taskRelease" Qualifier="BOTH"/>
        </Messages>

        <Handlers>
                <Handler AEImplementation="userRelease">
                        <MCOperation>
                                <AddSet>
                                        <Target>
                                                <MC>userHolds</MC>
                                        </Target>
                                        <Source>
                                                <MC>object</MC>
                                        </Source>
                                </AddSet>
                        </MCOperation>
                        <Code>
                                <Statement>actionCount ++;</Statement>
                        </Code>
                </Handler>
        </Handlers>
</Object>
```

The simple task world used in this example requires that users pick up and release an object from their desk a certain number of times to complete their task. A small collection of MCs and AEs are defined to model this behaviour (see Appendix G). Affective actions of the user are specified in the task hierarchy rather than in a state model; the effect of the desktop releasing an object to the user is described in the event handler. The attribute 'action Count' is used to monitor the number of times the user has picked up an object and is used in a rule for the task model.

6.1 TASK HIERARCHY

An ISML task tree consists of one or many task nodes, each of which may refer to zero or many actions performed by one object upon another. These action sequences may be repeated many times, exiting on the evaluation of 'true' of either an MC test or a conditional statement relating to an object's attributes. One or more 'node lists' sequences defines a series of nodes by specifying their ID, followed by either 'ENABLES' (completion of this task is then followed by the next in sequence) or alternatively 'OR' (this task is optional and the next in sequence is immediately available).

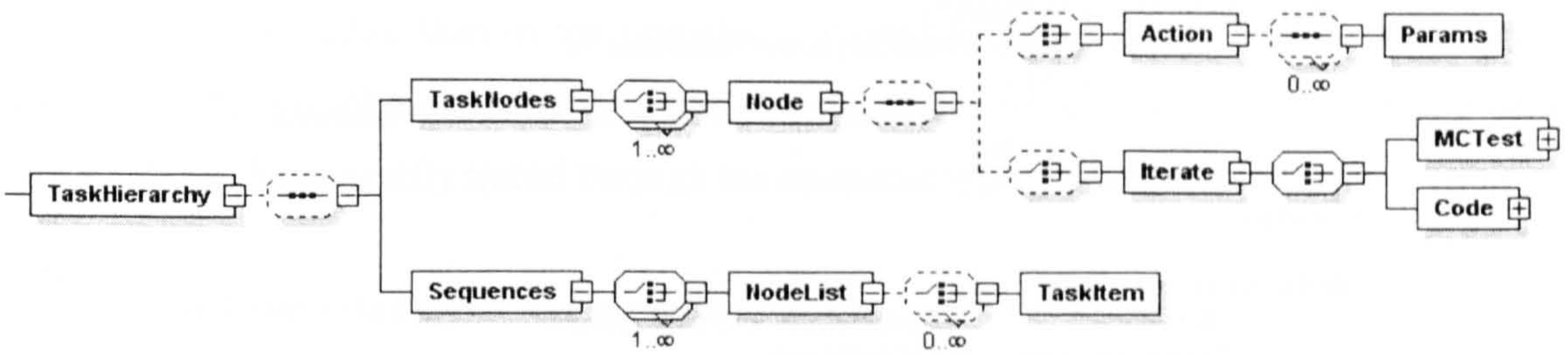


Figure 39 ISML Task hierarchy

Each node list may optionally specify a parent node; only one node list may contain no parent indicating that it is the tree root. In specifying the simple 'pick up and drop' task, the task tree in Figure 40 is constructed. Arranging a desk is simply a matter of picking an item, moving it and then release it.

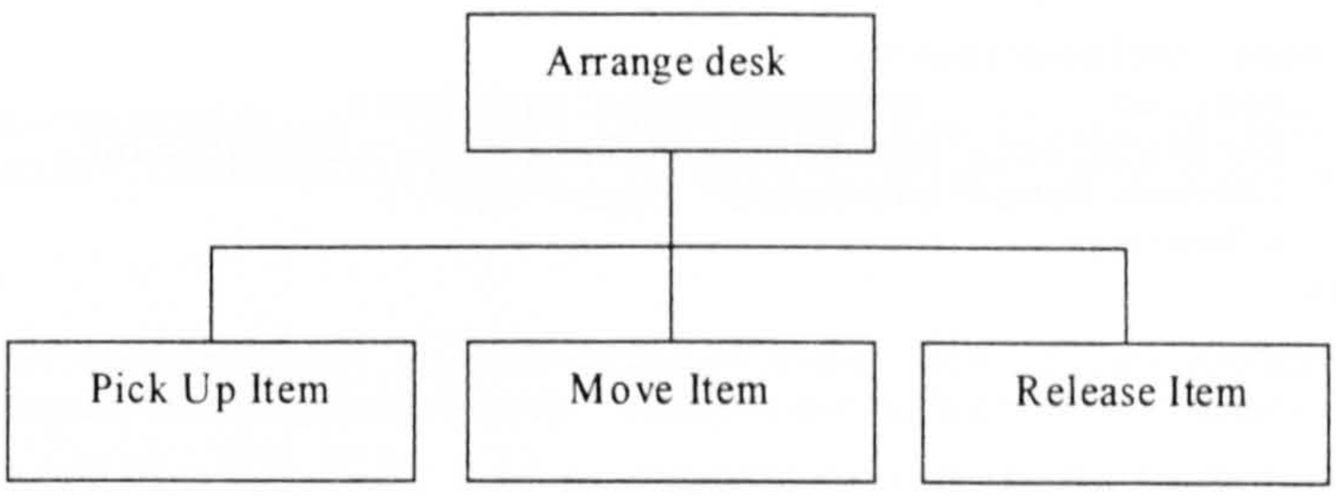


Figure 40 Simple task tree

Below is the complete specification for the task tree specifying nodes top-down, left to right. Actions are specified for the user in nodes 'Pick Up Item' and 'Release Item' whilst iterating conditions maintain task persistency for moving an item (the user has something in his/her hand) and completion of the overall task (the user has moved an object more than twice).

```
<TaskHierarchy>
        <TaskNodes>
                <Node Name="ArrangeDesk">
                        <Iterate SourceTaskObject="taskUser">
                                <Code>
                                        <Statement>(taskCount > 2)</Statement>
                                </Code>
                        </Iterate>
                </Node>

                <Node Name="PickUpItem">
                        <Action         SourceObject="taskUser"
                        TargetObject="taskDesktop"
                        TargetAE="taskPickUp"/>
                </Node>

                <Node Name="MoveItem">
                        <Iterate SourceTaskObject="taskUser" Logic="NOT">
                                <MCTest>
                                        <IsEmpty>
                                                <Set>
                                                        <MC>userHolds</MC>
                                                </Set>
                                        </IsEmpty>
                                </MCTest>
                        </Iterate>
        </Node>

        <Node Name="ReleaseItem">
                        <Action         SourceObject="taskUser"
                        TargetObject="taskDesktop" TargetAE="taskRelease">
                        <Params Name="userHolds" Type="SET"/>
                        </Action>
        </Node>
</TaskNodes>

<Sequences>
        <NodeList ParentNode="ArrangeDesk">
                        <TaskItem NodeRef="PickUpItem"/>
                        <TaskItem NodeRef="MoveItem"/>
                        <TaskItem NodeRef="ReleaseItem"/>
                </NodeList>
                <NodeList>
                        <TaskItem NodeRef="ArrangeDesk"/>
                </NodeList>
        </Sequences>
</TaskHierarchy>
```

6.2 METAPHOR MAP

Mappings between the task and interface model are specified through the metaphor map. Two types of mapping may be specified: mappings between objects and mappings between action-events.

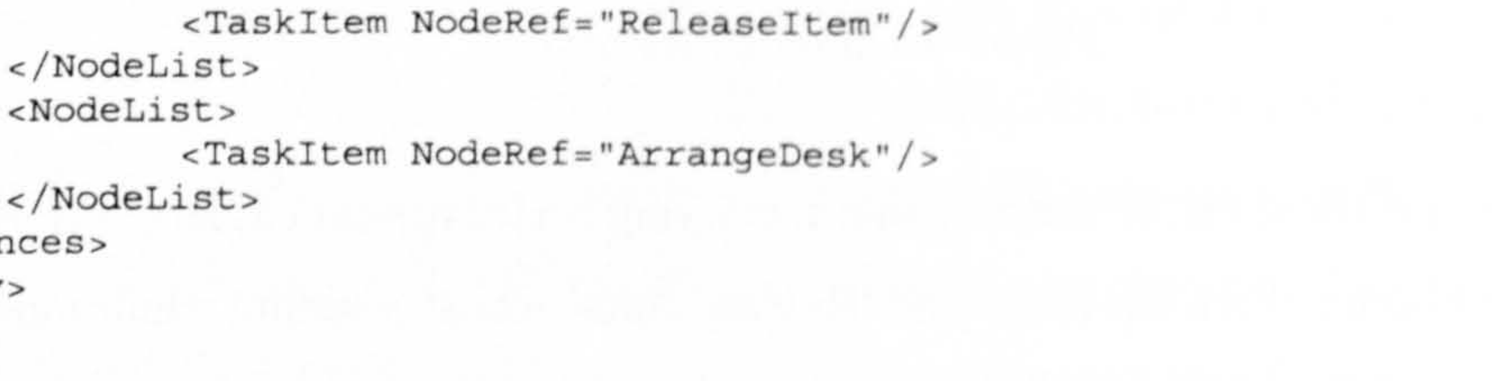


Figure 41 ISML Metaphor map

Both types relate task objects and actions to the *interactor* specification. The advantage of mapping to the interactor abstraction rather than to the underlying metaphor model is two-fold. Firstly, tasks are related directly to a design solution rather than just the metaphor and secondly, the relationship between task and metaphor can be indirectly traced through the interactor types used to enact the interaction.

```
<MetaphorMap>
<ObjectMap      TaskObject="taskDesktop" TargetInteractor="desktopInteractor"/>

<ObjectMap      TaskObject="taskFile" TargetInteractor="fileObjectInteractor">
                      AttributeMap Source="taskFileName" Target="fileName"/>
</ObjectMap>

…
```

Object maps indicate analogies between task objects and interactor based representations. These may be simple name-space mappings (as seen in the first example above) in which no specific abstractions are mapped or more detailed relationships in which attributes or specific states are considered equivalent (see second map).

```
<ActionMap      TaskObject="taskDesktop" TaskAE="taskPickUp" Qualifier="EFFECTS">
<Implements TargetInteractor="desktopInteractor" TargetAE="RequestOwnershipAction"
Qualifier="EFFECTS"/>
</ActionMap>

<ActionMap      TaskObject="taskDesktop" TaskAE="taskRelease" Qualifier="AFFECTS">
<Implements TargetInteractor="desktopInteractor" TargetAE="AcceptOwnershipAction"
Qualifier="AFFECTS"/>
</ActionMap>

<ActionMap      TaskObject="taskDesktop" TaskAE="taskRelease" Qualifier="EFFECTS">
<Implements TargetInteractor="desktopInteractor" TargetAE="AcceptOwnershipAction"
Qualifier="EFFECTS"/>
</ActionMap>

        …
</MetaphorMap>
```

Action maps relate operations in the task world with those interactively executed through interactor types. In the above example fragment, the AEs formed for the 'task world' desktop are mapped to the interactor equivalents – note that the *affective* and *effective* qualifiers must be specified for the target and the source.

7. Discussion

During the course of this chapter, a simple, direct manipulation system and related task model was specified. The example has been used to illustrate most of the features of the ISML framework, but it cannot be used to validate the language convincingly. In chapter 6, the results of an initial prototyping phase of a larger scale project is used as the basis for a more realistically scaled project. Despite its relative simplicity, the example provided here can be considered as a plausible basis for more realistic 'desktop' systems since the basic mechanisms for pointing, selecting, dragging and dropping have been

demonstrated. In constructing this facility, a number of issues regarding the use of ISML, particularly with respect to the separation of metaphor from interface design, have emerged.

One of the most striking features of this example specification is the number of pages required to describe a very simple interactive system which is 1138 lines long (see appendix G). There are various possible explanations to this problem, including notational issues and ISML concept limitations. A penalty for using XML to both describe the ISML grammar and write specifications is readability of the text. This is particularly problematic with respect to specifying mathematical and logical statements since they do not lend themselves well to the declarative form of XML. Balanced against this, XML is useful since a growing number of writing tools and parsers already exist which makes the ISML framework more accessible to the development community. Basic 'well-formed' tests are also built into the algorithms that verify the grammar of ISML and the documents written using its framework. Such tests are useful in automatically determining whether the minimal elements required in any ISML specification exist and, to a certain extent, of what type they should be.

At present, ISML does not support some of the more advanced data concepts enjoyed by other specification and programming languages, most notably data structures and object-orientation (with the exception of meta-object to interactor inheritance). This inevitably leads to unnecessary redundancy and is an issue that needs to be addressed in the future - device, component, meta-object and interactor definitions could all be potentially more manageable by implementing this feature. State models would also benefit by the extension of the statechart framework Harel (1987) in which parallelism and state-expansion also reduce scaling problems. Currently, ISML only supports parallel, independent state models without state expansion (i.e., states may not contain child states).

With respect to the de-coupling of the metaphor model, the simple, direct manipulation example highlights an uncomfortable conceptual stance in that some consideration to implementation creeps into the meta-object part. Specifically, two AEs 'Render Action' and 'Give Me Pixels' were declared (although not actively used) and assigned in the semantics for each of the meta-objects. Neither of these AEs play a relevant role in what is arguably the core of the direct manipulation environment, but both are required to provide a framework within which the interactor solution can provide a realisation. This problem may grow as the size, number or complexity of the metaphor models increase, leaving a question mark over the practicality of abstracting a metaphor model without specific reference to its eventual realisation.

8. Conclusions

Contemporary research in the model-based community is working toward a set of models that capture useful and diverse aspects of interactive system design. Ideally, each contributing model is at a level of abstraction suitable for its intended developers, has minimal commitment to implementation issues and can be integrated into an over-all framework for interactive system design. ISML contributes another perspective on integration of models, but does not solve this problem. There are many aspects of the language which require refinement or extension; some perspectives on interactive system design are entirely missing, such as user models or contemporary software engineering concepts such as class hierarchies and entity relationships. It may be possible to include these into the framework at a later date, but before doing so further investigation is required to determine the validity of the basic premise that explicit metaphor modelling is possible, practical or indeed, desirable.

BLANK IN ORIGINAL

CHAPTER 5 Urban Shout Cast case study

1. Introduction

This chapter describes a six month case study conducted with two final year undergraduate software engineering groups, each independently working on a multimedia project entitled 'Urban Sound Cast' (USC). The USC case study was created to evaluate ISML within the context of a more realistic user interface design project than the example previously considered. The purpose of the case study was two-fold: firstly to gather design data for the specification of a larger user interface design (see chapter 6) and secondly to examine the utility and practicality of ISML within a real software design project. During the course of the case study a series of meetings, held separately with each group, took place to discuss design decisions and elicit data to be used for the specification of the USC system using ISML. Towards the end of the project, the engineers were encouraged to discuss their experiences with the project, specifically relating to the use of ISML as a means of conceptualising and specifying their interface design. What follows is a brief outline of the USC project proposal and the participants; a review of the case study life cycle; a qualitative analysis of the project based in grounded theory Glaser and Strauss (1967) and a discussion of the lessons learned from the case study. The specification of the USC prototype is addressed in chapter 6.

2. Research method

The interface specification meta-language is new and unique in its organisation of HCI design concepts; as such it can be regarded as a 'prototype' language that requires testing and evaluation. Presently, ISML is in almost the earliest stage of development that any such enterprise can be. Even so, the XML expression of the ISML framework is already complex and specifications for even small examples are substantial (as chapter 4 demonstrates). Arguably, the concepts and relations in the ISML framework are independent of their expression in a language (indeed, a BNF version of ISML was created based on the ANSI C grammar Degener (1995), but later abandoned). Given these constraints, it is important to consider the potential evaluation strategies available and identify that which is most appropriate for the evaluation of ISML.

A range of evaluation techniques is already used within the HCI community including experimental laboratory, field-based observation, heuristic and model-based evaluation Dix et al. (1998). Laboratory experimentation is inappropriate simply because the number of confounding influences is enormous in a real-world design context. Whilst progression in the MB-UID community is considerable, rules have yet

to appear that govern the development of model-based languages for GUI design, ruling out both the heuristic and model-based approaches.

In evaluating model or interactor based specification languages, it is typical to find examples in the literature of small-scale component examples (such as input device simulation or simple buttons and forms) to demonstrate the potential application of the language or toolkit. The ability to verify reachability, feedback and reversibility in a design is a desirable analytical feature during specification of an interactive system Jambon et al. (1999). A number of the languages reviewed here have a formal underpinning (such as Petri-net models or a temporal algebra, see chapter 3) that offers verification methods to test such properties of potential models. Conceivably, some mathematical verification of the properties of the ISML could be applied but this approach would have little to say regarding the usability and practical application of ISML concepts within a real software development project.

Whilst larger case studies can also be found in the literature Markopoulos et al. (1999), Sage and Johnson (1998), reports on the use, impact and acceptance of specification languages and model-based approaches within a software engineering environment are few Markopoulos et al. (1998). Arguably, a design notation, regardless of its formal rigour, will be of limited benefit to a software engineering team if they are unable to express their ideas whilst using the formalism. There may be many potential reasons for this, including the need for special training, a lack of support for particular concepts within the design notation or problems with the level of abstraction or complexity.

For these reasons research objective 3, an assessment of the effect that an abstracted metaphor layer would have on design, was chosen rather than analysing the language itself. It can be further argued that an evaluation of the current structure of the ISML with designers also provides important feedback with respect to further development of the language. Without feedback from such an evaluation, it would be impossible to tell whether or not the concepts embodied in the language are useful in GUI design. Based on these evaluation requirements, a qualitative, case study based approach was identified as being a potentially fruitful means of gaining insights into the use of ISML within a software engineering project.

3. Qualitative research method

3.1 Brief introduction to qualitative research

The description and theory of some aspects of human behaviour, particularly where there are sophisticated interactions between individuals in complex environments, led to the evolution of qualitative research methods in the psychological and social sciences. Qualitative research attempts to build theories of human knowledge and behaviour in relation to a social context where it is unrealistic to

apply classical quantitative methods. It is argued that shared social concepts, behaviours and processes exist in a real sense and may be 'captured' by a variety of elicitation methods including semi-structured interviews, observational recording and archival analysis Fielding and Lee (1998). Analysis and theory building methods in qualitative analysis led to different forms of 'knowledge claims' and it is not surprising that there is still considerable controversy between these two outlooks on research Pidgeon (1996). However, such a philosophical discussion is beyond the scope of this thesis. Despite its criticisms, qualitative methods (such as ethnography) have been accepted in HCI Carroll (1997) and are beginning to gain acceptance in the software engineering community Avison et al. (1999), Viller and Sommerville (1999).

Little theoretical work exists on the usability of many of the specification languages and tools particular to user interface design, so a formalised methodology that naturally lends itself as a basis for analysis was not available. However, 'action research' as a qualitative method for understanding software engineering activities in context through participation and iterative reflection has been advocated by some researchers Avison et al. (1999). This method demands an actively participatory design role on the part of the researcher that enables him or her to reflect on context rich data within the development environment. The general approach suggested by the action research method was considered a useful heuristic in shaping the design discussions and general analysis method during the case study. Action research promotes a collaborative, iterative and interactive approach to gathering data on field-based design studies; the analyst and design team work together in understanding and solving problems. However, full design and development participation on the part of the interviewer was not possible since the role of the interviewer was strictly as the 'customer' of the product. In the light of this, a more passive qualitative methodology was adopted: the grounded theory approach Pidgeon (1996).

Historically, grounded theory was developed as a reaction to the quantitative inductive approach take by the social sciences in the early part of the twentieth century Fielding and Lee (1998). Grounded theory is an iterative process during which theories are generated and refined as textual data are codified and organised into categories and relationships between them emerge. A variety of strategies for coding and theory building can be found in the literature Kelle (1995), but the principles of description (often referred to as open coding) and relationship building (referred to as axial coding) remain a constant feature throughout. It is the nature of grounded theory to work within a general subject structure but with a degree of flexibility that will allow the researcher to alter the direction of the interviews with subjects where it is deemed appropriate. This gives the researcher the ability to reduce the time spent on unfruitful avenues of exploration or to pursue more useful topics of discussion; a more detailed account of this process is given in the analysis section.

3.2 ISML qualitative research questions

One of the strengths of the grounded theory method is that it allows the analyst to explore the behaviour of a person or a social unit where there is no previously prescribed theoretical framework upon which a data capture can be based. This is useful in the specific case of ISML, since it is new to the MB-UID community. However, at least some basic structure must be defined such that the qualitative evaluation has a focus likely to generate meaningful data with respect to the use of ISML within a user interface design project. To this end, two broad directions for analysis are posed (and subsequently refined) to address research objective 3:

Question 1: What are the reactions of developers to the use of ISML?

Questions 2: To what extent does ISML capture a design?

In this chapter, analysis question *one* is addressed through the application of a qualitative research method used to examine the USC case study. Analysis question one is further sub-divided into two parts:

Part 1: How is a user interface metaphor developed within the ISML framework?

Part 2: What is the perceived utility and practicality of the application of ISML to design?

Sections 5.1-5.3 describes the analysis of the USC design meetings, examining how design ideas were expressed and developed within the ISML framework, by each group. Issues regarding utility and practicality are examined in section 5.4. Analysis question *two* is addressed in chapter 6.

The USC case study commenced adopting the grounded theory approach using a series of semi-structured meetings. These were captured on digital mini-disc, converted to MP3 format audio files and transcribed using custom software. To provide a structure for qualitative analysis, the ISML framework was used as a basis for discussion during the course of the project, the expectation being that the framework would present a platform upon which to relate the engineer's understanding of the design to concepts provided by the ISML. Two of the major qualitative analysis software packages were reviewed in selecting a tool to aid the analysis of the design meetings, these were *N6* QSR (2002), and *Atlas.ti* Muhr (2002). These packages share many features, however Atlas.ti was chosen because of its unique, graphical handling of source documents and codes.

4. USC case study background

Final year undergraduate students on the software engineering management course at Bournemouth university are expected to participate in a group project (involving four members) in which they bid for ownership of a six month software project, outlined in brief by a pseudo company customer. The winning groups work with the customer through requirements, specification, prototyping, design and testing phases of the product. The company in the case study consisted of the author, acting as design and software engineering liaison, and a university member of staff acting as manager. Each group of four students was expected to have an understanding of the fundamental principles of HCI and have had experience of prototyping user interfaces with a variety of user interface design tool kits. It was anticipated that whilst both groups had training and industry experience of software engineering[19] as well as some basic HCI knowledge (both groups of four had one member who had chosen a final year module in usability engineering), neither would have had extensive knowledge of interface specification frameworks.

4.1 USC design problem

As a larger scale evaluation of ISML, it was considered a general requirement that the design problem should necessarily feature relatively large degrees of freedom with regard to interface design, implementation and most particularly, metaphor development. It has already been established that metaphors are devices used to employ knowledge from one domain to explain the features of another and it is for this reason that a problem domain was chosen that already has many potential analogical counter-parts in the real world. In addition to these considerations, it was also desirable for the target problem domain to be relatively novel so that existing, conventional solutions were not immediately apparent.

The central tenet of USC is that of a virtual radio broadcasting station, maintained by a number of networked (LAN or Internet) PCs and controlled by co-operating users (or 'virtual broadcasters') within the virtual environment. Virtual broadcasters working within this environment co-operate to produce a show that is streamed to listener clients on the local area network or Internet. A user may act as either a DJ, or as a producer – although only one producer role is permissible. Both DJs and producers were expected to be able to play music, or produce sound by other means. However, the producer is said to have executive control over what audio is broadcast to air, and at what point in time this occurs.

---

[19] Students on this course spend their 3rd year in an industrial, software engineering placement

At the outset of the case study, both groups were made aware that their main goal was to develop a prototype system and that full audio streaming functionality was not a priority. In addition to this, a requirement for this project was the development of novel and creative user interface solutions that avoided the ubiquitous WIMP interface style. Both USC groups (referred to here as group 1 and group 4, this numbering is an artefact of the SEM course programme) were presented with a project that had no previous development history. In light of this fact and the absence of any specific interface design from the customer, both groups began the project with a domain investigation through artefact collection (photographs of existing hardware) and consultation with individuals working in local radio broadcasting groups. Their initial investigation served as a basis for analysis of requirements and early design discussions.

4.2 USC project life cycle

Throughout the project life cycle, the author (acting as USC customer) held a total of seven meetings[20] with each group to discuss general project matters (such as progression and technical issues) and phase-specific issues. At the beginning of the project, each group estimated milestones that reflected the waterfall development model. However, it became clear that by the end of the life cycle, both teams were, in reality, adopting a strongly prototype oriented approach to design Sommerville (2001). Initially, it was proposed that in order to accommodate the design teams' milestones, conventional requirements and specification stages would be executed, followed by design and development phases during which an ISML model would be constructed. Running in parallel with both design groups' development phases, the interviewer conducted three main activities (see Figure 42).

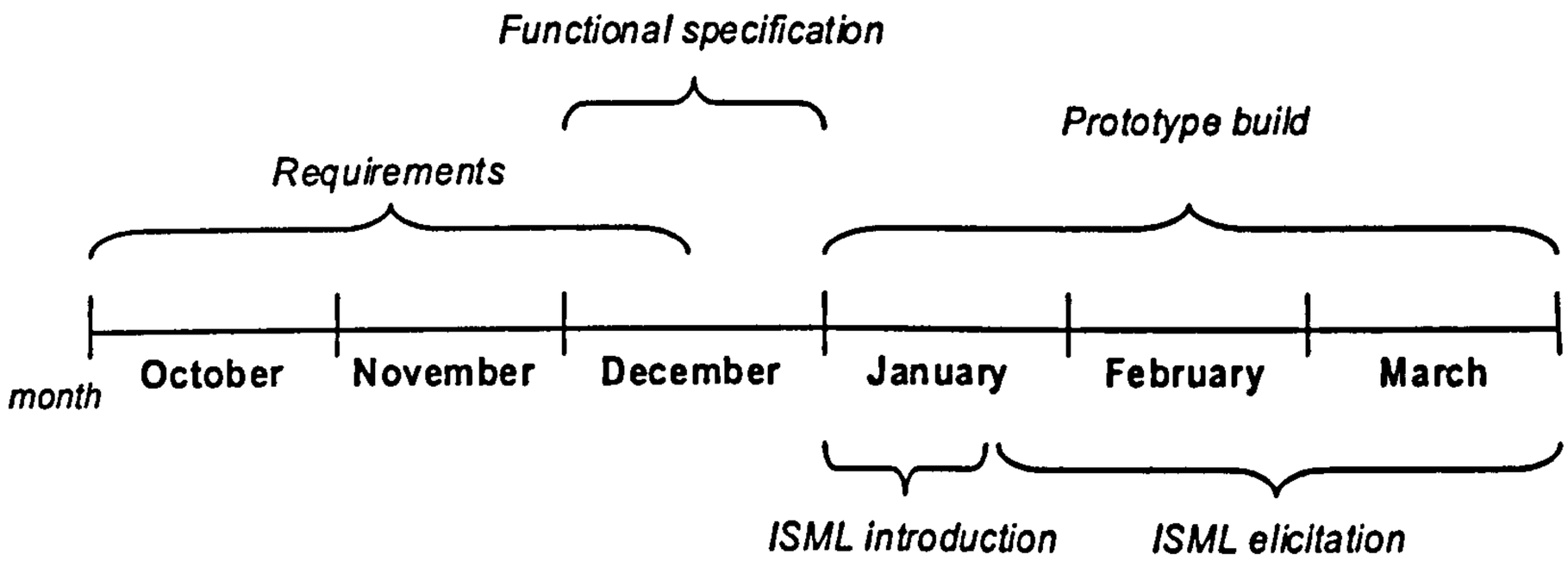


Figure 42 Interviewer activities

4.2.1 REQUIREMENTS

[20] These occurred in varying frequency according to project phase (varying from 1-3 week intervals)

A series of meetings was held to develop and refine requirements for the prototype system in which the high level functional and technical aspects of the system were discussed. In addition, the role of users as DJs or producers and their interaction with the system (in terms of expected function) was broadly and informally discussed. At this stage in the design life cycle, both teams were eliciting requirements from the interviewer acting as the customer during which the design and deliverables of the prototype were negotiable.

4.2.2 INTRODUCTION TO ISML

Subsequent to requirements gathering, both groups were introduced to the ISML framework over two meetings. The high level framework was introduced followed up by a more detailed discussion of meta-object, interactor and task model parts. Both teams were able to follow the ISML introduction but reacted strongly against the request that they attempt to write parts of the ISML specification themselves with fill-in forms provided by the interviewer. In each case, the team felt that a considerable amount of time and effort would be required to complete this task (unsupervised by the interviewer) and that it would significantly impact on their other development activities. For this reason a compromise had to be negotiated – each group agreed to engage in elicitation meetings in which the interviewer would elicit ISML models through a series of semi-structured interviews.

4.2.3 ISML ELICITATION

This last phase was used to gather design data suitable for the specification of the USC system and elicit qualitative responses from the engineers regarding their understanding and reaction to the ISML framework. Each group was taken through the same procedure (outlined in appendix B) in which data for task, meta-object and interactor abstractions were captured in sequence[21]. At the beginning of each session, the group was briefed on the nature of the abstraction they were constructing and reminded that the exercise was neither a test of their intelligence nor a design assessment exercise.

---

[21] Although for both groups there were occasions when the discussion temporarily returned to earlier elicitation stages

A high-level task model was interactively constructed by each group based on their experiences with the radio station interviews they conducted at the beginning of the project. At this stage, the 3 main objectives were to:

1. Generate a task hierarchy

2. Identify the associated task objects

3. Verify any specific conditions or constraints associated with the tasks

A basic set of goals was established and then progressively refined into sub-goals and eventually actions. Domain objects and potential actions directed toward each object from the DJ environment were then enumerated. These objects and actions were then married up with the leaves of the task tree to specify potential relationships between task and objects. Finally, each object was examined to identify any attributes or states that may constrain the behaviour of either itself or other objects (including for example, the DJ) within the task model. Any such occurrence was referred to as a 'stop-iterate' condition and identified as a situation in which an object and an associated task node modified or qualified the normal sequence of actions.

Following the task model, the metaphor designs developed by each group were discussed. The following objectives were addressed at this stage:

1. Identify principal metaphorical objects for use in USC

2. Identify actions associated with the objects

3. Identify potential mapping-constraints between objects

Initially, the group gave an informal description to each object identified. This included its basic features, states and likely metaphorical mapping to the task model. Following this, relevant actions that would be directed toward the object were considered. For each of these actions, two further issues were explored: 1) the focus of the action onto possible smaller, sub-ordinate parts of the object and b) any mappings or constraints that were enforced as a result of that action. In this way, potential areas for expansion or refinement were identified and the semantics of the interaction between objects revealed. An inverse

mapping of actions then took place in which those actions previously identified were mapped to their potential sources (such as the DJ). Any resultant mappings or constraints for the source (the target having been previously considered) were then specified. Having completed this 'first cycle', the attributes, states and actions were then verified for each object and any sub-ordinate objects associated were subjected to the same process described above.

In the last stage of the elicitation (conducted during the closing phases of the project) screen-shots from the emerging prototypes were examined to identify the interactors used to implement the metaphorical objects developed by the team. The objectives here were to:

1. Specify input and output devices

2. Identify high-level interactors that implement the principal meta-objects

3. Identify the interactions used to enact actions

Objective one identified the physical hardware used by the prototype and the software development environment used by each group to implement their system; this allowed the author to anticipate both the basic types of interaction and the rendering capabilities of the final design. Each interactor in the screen shots was labelled and associated with the meta-object model (if possible). Inputs from the user (mostly mouse based) and virtual actions executed in the implementation (mouse pointer movement, button clicks and direct manipulations for example) were subsequently associated with the underlying meta-object model, where appropriate. In a process similar to the meta-object refinement (see above), sub-ordinate interactors that received the specific focus of an enacted action where identified. Finally, those actions and objects described within the group's implementation were associated with the task model in an attempt to bind tasks, interactor implementation and the underlying meta-object model.

In the latter stages of the meta-object and interactor design discussions, both groups were asked at the end of each meeting to freely express their reactions to both the ISML framework and the specification elicitation process, this is discussed in more detail in section 5.5. Throughout the course of these meetings, data was captured both in the form of audio on mini-disc and on paper – mostly written by the interviewer, but occasionally annotated by the group. After each meeting, the models created were converted into electronic form by the interviewer. The purpose of this was to generate structured 'intermediary' design documentation for both the interviewer and the groups. This intermediary documentation is a high-level, human readable description of the ISML concepts used by each group to

describe their designs. Its translation into the formal ISML expressed using XML was performed 'off-line', after the elicitation process had completed. In doing this, there was also the opportunity to identify potential areas that required clarification or exploration for the next meeting. After completing the elicitation, the intermediary models were compiled into a single document that was supplied to each group; extracts from these documents are also used to discuss the designs of the USC prototype in chapter 6 (see appendix L for the complete intermediary documentation).

5. Post-project qualitative analysis

On the project deadline, both groups produced executable prototypes, user manuals and software engineering documentation. The ISML elicitation produced a high-level task and meta-object model, marked up screenshots indicating interactor implementation of the meta-models and a set of associations between each. Eight meetings with each group were transcribed into separate documents totalling approximately 37,000 words each. The large size of the transcribed data meant that some limitations had to be imposed on the scope of the analysis to make the exercise practical. A preliminary examination of the transcriptions suggested that phase two of the interview (ISML introduction) had little to contribute since it was a mostly one-way tutorial of the language given by the interviewer. Phase one promised a greater richness than this, but since it has a large functional and technical component and no discussion set within the ISML framework, this too was eliminated leaving the final elicitation phase as being the most likely to reveal interesting aspects of the design process.

5.1 Open coding

At the beginning of the qualitative analysis, it is typical to begin the process by looking for general structures and categories to start to make sense of the data Fielding and Lee (1998). Simple mark-up categories were applied to delineate basic features of the interviews including speaker identification, questions and 'conjunctive phrases' (phrases that are just incidental, or simple affirmations or disagreement statements without content). Following this, the classic problem of establishing categories of phrases Kelle (1995) emerged: attempts to categorise phrases at a high or low level proved too general and time-consuming respectively. As a result of this process, two 'middle ground' general categories suggested a more useful starting point. Phrases that discussed 'media objects' (such as CDs, tapes, media players and tracks) were referred to as *media phrases* whilst those which discussed DJ environment related objects (such as the DJ or producer, the mixing desk and play list) were referred to as *DJ phrases*. Having established these categories, it was possible to construct inventories of nouns associated with both phrases and, using Atlas.ti, it was then possible to search through all phrases to verify the phrase types already marked up and identify those that were missed.

It is common practice to make notes during open coding, and it is from the initial exploration of media and DJ phrases that other open category codes began to emerge, which include:

Phrase code category	Description	Example
Media	Discussion of media objects	Is there a minidisk device? Yes. And perhaps a tape.
DJ	Discussion of DJ objects	The actual list itself, what does it comprise of, just a list of DJs in order? Yes. I mean, in the real world, you might have time slots as well...
Interface component	References to interface components	We've got this scroll bar here as well. These adverts are draggable.
General interaction	References to artefacts that can be manipulated	I don't know....I would have thought I don't know, in a real DJ environment, you'd have a button which you press? Perhaps you have a twiddly knob where you tune into things?..
'Real world'	References to real world objects outside of the specific locus of the DJ environment	I think she was saying that their adverts exist in books. Or am I making a mistake there? We represent it as adverts sitting in a book, and then you flick over pages.
Task model	References to discussions of the task model	And then we've got this thing called play media, we need some action for that. You need to find the track, if we're still putting that in the..... Find track ...
Modelling	References to ISML modelling terms	OK. So, let's call it music stopped. And, that particular condition is a condition of the track itself, isn't it?
Meta-design	Discussion of the elicitation and design process	So the mixing desk, if we gave it a brief description, it would be? What form of description?
Abstract	References to abstract nouns such as 'sources' or 'inputs'	Would you be switching on sources, on and off? Or is that the function of the slider? Sorry, I'm trying to separate the implementation from the ..
Prototype	References to the prototype design or implementation	This is still in the metaphorical thing? That's right, not necessarily what you've implemented.
Programming	Programming terminology	No, they're not? OK. So, what do those tracks, they represent exactly the same ... No, they're not MP3 or WAV data, they have a pointer to them like a file name perhaps.
Software Engineering	References to the software engineering process	Is auto DJ part of the implementation? You'll have to check your requirements for that.
Computer software	References to computer software	I don't like the word write, because it could be printed. No I don't. Exactly. That's where I was going with it, I mean you probably wouldn't write it nowadays. You'd stick it in a spreadsheet.
Computer hardware	References to computer hardware	When you start the machine up, you need to index all the MP3s, all the ID3 tags and all that. So that the user doesn't think it's not doing anything, we bring that up.

Table 2 Open coding examples

Not surprisingly, any one phrase may have several over lapping codes. Finally, to get a broad picture of possible interesting relationships between the phrase categories, a matrix of co-occurrences between the phrase types and the appearance of individual phrase nouns from a specific category was assembled (see appendix B).

From this summary of relative frequencies (across both groups) it is possible to observe a few general distribution patterns. Most obviously, the highest degree of crossover in discussion is between the DJ and media phrases. Notably high co-occurrences of DJ related nouns within modelling, abstract, task and meta-design phrases can be compared with fewer relating to the media objects themselves. Although less frequent, a cluster of ISML modelling phrases coincide with abstract phrases, tasks and high-level design process phrases (meta-design phrases).

5.2 Axial coding

The frequency analysis suggests a more detailed examination of primary media and DJ objects between themselves, and with a secondary view to tasks and ISML modelling abstractions. The secondary view of tasks and modelling abstractions was broken down by using the three elicitation stages conducted during the case study as focuses for analysis. In this way, a separate view in each case can be compared and contrasted. To do this, a preliminary list of the major objects discussed by both groups was created and all individual references to them marked up as separate open codes. For each group, and for each elicitation phase, a systematic search was conducted on each of the major media and DJ objects. A conceptual graph was then constructed using the graphing facility in Atlas.ti in which quotes from the group are clustered around the object codes, linked by arcs (see appendix D).

The nature of each object and its relationship with others can then be compiled from the linked quotations whilst at the same time capturing other interesting features that relate specifically to the elicitation phase. In the following sections, evidence collected from the graphs for each of the major objects discussed by each group is summarised in table form, with individual references to transcripts indicated by a bracketed number. This number refers to a quotation stored in the Atlas.ti files, an example of which can be seen below:

> (2:426)
> "Yes, you press the button and the button physically lights to indicate that that's..And perhaps even the sliders, the sliders would activate a light..."

For brevity, quotations are summarised in this numeric form and presented in tables in the following sections for tracibility. For a more detailed explanation of the data, generated during the analysis and stored as notes in Atlas.ti, refer to appendix E.

5.2.1 THE DJ

Group 1	Group 4
Task phase	Task phase
(1:1330) (1:1323) (1:1304)	(5:1439) (5:1438) (6:564) (5:1434) (5:1436) (5:1428) (5:404) (6:564)
Meta-object phase	Meta-object phase
(2:1759) (2:1762) (2:1767) (2:1766) (2:1761) (2:1763) (3:1160)	(7:855) (8:26) (7:159) (8:1301) (7:117) (8:231). (7:854) (7:853) (7:852) (8:1299) (8:1300) (7:125).
Interactor phase	Interactor phase
(4:960) (3:1174) (4:959) (4:957)	(8:1316) (8:358) (8:1321) (9:440) (8:1317) (8:1319) (8:1311) (8:1314) (9:439) (8:1318) (8:1315)

Table 3 DJ evidence

5.2.1.1 Group 1 data

The DJ as a modelled entity is not highly specified throughout all the elicitation phases with group 1. Most of the discussions regarding the task model referred implicitly to actions executed by the DJ upon objects within his/her environment. The implicit actions of a DJ upon objects continues in the metaphor elicitation – a small number of potential objects (such as paper letters) is suggested as mechanisms for enabling some of the tasks that are later not supported in implementation. Other than actions, the only significant model for the DJ is that of a hierarchy of responsibility for channelling music out to air: broadcast sound is initiated by the DJ and then sent out to air by the producer. The interactor elicitation phase reveals small changes in the graphical appearance of objects that make the DJ, an otherwise implicit entity in the over-all prototype, more 'visible'. This includes differentiation of the mixer desk (for DJ or producer) and the appearance of DJ names under sliders.

5.2.1.2 Group 4 data

In contrast to group 1, the DJ stands out as a distinct entity from the beginning of the elicitation process that not only exercises actions, but also has properties and relationships with other parts of the model being discussed. During the task phase, the DJ is considered to be on or off air, have audio output and volume and to sit within a broadcast hierarchy of a similar type discussed above. Group 4 recognise that mixing DJ audio is a part of the task model but stipulate in advance that they would not be supporting this aspect of the prototype[22]. Another assumption suggested by this group was that of the role of the play list which rather than acting as a guide for the DJ, is viewed as a prescriptive list from which the DJ may not

[22] In fact, in the end, both groups did not support continuous mixing in their prototypes.

deviate. Phase 2 of the elicitation process unpacks a "rather complex" DJ object that shares many of the properties already suggested in the task model and is also supplemented with a 'profile' object. Initial discussions of the profile by group 4 suggested that the profile was a container for the properties of the DJ object – including his/her appearance within the environment. This de-coupling of the DJ object from its properties was resolved later as implementation detail rather than important distinction at the metaphor level. The final interactor elicitation sheds some light on the earlier contention between the DJ and DJ profile distinctions: the DJ object is graphically represented in the DJ booth but is non-interactive; whilst in the producer's environment the DJ object is an entity that is engaged by the producer. In the former case, the graphic provides a visual representation of the DJ's properties (a 'profile'), whereas the latter DJ object (visually identical) is considered as an entity that interacts with both the producer and the microphone.

5.2.1.3 Discussion

A clear difference between groups is apparent in the visibility and definition of the DJ within the metaphor and interactor models. In all but small changes between producer and DJ interactor displays, group 1 makes little attempt to include the DJ as an explicit part of the metaphor model. In comparison, group 4 combines the views of a USC user acting as a DJ (or producer) with an object that acts like a passive 'avatar' – an interactive entity that represents, but does not act on behalf of, the logged on user. Arguably, group 1 does present images of the logged on users on a notice board object at the back of the producer's room and as labels on the mixer desk. However, these indications of DJ presence are modifications of other objects, rather than the appearance of a DJ object itself. Whilst this dichotomy of views allows group 4 to strengthen the presentation of a shared environment to USC users, it also introduces a metaphorical paradox: a DJ is simultaneously present in both his DJ booth and in the producer's room.

5.2.2 THE PRODUCER

Group 1	Group 4
Task phase	Task phase
	(5:1424) (5:1441) (5:1435) (5:497) (5:1437) (5:1443) (5:1444) (5:404) (5:1001) (5:1440) (6:565)
Meta-object phase	Meta-object phase
	(7:125) (7:137) (7:858) (7:136) (7:856) (8:1301) (8:1302) (7:859) (7:850) (8:1298)
Interactor phase	Interactor phase
	(6:358) (8:1322) (8:1316) (8:1319) (8:1320)

Table 4 Producer evidence

5.2.2.1 Group 1 data

Constraints on the meeting arrangements combined with the time spent focusing on other aspects on the prototype resulted in virtually no time to focus on the producer's part of the system during discussions

with group 1. However, aspects of the producer's role in the system can be found in the specialisation of other objects within the environment, most particularly with respect to interaction with the mixer desk (see section 5.2.7).

5.2.2.2 Group 4 data

The principal role of the producer in the task model discussed by group 4 is that of a controller. He or she may place DJs on and off air and modify the audio properties of the sound generated by the DJ (expressed as a hierarchy). A high-level discussion of communication mechanisms between DJs establishes the assumption that DJs are located in different rooms and can communicate visually. Two types of list are introduced in the task specification for the producer: collections of play lists and a show list. Group 4 use terms familiar to them (a 'browser') to describe how a producer might collect and arrange the play lists of particular DJs, which, allocated with time-slots make up a list of DJs in a show list (see section 5.2.10). A refinement of the basic task model is elicited in the meta-object phase in which the mechanism for placing a DJ on or off air is described (see section 5.2.8 for more on this). Group 4 makes a subtle distinction at this point between the audio properties of the DJ and the transient audio properties of the sound being sent to 'air', defined as 'output' from the DJ. Discussions on the implementation of virtual rooms for DJs and producers, not modelled in earlier phases, clarifies many of the issues to do with DJ and producer interaction. DJs appear to co-exist simultaneously in both their booth and within the producer's room – a subversion of real world behaviour used to accommodate the microphone metaphor (see section 5.2.8).

5.2.2.3 Discussion

Little comparison between groups can be made with respect to the producer, since the role is only referred to indirectly during group1 discussions. Despite this, similar general patterns can be found in both groups' treatment of the producer by examining group 1's mixer model (see section 5.2.7). Specifically, a hierarchical organisation emerges in both cases in which the producer has ultimate control over audio output to air, manipulated via a mixing object. Group 4 extends beyond this basic model most notably during the discussion of tasks. Here, a number of physical relationships are drawn up between the DJ and producer to enable communication (such as line-of-sight visibility and hand gestures). These features did not find any equivalence in the subsequent metaphor model or implemented prototype, although the use of spatial features within the producer's environment is explored further in the show list discussion (see section 5.2.10).

5.2.3 MEDIA OBJECTS

Group 1	Group 4
Task phase	Task phase
(1:16) (1:1320) (1:1302) (1:1316)	(5:1409, 5:1412) (5:1411) (6:115) (6:561) (5:1423) (5:1423) (5:1410) (6:558) (6:560) (6:104) (6:92).
Meta-object phase	Meta-object phase
(2:1732) (2:1730) (2:1734) (2:1735) (2:1742) (2:1738) (2:1776)	(8:1288) (8:1283)
Interactor phase	Interactor phase
(4:951) (4:948) (4:946) (4:950) (4:952) (4:953) (4:950) (3:1173)	(8:1305) (9:433) (8:1306)

Table 5 Media objects evidence

5.2.3.1 Group 1 data

References to 'media' or media objects such as CDs, mini discs and tapes change substantially, taking on different roles during each of the three phases. During the task phase, there is some confusion over the differentiation of media objects that contain media and the use of the play list as a method of recording a selection of songs. Direction of actions such as "play" and "stop" towards a media player supports the playing of tracks – music located at some physical position on the media object. As the group progresses on to the meta-object stage they verify the relationship between media objects and tracks (a 'hi-fi' object is introduced). Although an agreed model is reached, the group then move away from this concept arguing that even though they recognise the concepts as a metaphor model, their prototype is not intended to support the actual physical objects themselves. Instead, one of the media objects, the CD, is put to use as a metaphorical entity in a slightly different way – used to *represent* the play list. This is achieved by the development of the 'CD rack' which holds many CDs from which a DJ might choose a track, however group 1 recognised that whilst this model holds true in the real world, it does 'not align' in the metaphor model. The reasons for this perceived misalignment become clear during phase 3 in which the actual implementation and mappings to the previously constructed task model take place. In fact the play list holds MP3 objects (the media) and is used for selecting songs to play whilst the mixer desk acts as the functional point for such actions as playing and stopping media.

5.2.3.2 Group 4 data

From the outset, group 4 recognise the relationship between media objects and tracks, however the tension between this model and the appearance of tracks on the play list becomes apparent earlier in the task phase rather than later in the meta-object discussions. Virtually no mention of media playing devices is made at all, despite the discussion being focused on real-world tasks – the reason for this becomes clear during phase 3 work. The concept of a media object as a 'collection' of tracks is developed and the creation of a play list is strongly tied into the selection of appropriate media objects. Meta-object discussions reveal a sudden 'folding in' of the media object into the 'track' for the same reasons as group

1 – the group only intends to support 'soft media' (digitally encoded audio found on the prototype system's hard disk). During phase 3, the playing of media is executed by the operation of a small collection of buttons that use graphical icons found on real world media players.

5.2.3.3 Discussion

In discussing media objects (such as CDs and mini-discs), both groups identified real-world relationships between media players, storage units and the notion of tracks. The difference in clarity and application of these relationships to their prototype design is noticeable between groups. During the exploration of the task model, both groups suggest manipulation of media objects in conjunction with a player and guidance from the play list. However, group 4 makes a relatively rapid departure from this model as they go on to describe the metaphor model in which the isolation of the track concept and its appearance in the play list emerges. In contrast, group 1 attempt to maintain some relationship with their task model by using a media object to embody a similar kind of 'play list and track' isolation. As a consequence, group 1 end up using media object ideas to graphically suggest the nature of the play list. Ultimately, media objects disappear entirely in the final prototype implementation.

5.2.4 THE PLAY LIST

Group 1	Group 4
Task phase	Task phase
(1:1296) (1:1304) (1:1308) (1:1303) (1:1298) (1:1305) (1:1310 – 1:1311) (2;1745) (2:1744) (2:1763) (2:1746)	(5:1422) (5:1438) (6:115) (6:561) (5:1408) (5:302) (5:1423) (5:1413) (6:558) (6:560) (6:562) (6:92) (6:566) (5:1447) (5:312) (5:1420) (5:1430) (5:451) (5:1429) (5:1427) (5:1428) (5:1414).
Meta-object phase	Meta-object phase
(2:1738) (2:1731) (2:1741) (2:1742) (3:1176) (3:1154) (3:1159) (2:1763) (3:1154) (2:1743) (2:1748) (2:1747) (2:1744) (3:1158) (2;1749) (2:1776) (2:1738) (2:1731) (2:1742) (2:1761) (2:1740) (2:1741) (2:1749).	(7:844) (7:847) (7:204) (7:848) (7:845) (8:1285) (8:1290) (8:1287) (7:847) (8:1291) (8:1293)
Interactor phase	Interactor phase
(4:967) (3:1168) (3:1170) (3:1169) (3:1172) (3:1173) (4:962) (3:1172) (4:494) (3:1171) (4:967) (4:949) (4:950) (4:954)	(8:1307) (8:1308) (8:1325) (9:435) (9:438) (8:1309).

Table 6 Play list evidence

5.2.4.1 Group 1 data

The play list is identified in the task phase as a mechanism by which desired songs might be recorded in the order in which the DJ wishes to play them – these operations are discussed as potential write, erase and shuffle operations. However, during the specification of the *task* view of the play list, implementation details already begin to emerge that suggest that the role of the play list as just a scheduling device is not shared by all the team members. This view is reinforced as discussions relating to whether or not DJ commentary might appear as a part of the show plan continue. Disallowing a 'time slot' for the DJ to talk on air suggests that probably prototype functionality (which it turns out does not

support the microphone) is of primary importance for this object. However, this impasse does yield recognition of an important aspect for the creation of the play list – the need to establish a source of songs from which to choose. The group recognises this problem but finds it hard to suggest where or what this source might be, so the interviewer suggests the word 'inventory'. Development of the play list and the inventory part continues in the meta-object phase – both are encapsulated within the 'CD rack'. By this stage, the idea that tracks are properties of a media object has diminished and instead the inventory becomes a list of potential tracks that can be 'copied' to the play list. The notion of 'copying' is also discussed as a metaphorical operation for the monitor (an object that displays what is playing at any one time), but dismissed as being inappropriate. Actual implementation of the play list discussed in phase 3 reveals some remarkable distortions of the metaphor model. The play list itself is represented as an open CD case (mistaken at first by the interviewer as a book). Additionally, the play list is used not only for playing songs but also for broadcasting 'jingles' – a secondary source of audio, but used for advertisement. Whilst the two lists are maintained separately, they are both implemented using the same interactor design. To view and select the jingles version of the play list, the user now clicks on the hi-fi object – which does not, in itself, play media at all.

5.2.4.2 Group 4 data

Similar properties identified by group 1 for the play list are identified by group 4 during the task phase elicitation. Analogical descriptions of the play list as a box containing media objects are soon replaced when it is decided that the list should serve to maintain a collection of tracks, not media objects. Adding a track to the play list is suggested as writing track information – this property is 'transferred' to the play list, however there is considerable opposition to the idea that this operation has anything to do with media objects. The role of tracks as information carriers, rather than properties of a media object, is emphasised further during meta-object discussions. Tracks can be added and removed from the list as well as edited. At this stage, group 4 recognise that there is a problem with the metaphorical operation of the play list since they have not adequately considered a source from which to choose tracks and instead are forced to rely on implementation details. The implementation of this list departs entirely from any real attempt to sustain a metaphor other than that provided by the operating system.

5.2.4.3 Discussion

For both groups, the role of the play list becomes central in not only scheduling tasks but also the playing of tracks. Similarly, both groups experience tension in treating the play list as an information carrying device (predominant in the task model) and its adaptation as an interactive 'track' container. During task elicitation, the use of the play list as a means of recording information that will help the DJ to remember which track to play during the course of the show reveals the notation of a collection of media objects from which to choose - at least for group 1. This separation of media collection and play list is not

suggested by group 4 during either the task or metaphor phases and leads to problems during interactor discussions. Each group only implements soft media audio reproduction (these are stored as 'files') and this may partially account for the emergence of the play list that acts much less as a scheduling service and much more as a container for singular media objects specified as single tracks.

5.2.5 PLAYER DEVICES

Group 1	Group 4
Task phase	Task phase
(1:1301) (1:1316) (1:1302) (1:1320) (1:1317) (1:924) (1:1320) (1:1300)	(5:1411) (6:560) (6:92) (5:1432)
Meta-object phase	Meta-object phase
(2:1733) (2:1752) (2:1754) (2:1732) (2:1736) (2:1753) (2:1755) (3:1162) (2:1754) (3:1157) (2:1735) (3:1163)	(8:1282) (8:1287) (8:1284) (8:1287)
Interactor phase	Interactor phase
(4:950).	(8:1305) (9:433) (8:1306)

Table 7 Player devices evidence

5.2.5.1 Group 1 data

The generic term 'device' is most frequently used to refer to a media playing machine and, during the task phase, has abstract and literal associated actions that include activation, play and stop. Resolution of the conceptual role of the track and its relationship to media objects served to clarify the operation of the player device. This disambiguation is further refined in the meta-object work with group 1 who suggest that a 'hi-fi' object should be employed to embody media playing devices for specific media objects such as CDs or tapes. However, despite recognising the hi-fi as a model for media playing, it is dismissed since it will not be functionally supported in the implementation. At the same time the role of the mixing desk becomes more important as implied references to sliders controlling the audio output to 'air' are introduced. By the third phase of elicitation it becomes apparent that the mixer desk has not been developed to mix audio output from a playing device but is in fact the playing device itself, operated by sliders which really act as mutually exclusive switches.

5.2.5.2 Group 4 data

Throughout the three elicitation phases group 4 elaborates very little with respect to devices that play media objects. Their implementation of a 'virtual' media player suggests that they were not inclined to recognise these devices either as valid parts of the prototype metaphor, or as machines that are actually used in the real world.

5.2.5.3 Discussion

Media playing machines feature frequently in elicitation phases 1 and 2 for group 1 but hardly at all throughout group 4 discussions. In either case, the strongest association of devices with media objects

occurs in the task description and then becomes increasingly less important in meta-object and interactor phases. The continual reduction of involvement of player devices seems to occur with the evolution of the play list as MP3 container. Group 1 suggest the use of a hi-fi object as a media playing device for various audio formats and developed to the extent that panels and buttons are defined (however, this model is not followed up in their prototype implementation). Rather than developing a partial hi-fi model for their implementation, group 1 instead pushes media playing operations onto the mixing desk. Ironically, group 4 who in previous task and metaphor models make little use of a media player references, choose to use iconic buttons commonly found on a hi-fi in the final prototype.

5.2.6 THE TRACK

Group 1	Group 4
Task phase	Task phase
ON MP3s (1:1299) (1:1303)	(6:115) (6:563) (6:121) (6:560) (6:558) (6:562).
Meta-object phase	Meta-object phase
(3:1176) (3:1154)	(8:1289) (8:446) (7:204) (7:848) (8:1292) (8:1283) (7:846) (7:845) (8:1285) (7:842) (7:840) (8:1290) (8:1286) (7:841) (7:847) (8:1287)
Interactor phase	Interactor phase
(4:953) (3:1171) (4:334) (4:950) (4:953) (4:954) (4:334) (4:950).	(9:436) (8:1306) (8:1308) (8:1326) (8:1325)

Table 8 Track evidence

5.2.6.1 Group 1 data

The nature of the track is discussed with respect to media player devices, media objects such as CDs and the play list in other sections (see 5.2.3 and 5.2.4). Group 1 makes generic statements regarding tracks as both objects that feature on a play list as well as technical media files (MP3s); this causes difficulties in identifying the exact nature of the concept of track in the task model. Phase two discussions expand the use of track still further by examining how track 'information' can be derived from different media objects and be displayed in other metaphorical objects such as the play list and monitor. The track concept is further specialised by group 1 suggesting that tracks found on mini-disks are jingles. Interestingly, group 1 uses a metaphorical description in phase three discussions to outline 'movement' of the track around other objects in the interface even though this is not actually manifest in the interface prototype itself.

5.2.6.2 Group 4 data

The role of the track does not have as broad a scope for group 4 as it appears to for group 1. During the task phase group 4 agreed on a 'real world' relationship between track and media objects, but chose to isolate the track as an abstract entity. Further abstractions of the track emerge in phase 2 where track information is regarded as important; whilst reflecting on the possibilities of a more concrete treatment of tracks group 4 relate to technical problems that have not allowed them to pursue this design. The

abstraction of the track results in a problem with respect to their source during play list creation and group 4 eventually resort to the desktop metaphor to explain their design. In discussing this issue, the group remark that the design of the play list arises from the problem of track 'tangibility'. Whilst describing the nature of the track in phase 2, the group also use strongly implementation orientated views including the idea that tracks "play themselves". Implementation of the play list and the 'virtual media player' during the final elicitation phase suggests that the group has been heavily influenced by coding issues with respect to this object.

5.2.6.3 Discussion

The track entity is one of the principal concepts in the USC prototype for both groups and yet, with the exception of the task model, it remains relatively abstract and unintegrated with other related domain concepts. Early discussions of the track disclose a container like relationship between it and a media object and made accessible through the use of a media player. As the role of the play list emerges, this model is gradually dropped as the need to maintain a list of single tracks from multiple media objects becomes apparent (group 4 makes this point explicitly). More specifically, it is the track 'information' that is revealed as important: at the interactor phase for group 1 and at the metaphor stage for group 4. Early examination of the relationship that tracks have with media outlined the need for group 1 to include an inventory from which to choose songs. Group 4's relatively narrow abstraction of the track is revealed by their view of "tracks playing themselves" and their subsequent reliance on the desktop metaphor to support play list creation.

5.2.7 THE MIXER OBJECT

Group 1	Group 4
Task phase	Task phase
(1:246) (1:1323) (1:1333) (1:1322) (1:1318) (1:1315) (1:376) (1:1322)	(5:1431) (5:1445) (5:1446)
Meta-object phase	Meta-object phase
(2:1768) (2:1759) (2:1730) (2:1752) (2:1755) (2:1756) (2:1762) (2:1771) (2:1767) (3:1155) (2:1755) (2:1756) (2:1771) (2:1730) (3:1160) (2:1762) (3:1161) (2:1770) (2:1739) (2:424)	(8:26) (8:1327) (7:843) (7:137) (8:1302) (7:857) (7:843) (7:857) (8:1302) (8:1301) (7:137) (7:858) (8:1327)
Interactor phase	Interactor phase
(4:952) (4:960) (4:962) (4:947) (4:960) (4:957) (4:956) (4:948) (4:960) (4:946) (4:962) (4:947) (4:948) (4:963) (4:964) (4:965) (4:966)	(8:1315) (8:1314) (9:441) (9:443) (9:444) (8:1323) (8:1324)(8:1320)

Table 9 Mixer object evidence

5.2.7.1 Group 1 data

For group 1, the mixer desk definition begins broadly within the task phase and becomes progressively narrower as subsequent elicitation phases are carried out. The task view of the mixer is that of a machine maintaining sliders that control audio output to air from a variety of audio sources (including the microphone and the 'output' from DJs situated in rooms). However, even at this early stage one member

of the group challenges this role of the mixer desk, suggesting that the object should provide an interface to media playing actions (such as 'play' and 'stop'). During phase 2, the beginning of an apparent transformation of the mixer desk's function as an entity for mixing audio to that of one for playing media takes place. A discussion of the slider reveals confusion as to whether or not the sliders are to be used as modifiers to local media output (limited to just the DJ scope) or as a means of determining output strength to 'air'. Possibly as a means of resolving this problem, a button and light on the desk associated with a particular slider are suggested as a means of determining whether the audio from a particular device is sent to the producer or not. The introduction of an 'exclusive switch box' model for sliders on the mixer desk, in which only one slider at a time may be up, begins to shed some light on the final implemented prototype. In phase 3, the original idea of the mixer desk as a device for modifying the audio that is sent out to air is changed to that of a media-playing device. Mixing is exchanged for 'switching' for all but one slider; media tracks are played and stopped by clicking sliders (note no dragging is used for this action). Overall volume for the output of a mixer desk is determined by the volume slider – the value of which is scalar and can be modified by a mouse dragging action. The function of the mixer desk for the producer is identical in all respects other than that media sources are now replaced by DJ sources that are turned on and off using the same exclusive switching model.

5.2.7.2 Group 4 data

In discussing mixing tasks in phase 1, group 4 makes few references to real mixing machines from the real world; early discussions resulted in a severe reduction in description since they perceived the entities to be highly complex. During phase 2, the development of two types of 'mixing board' as fixed or moveable collections of sliders emerges. All slider objects maintain the same basic, modifiable scalar value manipulated by a 'slide' action. However, movable sliders (described by the familiar 'drag-and-drop' desktop action) are 'attached' to DJ objects within the producer's environment and as such dynamically change the focus of their scaling behaviour. Further discussions in the final elicitation phase support this model through a brief explanation of the software components used to implement the design.

5.2.7.3 Discussion

In different ways, both groups can be seen to accept a conventional view of the mixer desk (for group 4 this is a superficial view) before subverting its behaviour to support alternative tasks. For group 1, the alteration of the mixer desk and associated sliders occurs at the very outset, moving through a resolution stage in which the audio mixing function is combined with output control, resulting in a 'switching desk'. The function of mixing remains constant throughout the elicitation phases with group 4, however the focus is on the slider object rather than the desk itself which is understated throughout. Unlike group 1, sliders are scalar and only used to change audio properties (including volume, treble and bass). Group 4 choose to turn the conventional mixing desk model upside down with respect to modifying DJ audio for

broadcast - a single slider is dynamically associated with any number of DJs. Arguably, both groups' solutions are logically similar in that they only allow the producer to modify one DJ's output to the exclusion of all others, however the mechanism through which this is achieved is quite different.

5.2.8 THE MICROPHONE AND AIR

Group 1	Group 4
Task phase	Task phase
(1:1333) (1:1315) (1:1325, 1:1326) (1:1323) (1:1334).	(5:325) (5:358) (5:113) (5:497) (5:1442) (5:1443) (5:1437)
Meta-object phase	Meta-object phase
(2:1758) (3:1155) (3:276) (2:1755) (3:1175) (3:276) (2:1771) (2:1756) (3:1161)	(8:1298) (7:849) (7:850) (8:23) (8:25) (7:851) (7:136) (7:859) (7:851) (8:25)
Interactor phase	Interactor phase
	(8:1311) (9:442) (9:115) (8:1312) (8:1311)

Table 10 Microphone and air evidence

5.2.8.1 Group 1 data

Models regarding the nature and use of the microphone remain relatively consistent throughout all elicitation phases for group 1. Audio output from the microphone is modified by the mixer desk and sent to air according to slider status in both task and metaphorical views. This consistency may be accounted for by the fact that no microphone implementation took place.

5.2.8.2 Group 4 data

The use of the microphone and air space is interesting. During the task elicitation, the model for putting a DJ on and off air is described with reference to using a mixing desk to fade DJ output (from the microphone or other media sources) in and out. A novel modification of the use and behaviour of the hardware discussed in phase 1 is used as a basis for a metaphor in phase 2. The producer still controls which DJ is 'on air' but this is now performed via the manipulation of a microphone that is 'attached' to a DJ object within the producer's booth. Once attached, the microphone effectively acts as a conduit passing DJ audio out to air – placing the microphone back on the stand empties the air. Implementation of this model is achieved through a direct manipulation model and includes 'snapping' behaviour borrowed from the desktop metaphor.

5.2.8.3 Discussion

It is not surprising that group 1 view the microphone entity in a consistent manner throughout since it was not developed by the group; if it had changed over time this would have added a concern over the reliability of the elicitation process. Group 4 use a similar mechanism for the operation of the microphone as for the use of the mixer object by the producer (see section 5.2.7). The combination of the microphone and movable volume slider acts as an almost logically identical switch-based system as

proposed by group 1. However, unlike group 1, the objects used to perform this operation do not exhibit as many unexpected behaviours.

5.2.9 THE ADVERT

Group 1
Task phase
(1:310) (1:1300) (1:341)
Meta-object phase
(2:367) (2:1774) (3:1167) (3:1164) (3:1167) (2:1757) (2:1772) (3:1165) (3:1165) (3:1166) (2:1775) (2:1773) (2:1765) (3:1159) (2:563) (2:1737) (3:1167) (2:1751) (3:1159) (2:1751) (2:1750)
Interactor phase
(4:978) (4:975) (4:972) (4:970) (4:969) (4:976) (4:974) (4:973) (4:979) (4:971)

Table 11 Advert evidence

One of the requirements of the USC prototype was to provide support for the management and transmission of advertisements. Both teams were expected to address this demand but only group 1 found resources to devote to its design and implementation. Adverts were considered as both audible and visual[23] during the task elicitation discussion, however the general consensus was that adverts, like tracks, were contained on media objects and broadcast using a media player. The second phase of the elicitation reveals an alternative to an analogous set of objects that could have been taken from the task model, however. Here, a book containing pages is used as an inventory from which to choose adverts. An unusual addition to the book object comes in the form of a 'time line' that allows the DJ to place adverts from the pages in sequence. Group 1 makes a number of references to the similarity of the play list and the advertisement features in this elicitation phase. During the course of the discussion, the nature of moving adverts from the inventory to the time line is discussed in more detail, particularly with regard to natural objects and interactions found in the real world. Adverts are copied from the inventory on to the time-line in the order in which they are expected to go out to air. Surprisingly, the group struggled to suggest metaphorical alternatives for the relatively abstract time-line; similarly, the idea that the mouse might mimic the actions of the hand with the book metaphor was understood but not considered important. Implementation discussions served to clarify the underlying model of the book and the time line and include token button components that would allow the DJ to turn the pages of the book. At this time, it occurs to the group that the time-line has been implemented twice within the user interface and that it was a strong candidate for re-design.

5.2.9.1 Discussion

The task model suggested by group 1 drew parallels between the use of media objects to play songs as well as 'jingles' (tracks containing audio advertisements). Perhaps, not surprisingly, many of the

[23] One team member argued that digital radio may post text advertisements to radio displays

underlying features of the resultant advertisement objects (the book, its pages and the adverts contained therein) are suggested to operate in a very similar way to the play list (see section 5.2.4). However, their metaphor model builds on the essential proposition of an inventory and a list by the use of pages (from which adverts may be chosen) and a time-line upon which adverts are ordered. The implementation of the advertisement model is graphically and behaviourally different however, whilst the underlying mechanism remains constant.

5.2.10 THE SHOW

Group 4
Task phase
(5:1424) (5:376) (5:1428) (5:302) (5:1441) (5:1421) (6:565) (5:1425)
Meta-object phase
(8:1294) (8:1296) (8:1295) (8:1319)

Table 12 Show evidence

The role of the show list evolves from a generic collection of media to a scheduling model for placing DJs on air within a time frame, and in order. In discussing the physical mechanism for show list management, group 4 preferred to cite a software tool (a spreadsheet) rather than a paper based system for the task model. Later discussions of show list support in the metaphor model have not been developed since group 4 argued that the development of scheduling features would be untenable within the project's constraints. However, in pursuing the likely features of the show list for the metaphor model two explanations are given. The first relates to implementation details – DJ profile information is presented textually in a list, each item of which can be moved up and down. Later discussion regarding the nature of the DJ object (as seen in the producer's environment) show that group 4 had considered the arrangement of DJ objects in a visual queue as another means of managing the show.

5.2.10.1 Discussion

The rapid reduction in design complexity can be seen in group 4's treatment of the show list. Implementation of the list is almost identical to the play list; the visual queue suggested by the group is partially supported by the system in that DJs are horizontally aligned in the order of their connection (they are not movable, however).

5.2.11 THE ROOM

Group 1	Group 4
Task phase	Task phase
(1:1286)	(5:1440).
Meta-object phase	Meta-object phase
	(8:1303)

Table 13 Room evidence

Both groups recognise the physical separation of DJs in their own rooms or booths, however the impact of this concept is not pursued in any detail.

5.2.11.1 Discussion

During the elicitation phases, the concept of virtual environments (booths or rooms) within which DJs and producers exist is, at best, tacitly discussed between group members. In each final implementation, an indication of a shared environment is graphically depicted weakly by group 1 and strongly by group 4. Only small changes in the appearance of the user interface indicate DJ role and persistence in a shared environment for group 1; a notice board displays logged on users whilst minor modifications to the mixer desk indicates producer modality. In contrast, group 4 makes graphically distinct presentations for both the DJ and the producer as well as engaging producer interaction with DJ avatar graphics.

5.3 Model summary

In reviewing the treatment of the eight, core USC objects expressed within the ISML framework, a number of common design behaviours emerge:

- Design reduction
- Non-concrete concepts
- Implementation bias
- Metaphor mangling
- Common models and re-use

5.3.1 DESIGN REDUCTION

In many cases objects initially specified in the task model are subjected to a progressive reduction of complexity as they are re-represented in the metaphor model and subsequently re-represented in their implementation as interactive GUI components. Some of the objects in the task domain (such as the media objects and player devices) disappear almost completely. Reduction occurs most extensively where the design must address the use of media and tracks, in which a structured hierarchy of objects is made redundant by the play list.

5.3.2 NON-CONCRETE CONCEPTS

The manipulation of the play list and the output of specific DJs to air features strongly as the functional 'core' of the prototype around which the various metaphorical objects orbit. Both groups struggle to resolve a sufficiently robust metaphor and interactor model that would capture the physical features of media objects and players from the real world and the more abstract concepts of the track and play list. Consequently, a tension appears to emerge between the abstract (invisible) and the concrete (visible), leading both groups to fall back on software engineering or WIMP concepts where no obvious metaphor presented itself.

5.3.3 IMPLEMENTATION BIAS

The influence of implementation concerns is likely to have reinforced the need to resort to more conventional solutions. Examples of strong biases on the metaphor design can be found in the functionality constraints imposed by both groups with respect to support for media sources other than MP3 files and the availability of mixing audio streams. These thoughts are particularly apparent in discussions with group 1 during the task phase (which ideally should be void of all implementation details) and through later discussions regarding early design ideas with group 4.

5.3.4 METAPHOR MANGLING

Model feature deprecation and functional trade-offs may also be partly responsible for 'metaphor mangling' evident in the course of both groups' prototype. Group 1's transformation of the mixer desk and group 4's play list that contains 'self-playing' tracks are two examples of severe metaphor distortions (an effective reduction in model complexity). Other, less destructive alterations to the metaphor model are also affected by each group. The single microphone model adopted by group 4 is intuitive and whilst group 1 borrows a metaphor for their advertisement book, rather awkwardly, from an unrelated but natural real world object.

5.3.5 COMMON MODELS AND RE-USE

There are also occasions where the general, underlying structure and behaviour of an object within the metaphor model are re-used (without necessarily resulting in the same implementation). 'Inventory and list' combinations appear in both designs to support the play list, show list and advertisement solutions (this is only explicitly specified in the metaphor model by group 1, however). Group 1 also re-uses the mixer desk model for both DJ and producer. An 'attach and map' model is re-used by group 4 to associate mappings between a DJ and his/her presence on air and also a modification in audio output.

5.4 Summary of design behaviours

The effects on the over-all design of the USC prototype from these behaviours are summarised in Table 14.

	Design behaviours				
Objects	Design reduction	Non-concrete concepts	Implementation bias	Metaphor mangling	Common models & re-sue
DJ	G1	G1			
Producer	Both	G1			
Media Objects	Both	Both	Both	G1	
Track	Both	Both	Both	G4	
Play list	Both	Both	Both	Both	
Player devices	Both		Both	G1	
Mixer	Both		Both	Both	Both
Mic and Air			Both		
Advertisement	Both				Both
Show	Both	Both			
Room	Both				G1

Table 14 USC design behaviour summary

From this summary, the distribution of these effects can be seen for both groups. Both groups could not, for the most part, avoid quite severe reduction in design. There is some evidence to suggest that the reasons for this may lie in both the complexities of the metaphor chosen and also the implementation bias exhibited by both teams: implementation bias occurs in discussions concerning almost all the core, metaphorical objects and relates frequently to the treatment of tracks. Both groups experience difficulty in managing the relationship between media objects, the play list and media player devices, so it is not surprising to see that this is also where problems with non-concrete concepts and metaphor mangling also occur. Perhaps as a reaction to the inherent complexity of the design, both groups show some degree of re-use (or at least repetition) in the design of the 'periphery' objects – i.e., the advertisement book and show list.

One possible conclusion from these observations is that the design reduction and resultant metaphor 'mangling' behaviours exhibited by both groups are at least partially the result of:

- the complexity of the metaphor model
- the lack of support for non-concrete concepts in a metaphor abstraction
- implementation bias

This theory is discussed further in chapter 7; evidence for and against this theory (developed post-project) can be found in each group's reflections on their design experiences using the ISML framework, which follows.

5.5 Group reflections

After the USC software engineers had been introduced to ISML and the subsequent elicitation meetings were drawing to a close, each group was asked to reflect on the process of specifying the prototype using ISML framework. Each group was asked a series of five open-ended questions to gather their views on how ISML specification related to their design activities with respect to generation, modification and practicality.

5.5.1 Q1. VERIFICATION OR GENERATION

Was the process of specifying their design using the ISML framework regarded as only a verification of their extant design ideas or did the group feel that new aspects of the prototype design were revealed?

> Group 1:
> "[Interviewer] Has this been an exercise which has .. would you say it has verified what you have talked about? And/or brought up design issues you hadn't thought about?"
>
> "Definitely the latter, I think, yes. Because we, I think it makes you look at it in a different way, I think. Where as we ... I know we're probably not supposed to, but as we develop it, you're thinking ... as you design it, you're thinking of implementation things at the same time, and that obviously incorporates it where as ... doing it this way, you might do things a bit differently."
>
> Group 4:
> "[Interviewer] So, this exercise, again, is it a verification exercise or have I opened up design issues, that you guys.."
>
> "I think you've opened up design issues."
>
> "I would say that I thought we were pretty confident on where we were. More as a verification."

"Yes, I think so. You know, there might be little things that maybe we had to question and then think, you know, that stuff happens I think."

"Yes, but I think from an HCI point of view, that methodology has totally blown our ... has totally revealed a massive flaw in our system, hasn't it? With the media selection ... The metaphor breaks down, doesn't it? And the idea is to keep the metaphor going, isn't it, in this virtual ... It's ... you know, you've worded and looked at the design from a way I've never thought of it before..."

There seems to be some agreement by both groups that the elicitation process uncovered some design issues. An important influence that had already begun to emerge during the course of the interviews was confirmed by the groups, namely: the implementation target places biases and constraints on metaphor development. For group 1, implementation (or lack of it; audio mixing and transmission was problematic for the group) results in a distortion of objects and their eventual role in the prototype. The same issue forced group 4 to fall back on windows-based descriptions and implementations of certain objects (such as the play list). Group 4 elaborates this on further:

"[Interviewer]: ... It's great to see, for me, where this fails. I think, I'm not saying ... this is not a reflection on your design, I think it's a reflection on the fact that you can't necessarily fit everything into a concrete metaphor.

[Group 4]: I think that's exactly what I was trying to say. Yes, it's broken down there, because we've got this abstract thing which we can't define metaphorically."

Highly abstract objects (such as the 'track') proved to be difficult for both groups to specify easily within the ISML metaphor framework. It is perhaps not surprising then that these entities are framed in computer software terms such as 'file' and exhibit unusual behaviours for the metaphor, such as moving between objects (group 1) and 'playing themselves' (group 4).

5.5.2 Q2. ANALYTICAL OR CREATIVE

To what extent was the ISML specification process regarded as either an analytical or creative exercise?

Group 1:
"I think it's more creative .. maybe ... both... I don't know about creative. I wouldn't personally say it's creative, because we already had the ideas? That's why I changed my mind ... I think it's creative from a point of view that, if you've got some already, you can modify it a lot going through this process and create further ideas..."

Group 4:
"[Interviewer] Right, creative or analytical. Was this exercise more analytical than creative?"

"[Group 4] Oh yes.. More analytical. Yes."

This line of questioning revealed relatively little other than to support the perception that the ISML elicitation process was a catalyst for unconsidered aspects of design.

5.5.3 Q3. DESIGN MODIFICATION

Did the elicitation identify areas in the design that required changes? If the group started a second-generation prototype, would they make different design decisions as a result of the elicitation process?

Group 1:
"[Interviewer] OK ... would you make different design decisions as a result of this?"

"If we were going ... if we were using this prototype as a prototype for a newer version, then quite possibly you'd take some of the ideas and put it there. But I wouldn't say we got a lot of ideas out of this. It was just a couple of things."

"Perhaps ... I think if we were implementing it now on what we've got and modifying that then we would probably get rid of a few things there, I think. Like maybe the timeline here and a timeline here. Which, didn't occur to me, I don't know about you? ..."

"Yes, I saw it as a symbol, rather than ... Yes, in effect, we added development effort where we didn't need to. Because you developed a timeline in the advert where you drag and drop ... but then we developed it there as well, so we don't really need to do that."

Group 4:
"[Interviewer] If you went through this, ... , right from the start, ... would you have found it as easy to come up with the designs that you have? Or would you find that it might constrain the way that you thought about the problem such that you wouldn't have the opportunity to be creative?

"I think had we had done this process before we'd come up with our design, it would have been a creative tool, it would have helped in our design, and it wouldn't then be a verification thing. Are you trying to ask could we use this as a design tool rather than as a verification tool? Specification tool, sorry."

"Well, yes, if we had carried it out before we'd done any design, it would have been more creative than analytical, certainly."

"Yes, I think this would have helped us in doing our design, yes, I think so."

"I still think we might have taken the same route, the kernel of our main kind of idea would still ... the whole metaphor we came up with, would still be very similar, but I do think that, if this process was taken before we'd done any design, it would have been more creative in terms of unlocking ideas rather than analysing ideas that we've had."

"Yes, the point I was going to make was, yes, this method seems to be how, or certainly how I would think anyway, but it's just that you don't know that's how you think."

"Like, you do think, right, well what needs to be done, what are we trying to change, what is involved in that, how do we do it, you just don't .. you do it a lot quicker in your brain, you think ah, right, you need to change this property, ah it's a button or... But actually getting it down and writing it, I think that's where this is going quite well, I think."

In fact, each group was asked a slightly different question so some care must be taken in examining the two responses. Group 4 considers the use of ISML at the beginning of a second-generation prototype and speculates on its application in a creative sense (this question was re-phrased to combine the previous in the hope that more detail might be forthcoming). Later in paragraph 5, the group reveals a similar attitude to that expressed by group 1 on the issue of the re-development of USC – the ISML elicitation process

revealed small aspects of the design that could be changed. Group 1's more direct answer, identifying aspects they would remove rather than fundamental changes to the overall design, is surprising since the group's final design has a number of unusual metaphor subversions not just at the interactor level, but also in the metaphor abstraction.

5.5.4 Q4. PRACTICALITY

The group was asked to reflect on the models they had generated and asked, given their project resource constraints, whether specification using the ISML framework was a practical exercise.

Group 1:
"[Interviewer] OK. Is this a practical thing to do? Within the constraints .. if we had the timing a bit better? Given the output of what we've learnt about the design and what you've told me about the design. Would you consider doing this exercise as a useful part of a practical project?"

"I've got two answers.."

"At the moment, no. Because I've only had one go at it, so I've got no idea of what effect it might have if I did it for real sort of thing. But I could see maybe that it could be useful, but I think that would have to be a couple of goes at it, on actual things, to see what happens."

"I'd say, if you had a group of people and they were all developers and they were all constantly building the software, then this is good, because it steps you back. But if you had someone whose job was to check the HCI of stuff, check that the design ideas are right, then you wouldn't really need it? Because they aren't getting involved in .. what I have a problem is.. I always get involved doing it, so if there's something that would look better, I'm just constantly worried how it works rather than the design aspect. So if you had another person doing it, then perhaps it wouldn't be needed so much. But if it's like a group of all developers, then I suppose it would get someone to stand back and have a better look at it. But I don't know if the documentation may be a bit too much? That's my view. On what you've shown us on the documentation, it's .. there's a lot to do, and in industry you probably wouldn't have the time to do all that."

"I think that's probably what sort of industry you're in. OK. Some design stuff is quite complicated anyway, so? So initially, from the first view point it looks complicated, but maybe once you've used it for a while, then it's not too bad."

"Or perhaps just forcing these questions to be spoken about rather than just doing the documentation as well? I don't know. Maybe it's just because I don't like documentation! Maybe that's what it's all down to."

Group 4:
"[Interviewer] This whole process, given what's still coming out of it, and the types of things we've discussed, like the tasks, the abstract design, some of your implementation. Is this a practical exercise to do, given the output and the time it's taken? If you had another go at this, with perhaps the same sort of time constraints, would it be a useful thing to do?"

"Yes, but earlier."

"Yes, I would certainly do it earlier, and I think, it's a really good way of everybody making sure of what the system does. I mean, literally, other people in the group, in the team because you're all sat here, it's being explained in fairly simple terms, so people can .. that's what's happened here. People have gone 'oh right, I didn't know that'. It's made things clear."

...
"It's possibly a requirements, kind of that phase."

"It took a while for the whole group to get a handle on the actual the problem, collectively, if you see what I mean."

...
"Each of us still had .."

"Yes, we had different views."

"We had different views on how we were going to look at it."

"Yes, perhaps if this was done earlier in the process, rather than being a verification, it would have been a practical approach to doing some design."

Again, some caution must be taken in examining the two responses as group 1 were asked to consider the scheduling aspects of the project whilst group 4 were not. However, despite the qualification of modifying the scheduling of the ISML specification, group 1 remains focused on the problems of workload and their lack of experience with the framework. Conversely, without suggesting a modification of specification schedule, group 4 suggests an earlier introduction. Whilst group 1 are fairly certain that ISML specification is a lot of documentary work rather than concentrating on what to them is important – the implementation - there is a concordance with group 4 on the issue of interface design. Both groups independently offer the view that the process offers an opportunity to examine their interface design ideas from a new perspective that was not just from their software-engineering point of view.

5.5.5 Q5. ELICITATION DIFFICULTY

Were the questions asked during the elicitation easy or difficult to answer?

Group 1:
"[Interviewer] OK. Was it easy or hard to answer my questions..."

"Abstraction was very difficult. Yes. It was hard to separate the abstract thing from the actual.. what we'd already implemented. But I think what complicated it more was the fact that we had what we're going to implement and we had like the ideal system, so you've got the design for the implementation and the design for the ideal system, and it was like pulling the two apart. Yes. The two different designs."

"[Interviewer] Right, can you explain what you meant by pulling them apart?"

"Well, you've got the design for the perfect system and then you've got the design for what we implemented. Now, it was very difficult to differentiate between the two. When you were asking us questions, because half the time we sort of like going, should we be talking about that one or this one?"

Perhaps, if this was to be done again, it would be better just having the one? So this is what I'm going to design and then ask questions about that?

"[Interviewer] Would that be the implementation? Or the abstraction?"

"You'd have your abstraction, but that would be what you're going to develop. Because we'd got loads of features on there, which haven't actually gone through and done? So it's ..."

"Yes. I think what would have been better with this, was if a) we'd have got our requirements off you sooner and maybe had look then, before Christmas, at this sort of thing, got ideas and come back to you and then gone through this before the end of term and then done this. This would have been a lot more helpful, because then it would have been easier to base a design and come up with something that was a bit more robust maybe."

Group 4:
"[Interviewer] Was this an easy or hard exercise?"

"Well, I found it easy."

"[Interviewer] I did notice that you guys were struggling a bit with, again not a judgement thing, I was struggling as well, with trying to fit some of these..."

"Yes, trying to .. yes, the abstract stuff was a little bit hard."

"Yes, coming up with single words to describe things that you know.. obviously can be quite complicated, that's quite difficult."

"But that's a language thing, descriptive thing."

"English language, again, very poor at describing things."

Without prompting, group 1 immediately identifies the problem of separating the metaphor model from the implementation details and further, they would seem to prefer it if the specification dealt only with how such a model might be implemented. Although disappointing, a response like this is not surprising when considering their reflections on verification and the frequent references to implementation issues during the elicitation phases, including the task model. Group 4 identifies the same issue, siting the narrowness of some of the definitions required from them during elicitation.

6. Discussion

A number of practical issues challenged the USC case study, including:

- Case study life cycle limitations
- Elicitation and analysis limitations
- ISML novelty and complexity

For both groups, the six-month time frame was a challenge since each individual had other work commitments. Within this scope, each team was faced with a significant requirements gathering phase that not only involved the author but also another university member of staff acting as a company manager. In addition to this, specifying their design in ISML was an additional task added to their project schedule. Consequent limitations on time forced the ISML documentation to change from the (originally planned) pre-development specification stage to a 'rolling' exercise lasting most of the second part of the project. For the same reason, each group was strongly resistant to giving up time to learn to write, at a

detailed level, an ISML specification. Pressure on time also required an elicitation strategy that resulted in a limited focus on those aspects of design that appeared important at the time.

It is important to consider the impact of the quantity and quality of the elicitation execution whilst examining the qualitative themes and patterns identified in the data. A number of problems and limitations are also introduced by the application and execution of a grounded theory based approach to the case study. An adaptive approach to data gathering had to be taken to a) accommodate and not significantly interfere with group working practice and b) pursue interesting veins of discussion as they occurred - this is common in qualitative method, see Pidgeon (1996). During the course of the case study, there were two occasions where data were lost due to technical failure[24] – once during the early requirements phase and once during the meta-object elicitation. The latter had a more serious impact on analysis since data were lost on a) the initial reflections on ISML from group 1 and b) a meta-object discussion with group 4. With respect to quality, the transcript records occasions when the interviewer, in an attempt to stimulate discussion, suggests ideas or phrases for the specification. In addition, during reflection on the ISML elicitation, each team received the open-ended questions phrased differently. Both of these issues raise questions of reliability and interpretation since a bias is introduced.

The selection of the designers of the USC prototype will also have had an effect on the data; as software engineers there were naturally going to be a strong influences on design from the computing domain. In designing the case study, some consideration was given to other potential design cohorts, including graphic designers who might not have been influenced in this way or indeed have had so many concerns regarding implementation. Choosing such a team also has disadvantages however, since some understanding of software engineering and HCI terms was considered essential during discussions. The chosen groups were familiar with the general principles of HCI (these included task analysis, UI prototyping techniques and metaphor), which even though the ISML as a framework was entirely new to them, made them the strongest candidates for the study.

Despite the restricted time available for the case study, a significant volume of transcriptions was generated. Since only the guided elicitation phases were chosen for analysis, there is a possibility that important design issues discussed earlier in the case study have not been uncovered by the analysis. Indeed, there is some evidence in the reflections by each group that some design decisions were being made during the (functional) requirements gathering – group 4 actually suggests that ISML specification might have been useful at this point. However, the focus of the analysis reflects the case study qualitative

---

[24] The mini-disc device failed to record audio.

data gathering strategy. An early decision had to be made with respect to the likely richness of a potential specification retrieved from either a very early stage in the case study or at a later time. It was decided that whilst an early elicitation may have resulted in the capture of interesting developmental ideas, these were likely to be highly volatile and, at least in theory, not easily mapped interactor solutions since these would not have been considered at that stage. Scheduling the ISML elicitation later, it was hoped, would have allowed the USC groups time to better understand the functional requirements of the prototype and thus have a more stable basis for discussing the user interface design.

7. Summary and conclusions

Through the specification of the USC design using the ISML framework, five common design behaviours have been highlighted. Of these, the effects of design reduction, non-concrete concepts and implementation biases can be seen to affect the treatment of media objects, tracks and their management in the DJ's broadcasting role. A constrained user interface design project is highly likely to suffer from a progressive reduction in design features; one of the subsequent effects of this is the mangling of metaphors. Non-concrete concepts caused both groups difficulties in finding suitable metaphor representations, resulting in either the removal of metaphor features or a design solution based on conventional WIMP-based components.

The post-development reflections by each group appear to support the theory that difficulty in expressing some concepts and implementation issues influenced the metaphor development (i.e., the reduction of its scope, and subsequent reliance on the desktop metaphor). Group reflections also indicated problems with abstracting the metaphor model using the ISML framework, making particular reference to a limited range of expression and confusion with implementation details. However, in addition to this, both groups independently observe some utility and benefits from the process in that a) it highlighted problems in their design and b) they could identify potential changes for a second version of the prototype. Finally, concerns regarding specification effort and scheduling of the specification suggest that improvements in ISML specification capture method are required.

The USC case study has generated a rich data set from which only a small sample has been drawn upon for analysis. This qualitative analysis has identified five design behaviours that occurred whilst the groups attempted to express their design ideas using the ISML framework. There are clear indications that the separation and abstraction of the metaphor model was difficult for both teams and that implementation issues have an influence in this problem. However, both teams were able to provide task, metaphor and interactor models as well as provide mappings between each during the elicitation. Their perceptions of

the specification exercise overall was that it had some genuine utility whilst at the same time requiring changes to the scheduling and documentation strategies.

BLANK IN ORIGINAL

CHAPTER 6 Evaluation of the USC specification

1. Introduction

In this chapter, the Urban Shout Cast ISML models are analysed to discover to what extent the ISML framework captures the USC user interface design. This question is answered in two parts. Part one analyses each of the five ISML abstractions to see what design data was captured and what was missing. Data collected for each of the group's design were collected during the USC design meetings and subsequently collated into three main parts: tasks, meta-objects and interactors for comparison. During analysis, transcription data were also used as a means of clarifying parts of the specification where necessary. Part two evaluates to what extent a unified metaphor model can support either USC teams' concrete prototype design. The scope of this evaluation is necessarily limited to the tasks commonly supported by both USC implementations, namely: the playing of media, play list management, mixing and a simple broadcast model. The chapter concludes with a summary of the findings, outlining the strengths and weakness of ISML.

2. Comparison of USC models

In the following sections, data are summarised for each of the ISML parts interactively specified with the interviewer during design meetings. It is important to note that the models examined here do not reflect the entirety of each group's final prototype; during the course of the project, each group diverged not only in the metaphors they developed but also in terms of the underlying functionality of the system as a whole. A discussion of each complete system is beyond the scope of this analysis. As before, analysis question two (see chapter 5, section 3.2) is further sub-divided into two parts:

Part 1: What aspects of design did ISML capture and what was missed?

Part 2: To what extent can the ISML abstract the USC metaphor?

By posing these questions, some determination can be made with regard to the fitness of the ISML framework for capturing the design of the USC prototype and also its ability to separate metaphorical aspects from other design views. [Section 2 addresses part 1 of analysis question 2; whilst section 3 addresses part 2].

2.1 Task

High-level task groups were initially established resulting in similar collection of basic tasks from each group, see Figure 43.

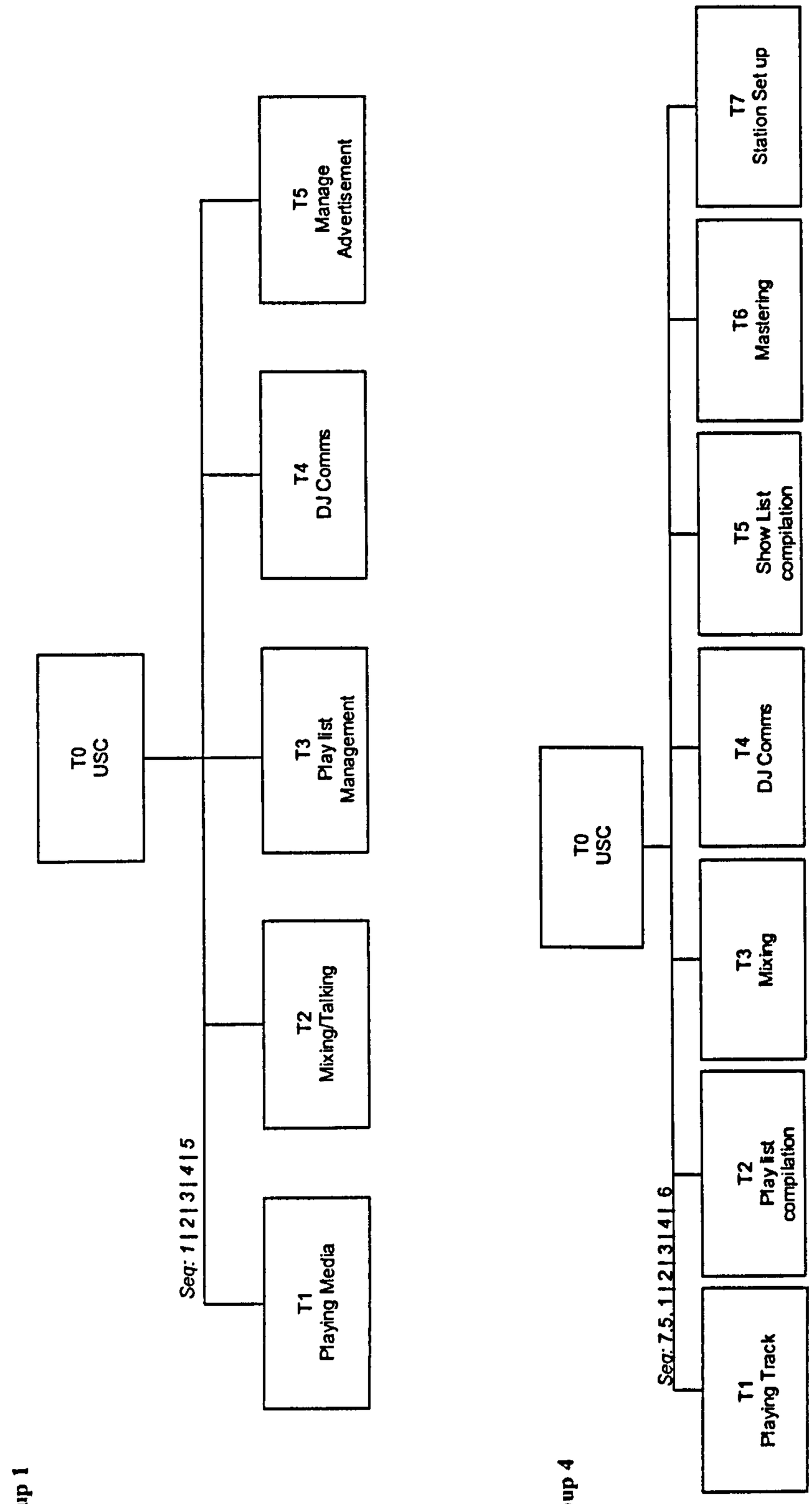


Figure 43 USC top level tasks

The following task groups from each design team are sufficiently similar to afford comparison:

	Group 1	Group 4
Playing Media	T1	T1
Mixing	T2	T3 and T6
Play list	T3	T2
DJ Communication	T4	T4

Table 15 USC common task groups

Each task group is considered in more detail in sections 2.1.1-2.1.4. In addition to these common tasks, advertisement management (group 1) and station set-up (group 4) were also modelled at a very simplistic level. Whilst advertisement management is considered in more detail later on in the ISML framework, both of these additional task groups are not considered further here since they were not developed beyond a cursory level during elicitation.

After the initial task elicitation had taken place, a verification exercise took place to confirm the overall structure and task objects and actions. The result is an inventory of objects, actions and 'stop-iterate' conditions (a state, which when reached, indicates the end of some task). Each object will be considered as a potential meta-object candidate in the task specification. During the elicitation, all actions were discussed from the point of view that the DJ or producer would enact them. A summary of the task models is given in appendix L.

Action names do not necessarily match up with the 'leaf' nodes in the task model and so are cross-referenced against node numbers as well as the objects that are subject to them. Additional verification of each object follows, detailing the actions directed toward it and the associated task nodes. Finally, 'stop-iterate' conditions are referenced against the task nodes and objects to which they apply; the test condition is specified against the target attribute, state or object set. Each stop-iterate condition is formally re-specified as a mapping-constraint class. The summary is by no means complete or rigorous – its principal use is to serve as a basis for further discussion during the design process and as a means of documentation to aid formal ISML specification. In examining the four common task sub-groups, some of the weaknesses of the initial task summaries can be identified.

2.1.1 PLAYING MEDIA

Both groups realise a similar model for playing media and make some implicit assumptions regarding the over-all task model. This can be illustrated in the way both groups handle the concepts of tracks, media and media players.

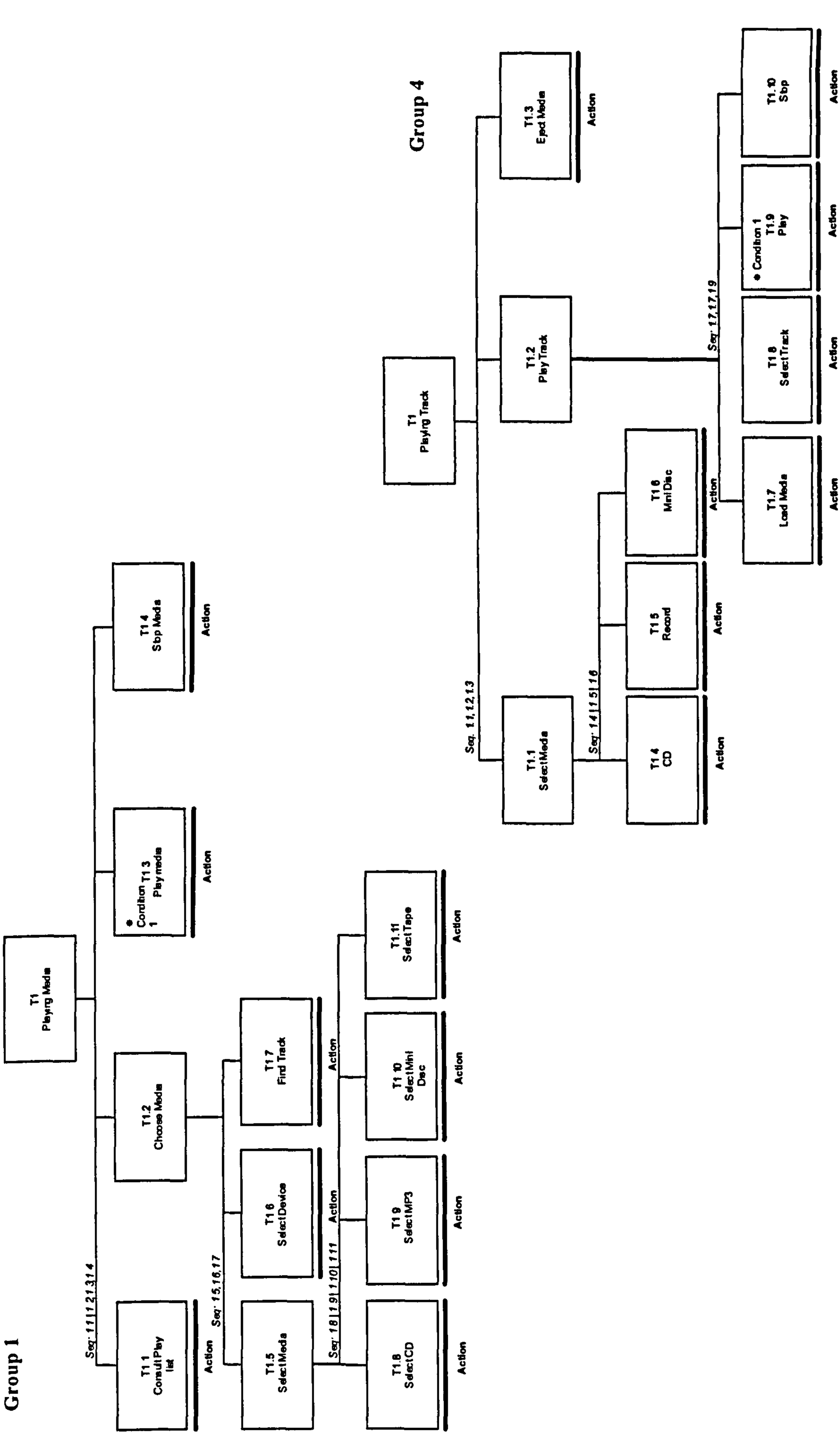


Figure 44 USC Playing media tasks

Group 1 makes an explicit reference to the media playing device object, but does not relate the media object (such as a CD) to the device itself through any loading action and implicitly refers to a media 'track'[25] (not an explicit object). Conversely, group 4 makes explicit references to media and track objects but implicitly refers to a media-playing device (see Figure 44). In both cases, the treatment of the playing device was light; no considerations were made as to the state of the device before the interaction commences (the device could already be playing). However, Group 4 describes a task model that ameliorates this problem to some degree by including an 'eject' action at the end of the task sequence. Both groups identified a stop-iterate condition on the play action for this task in which iteration stops upon a track reaching a 'stopped' state.

2.1.2 MIXING

In sharp contrast to group 4, group 1 is highly simplistic and relates only to localised actions taken by the DJ. Group 4 refer to two types of mixing – that performed by both the DJ or producer (who may also act as a DJ) in task group T3 and that of 'mastering', managing the output from other DJs to 'air'.

[25] In fact, a single reference to a track is made in stop-iterate condition 1, but during the elicitation this was not further expanded.

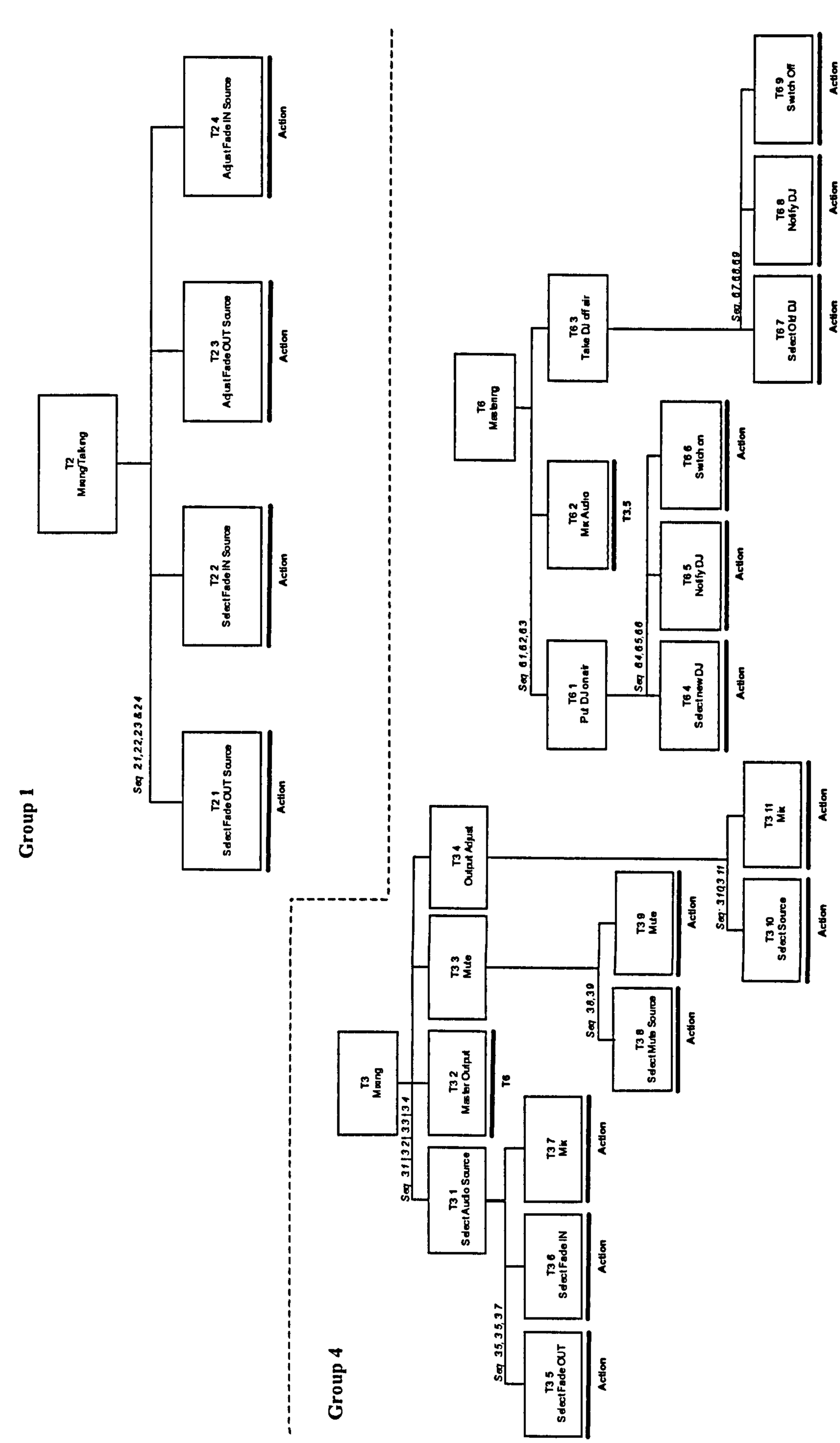


Figure 45 USC Mixing tasks

It becomes apparent that within the 'mixing' task, each group relies on a number of complex concepts which, whilst understood by the design team at an informal level, are implicit and difficult to express within the ISML framework. Notably, these include 'audio sources' (found in both groups) and being 'on air' – neither of these concepts is explicitly expressed in the task summary. The 'air' and 'audio sources' are intangible and yet important concepts to the USC project. The use of audio sources in this task also implies the channelling of audio data from one distinct entity to another. For group 1, their design determined this as transmission from the media player devices to the mixer desk; group 4 do not make any explicit reference to an object here (see chapter 5). Mixing is extended to mixing DJ transmissions to air (the producer's role), referred to as 'mastering' by group 4, see Figure 45. This sub-model outlines the process for the producer, however the actions of notification and DJ 'activation' and 'de-activation' are not 'unpacked'.

2.1.3 PLAY LIST

A new problem in the expression of tasks within the ISML framework begins to emerge with the specification of the use of the 'play list'; the symptom arises in both groups' model however group 4 is helpful in expressing it in their task description. Specifically, the songs or tracks to be played are references to other objects within the task environment and are not readily considered as objects in their own right.

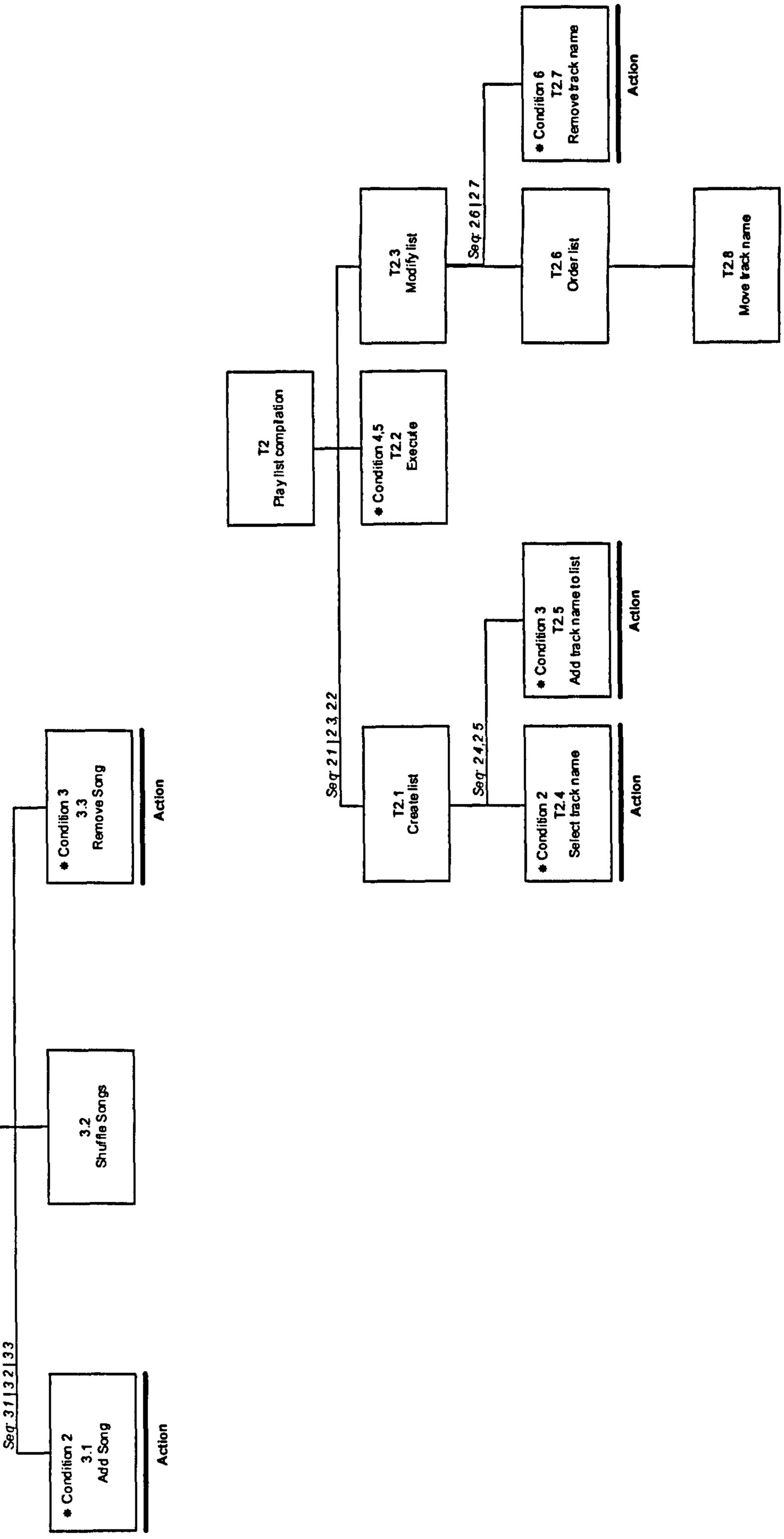


Figure 46 USC Play list tasks

This is apparent in the node descriptions from group 4 in which track 'names' are manipulated, rather than the tracks themselves, see Figure 46. At this point, the concept of 'track' also requires re-examination since, as seen in chapter 5, a track in the context of the USC is more an abstract object than a physically tangible one. Particularly, songs or tracks are observed and manipulated in the DJ environment through interaction with other devices – so the track object has more than just a single relationship with the physical media with which it ordinarily associated.

Although identified in the meta-object model, the source of tracks in both groups remains unspecified in the task view. A number of stop-iterate conditions have been applied in both cases, giving clues as to some of the potential attributes of the play list object including a containment set (group 1: condition 3 and group 4: condition 6). Maximum 'play time' or track count property for the list (group 1: condition 2 and group 4: condition 3) is also specified. Group 4 makes more extensive use of stop-iterate conditions to describe the task group, although their application implies more detail than is explicitly stated. Conditions 4 and 5 in T2.2 terminate an ambiguous 'execute' task not expanded any further in this model. A trailing node, T2.8 in this model is the result of the removal of an extraneous 'shuffle' sub-task during the verification exercise.

2.1.4 DJ COMMUNICATION

The problems encountered in the mixing task group re-emerge in DJ communication: 'air' and 'voice channels' appear in the model developed by group 1 whilst no explicit medium through which DJs communicate with each other is expressed by group 4.

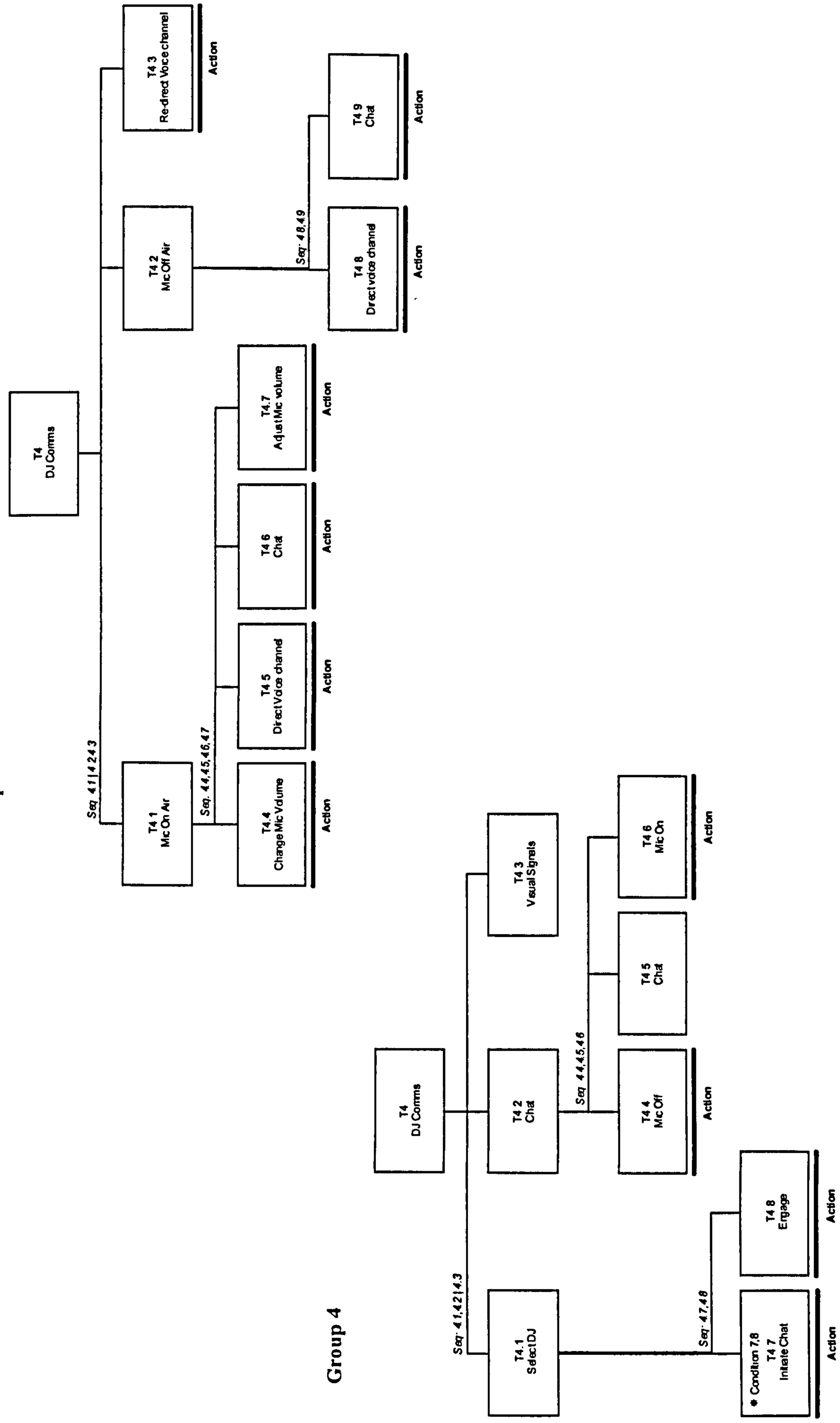


Figure 47 USC DJ communication tasks

A failure in object verification becomes apparent in the summary given by group 1 as a microphone is in use in the task model, but fails to be ratified in the task inventory. In contrast to group 4, the microphone has a volume property (although no further details are given) rather than a binary on or off state. In both cases, the engagement with another DJ is highly ambiguous – involving either 'chat initiation' and 'visual signals' (group 4) or 'directing voice channel' (group 1), see Figure 47. DJs in both designs use the microphone to communicate verbally and it was recognised by both groups that the need to avoid broadcasting unwanted chatter had to be addressed. Group 4 specified a perhaps more succinct model in that the microphone is 'off-air' (to avoid unwanted broadcasts) is ensured through action (T4.4) rather than a condition (group 1, T4.1[26]). Some attempts to qualify the behaviour of the DJ and producer is made using stop-iterate conditions (references 7 and 8 in the summary). However the combination of implicit references to audio sources and the complexity of real-world gestures defeats expression in ISML.

2.2 Meta-object

Once again, the summary interactively reached with each group is incomplete and requires further post-interview analysis before a formal ISML meta-object specification can be developed. Discussion of the metaphor design was at a high and informal level (see chapter 5 regarding the elicitation method) but guided by the structures needed for ISML development. The result of the meta-object elicitation strategy is an inventory of objects, actions and consequences of those actions with respect to the metaphor design only, see appendix L for a summary.

The elicitation of the metaphor design marks the beginning of significant divergence between the design groups with respect to diversity and completeness of the proposed metaphor environment. Group 4 maintained a focus specifically on the 'core' features on the environment - the DJs, media objects and the play list, whilst group 1 broadened their design remit to include advertisement and other miscellaneous objects.

2.2.1 TRACK

The track entity specified by both groups is simple and with the exception of the editing state, expressed by group 4, is a passive object – see Figure 48. Changes to the track's title are effected through the play list meta-object (see section 2.2.4) and implemented through the higher level interactor specification.

---

[26] A stop-iterate condition testing the state of the microphone would have been appropriate here, but this was not developed during design discussions.

Figure 48 USC Track model

Metaphor Object: MP3

Entity encapsulating a digitally stored music track

HTA links

Unspecified

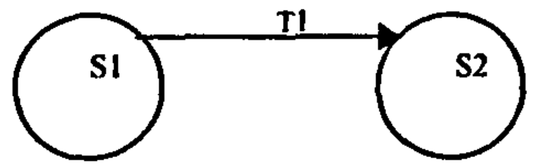
Efferent Actions

Name	Source(s)	Subordinate focus object(s)	Consequence(s)
Select	Inventory list Play list	None	Activated

Attributes

STRING title

States



S1 = InActive S2 = Activated	T1 = ACTION Select

Sets

None

Group 1

Metaphor Object: Media Track

Visual manifestation of abstract soft media track

HTA links

None specified

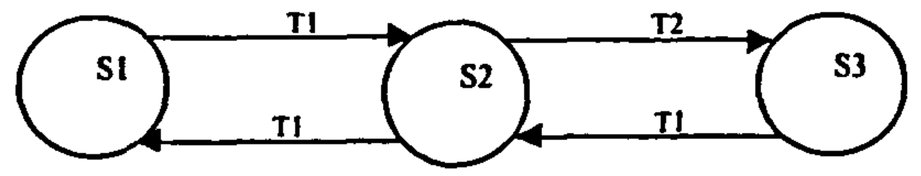
Efferent Actions

Name	Source(s)	Subordinate focus object(s)	Consequence(s)
Hi-light	Play list	None	Track Hi-lighted
Edit	Play list	None	Changed Text

Attributes

STRING trackDetails

States



S1 = Non Lit S2 = Lit S3 = DetailsChanging	T1 = ACTION Hi-Light T2 = ACTION Edit

Sets

None

Group 4

With the insights to the nature of the track object discussed in chapter 5, it is not surprising to find that, for group 1, the track entity does not feature in the media player object or that, for group 4, its relationship within summary of the media player is unspecified.

2.2.2 MEDIA PLAYER

Treatment of the media playing object by each group is illustrative of the hierarchical design of group 1 versus the 'encapsulated complexity' strategy of group 4 – this is also apparent in the mixer and play list objects.

Metaphor Object: Hi-Fi

Houses CD, Tape, and Mini Disc (physical) devices

HTA links

1.3-1.8, 1.10, 1.11

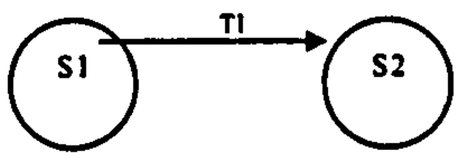
Efferent Actions

Name	Source(s)	Subordinate focus object(s)	Consequence(s)
Select	DJ	Media Player Device	Activated

Attributes

None

States



S1 = Inactive S2 = Active	T1 = ACTION Select

Sets

MediaPlayerDevices

Metaphor Object: Media player device

Encapsulation of a specific media playing device

HTA links

Unspecified

Efferent Actions

Name	Source(s)	Subordinate focus object(s)	Consequence(s)
None specified			

Attributes

None

States

None

Sets

MediaPanel

Metaphor Object: Media player device panel

Operational interface to media players (CD, Mini Disc, Tape)

HTA links

Unspecified

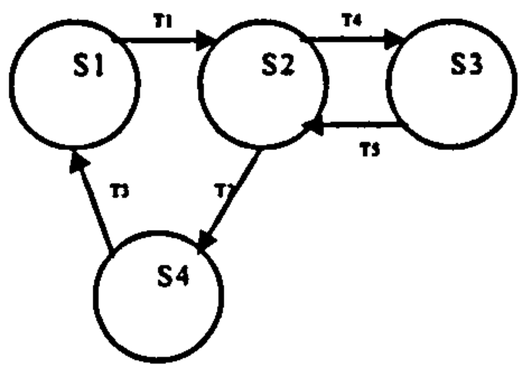
Efferent Actions

Name	Source(s)	Subordinate focus object(s)	Consequence(s)
Play	DJ	Play button	Local play
Stop	DJ	Stop button	Stop local play
Pause	DJ	Pause button	Unspecified
Skip	DJ	Skip button	Unspecified

Attributes

None

States



S1 = Stopped S2 = Playing S3 = Paused	T1 = ACTION Play T2 = ACTION Stop T3 = Transient T4 = ACTION Pause T5 = ACTION Pause

Sets

ButtonItems

Figure 49 USC Group 1 Media device model

Group 1 describes a relatively empty hierarchy in which a 'Hi-Fi' object maintains a set of media player devices, which in turn contain a set of media panels, see Figure 49. Each panel is then used as an interface for the playing of media objects such as a CD or mini-disc. This model is incomplete in that not all the consequences of actions are addressed and as a result of these admissions: the state model can only be partially completed. The concept of 'local play' (which must be clarified through recourse to the interview data) refers to non-streamed audio playback (i.e., the player device in this case is not sending audio to the producer).

Metaphor Object: Media Player

Playing and manipulating media tracks

HTA links

1.1-1.3,2.2

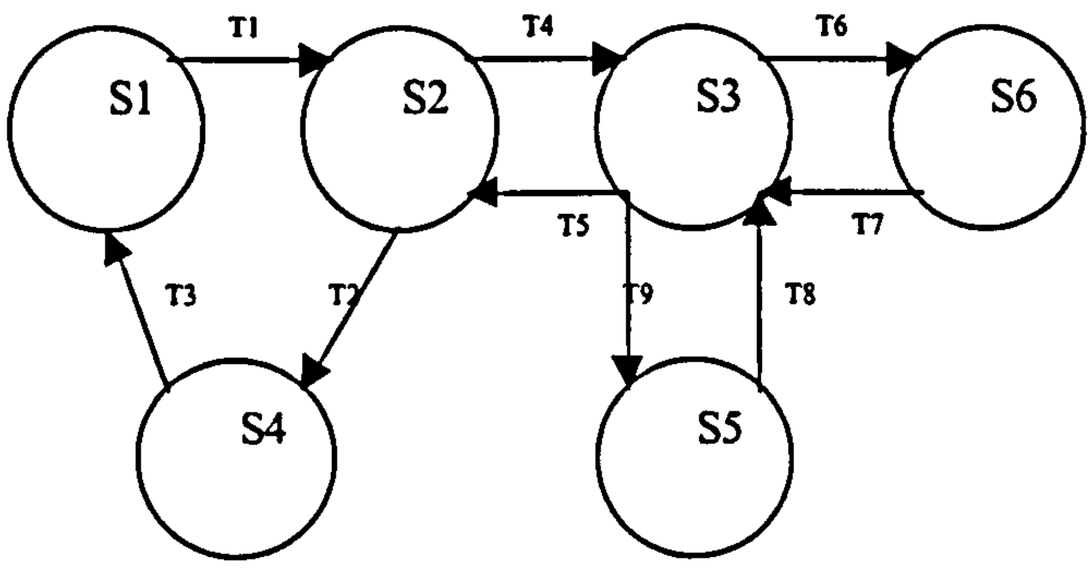
Efferent Actions

Name	Source(s)	Subordinate focus object(s)	Consequence(s)
Play	DJ	Play button	Device Plays
Stop	DJ	Stop button	Device Stops
Pause	DJ	Pause button	Device Paused
Skip Forward	DJ	Forward button	Device Winds forward
Skip Backward	DJ	Backward button	Device Winds backward

Attributes

None

States



S1 = Stopped S2 = Playing S3 = Paused S4 = Wind To Start S5 = Winding Forward S6 = Winding Backward	T1 = ACTION Play T2 = ACTION Stop T3 = Transient T4 = ACTION Pause T5 = ACTION Pause T6 = ACTION Skip Backward T7 = ACTION Play T8 = ACTION Skip Forward T9 = ACTION Play

Sets

currentTrack
mediaButtons

Figure 50 USC Group 4 Media player model

Comparatively, group 4 describes a more abstract version of the media player that makes no reference to the real-world concepts or structure that group 1 employs, see Figure 50. A relatively close mapping between the actions executed upon the object and its resultant state is specified. A 'media buttons' set maintains the collection of subordinate control objects for this version of the media player, whilst a 'current track' set is used to hold the track object being played.

2.2.3 MIXER

The marked difference in specification style continues for the mixer object, although the mixer design developed by group 1 would appear superficially more complicated since it has more subordinate parts, two of the objects captured in the summary documentation appear to be little more than affectations (see Figure 51). Group 4 suggested a mixer design that comprises of only one parent meta-object and three slider children (see Figure 52).

Figure 51 USC Group 1 Mixer model

Metaphor Object: Mixer desk

Controls the audio output that goes to the producer's mixer desk

HTA links

T1 3, 1.4, 1 6, 2.1-2.4, 4 1,4.2, 4.4, 4.7

Efferent Actions

Name	Source(s)	Subordinate focus object(s)	Consequence(s)
Slide	DJ/Producer	Sliders (mic,CD,miniDisc,MP3)	Unspecified
Push	DJ/Producer	Buttons (mic,CD,miniDisc,MP3)	Unspecified

Attributes

INTEGER micVolume
INTEGER cdVolume
INTEGER miniDiscVolume
INTEGER MP3Volume

States

None

Sets

MicControlSet
CDControlSet
MiniDiscControlSet
MP3ControlSet

Metaphor Object: Button

A bi-state button

HTA links

Unspecified

Efferent Actions

Name	Source(s)	Subordinate focus object(s)	Consequence(s)
Deactivate	Mixer desk	None	Pushed
Push	Mixer desk	None	Pushed

Attributes

None

States

S1 = On S2 = Off	T1 = ACTION Push | ACTION Deactivate

Sets

None

Metaphor Object: Slider

Virtually movable slider objects

HTA links

Unspecified

Efferent Actions

Name	Source(s)	Subordinate focus object(s)	Consequence(s)
Slide	Mixer desk	None	ContrainedVerticalMove

Attributes

INTEGER minimumY
INTEGER maximumY
INTEGER value

States

None

Sets

None

Metaphor Object: Light

Feedback for mixer desk

HTA links

Unspecified

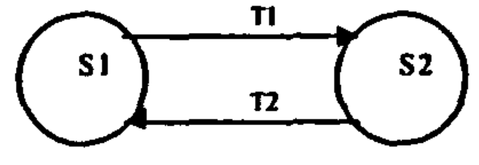
Efferent Actions

Name	Source(s)	Subordinate focus object(s)	Consequence(s)
Activate	Mixer desk	None	Activated
Deactivate	Mixer desk	None	Deactivated

Attributes

None

States



S1 = Activated S2 = Non-Active	T1 = ACTION Activate T2 = ACTION Deactivate

Sets

None

Metaphor Object: DJ Mixer

Changes audio OUT properties

HTA links

3.1,3.2,3.4

Efferent Actions

Name	Source(s)	Subordinate focus object(s)	Consequence(s)
Drag Vertical	DJ	Volume/Bass/Treble Slider	Modified Power

Attributes

INTEGER volumePower
INTEGER bassPower
INTEGER treblePower

States

None

Sets

SliderSet

Metaphor Object: Slider

HTA links

None specified

Efferent Actions

Name	Source(s)	Subordinate focus object(s)	Consequence(s)
Drag Vertical	DJ Mixer Master Mixer	None	Modified YPosition

Attributes

INTEGER yMinimum
INTEGER yMaximum
INTEGER yPosition

States

None

Sets

None

Figure 52 USC Group 4 Mixer model

In both cases, values for each of the audio volume inputs are stored as attributes whilst subordinate slider meta-objects independently maintain relative local values. Similarly, both mixer objects receive slide actions from the DJ and pass them on to the contained sliders.

Whilst both groups define audio volume values within the main mixer object and identify a constraint on slider manipulation ranges, only group 4 explicitly defines a relationship between the slider's value and the audio values belonging to the mixer.

2.2.4 PLAY LIST

The number of objects used to model the play list is of a similar ratio to that used for the mixer. In this case, group 1 makes two distinctions to their abstraction of the play list; the first (the MP3 rack) describes likeness to an object, the second (inventory and play list) describes function. The MP3 rack acts primarily as a parent object containing the two track lists in a set called 'lists' – see Figure 53. From the summary elicitation, it is unclear as to what relationship the rack's state of activation is to the child list object it maintains (this requires refinement, see section 3).

Metaphor Object: MP3-Rack

Holds a list of songs in order that are modifiable and that the DJ plays.

HTA links

Not specified

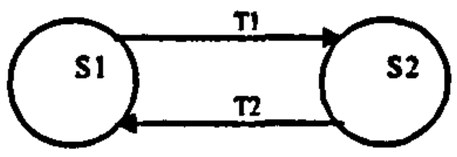
Efferent Actions

Name	Source(s)	Subordinate focus object(s)	Consequence(s)
Activate	DJ	None	None
Deactivate	DJ	None	None

Attributes

ATTRIBUTE mp3Count

States



S1 = Non-active S2 = Active	T1 = ACTION Activate T2 = ACTION Deactivate

Sets

Lists

Metaphor Object: Inventory List

Static list of MP3 objects

HTA links

1.1-1.2, 1.7, 1.9

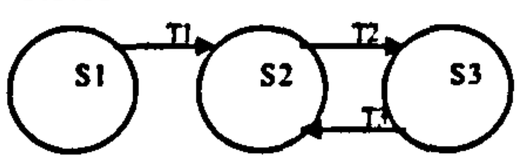
Efferent Actions

Name	Source(s)	Subordinate focus object(s)	Consequence(s)
Select	DJ	MP3	None

Attributes

None

States



S1 = No MP3 selected S2 = Unselect last MP3 S3 = Selected MP3	T1 = ACTION Select T2 = Transient T3 = ACTION Select

Sets

MP3Items
SelectedMP3
LastSelectedMP3

Metaphor Object: Play list

Ordered list of MP3 objects

HTA links

3.1-3.3

Efferent Actions

Name	Source(s)	Subordinate focus object(s)	Consequence(s)
Select	DJ	MP3	None
Add	DJ	None	Added Item
Remove	DJ	None	Removed Item

Attributes

INTEGER playListCount

States

S1 = Empty S2 = Non-empty S3 = No MP3 selected S4 = Unselect last MP3 S5 = Selected MP3	T1 = ACTION Add T2 = ACTION Remove T3 = ACTION Select T4 = Transient

Sets

MP3Items
SelectedMP3
LastSelectedMP3

Figure 53 USC Group 1Play list model

In modelling their play list, group 1 makes an important distinction between a source of track objects (the inventory list) to choose from and the final selection (the play list proper). Both lists contain a set of MP3 items (track objects), the contents of which change as 'select' actions are passed to potentially selectable MP3 objects contained within each list. Once again, this model is only partially specified by the elicited summary and requires refinement (see section 3).

Group 4 build a play list model (see Figure 54) that whilst being more sophisticated also exposes some fundamental problems with metaphor model construction using the ISML framework (see chapter 5).

Metaphor Object: Play list

Encapsulation of tracks, with order

HTA links

1.1,2.1-2.3

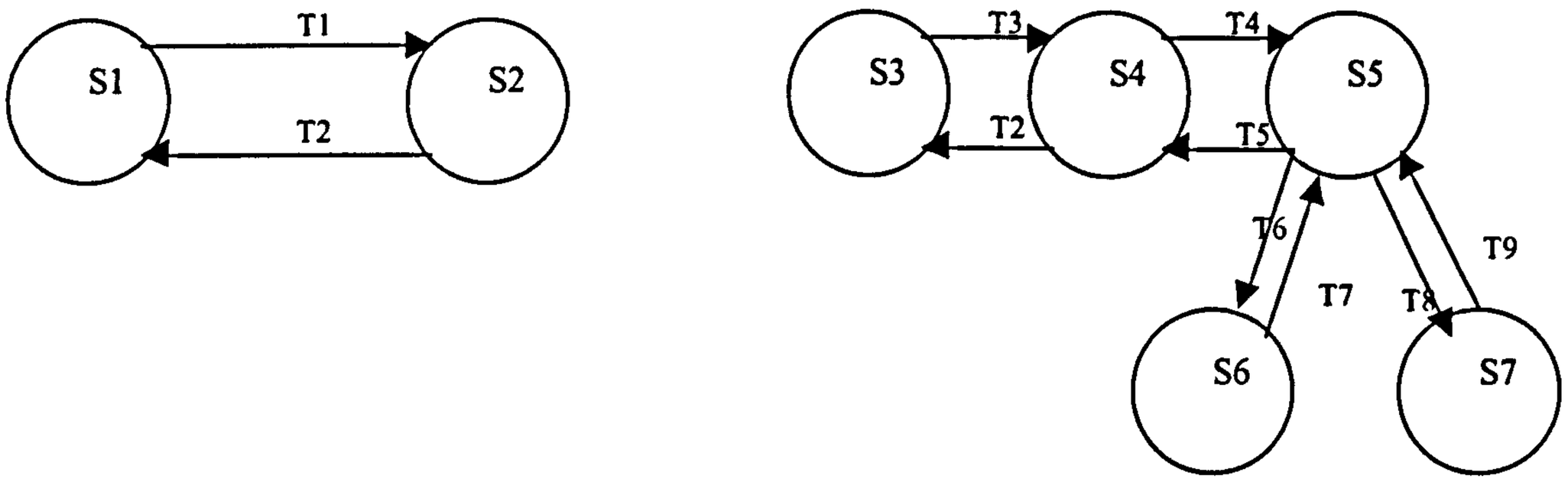
Efferent Actions

Name	Source(s)	Subordinate focus object(s)	Consequence(s)
Add Track	DJ	{none ~ abstraction problem}	Track Added
Remove Track	DJ	Button	Track Removed
Move Up	DJ	Button	Track Moved Up
Move Down	DJ	Button	Track Moved Down
Hi-light	DJ	Track	None
Edit	DJ	Track	None

Attributes

INTEGER totalNumberOfTracks

States



S1 = No track items S2 = Track items available S3 = No track selected S4 = Unselected last play list item S5 = Selected play list item S6 = Play list item moved up S7 = Play list item down	T1 = ATTRIBUTE totalNumberOfTrack > 0 T2 = ATTRIBUTE totalNumberOfTrack < 1 T3 = ACTION Hi-light T4 = Transient T5 = ACTION Hi-light T6 = ACTION Move Up T7 = Transient T8 = ACTION Move Down T9 = Transient

Sets

playList
selectedPlayListItem
lastSelectedPlayListItem
playListButtons

Figure 54 USC Group 4 play list model

The play list described here uses two parallel state models to specify the availability of tracks for manipulation within the list (S1 and S2) as well as the effect hi-light and movement actions have on the track items maintained by the 'list' sets (see section 3 for refinement). Two unusual features appear in the abstraction of the play list developed by group 4: 1) the group could not specify how tracks became added to the list in this part of the ISML framework and 2) 'buttons' appear to be a component part of this (metaphorical) object. The reason for the absence of a source of tracks from which a DJ can construct his/her play list was revealed later in the interactor elicitation stage (see chapter 5).

2.2.5 OUTSTANDING OBJECTS

Each group developed their designs in unique directions that can be seen, in part, by the outstanding objects included in the summary:

Group 1	Group 4
Bookshelf Book Advert Time line Monitor	DJ Object/Profile Microphone Microphone stand Master mixer

Table 16 USC Outstanding objects

2.2.5.1 Group 1 outstanding objects

Unlike their sister team, group 1 addressed advertisement management in their design and, consistent with their specification style, developed a hierarchical collection of metaphor objects – see Figure 55. In addition to the development of advertisement objects, group 1 also declared a 'monitor' object which was not sufficiently developed further in the meta-object summary and so will not be considered further here.

Figure 55 Group 1 Advertisement model

Metaphor Object: Bookshelf
Holds advertisement book.

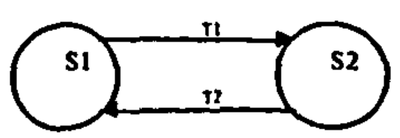
HTA links
5.1-5.4

Efferent Actions

Name	Source(s)	Subordinate focus object(s)	Consequence(s)
Activate	DJ	Book	Activated
Deactivate	DJ	Book	Deactivated

Attributes
None

States



S1 = Non-active S2 = Active	T1 = ACTION Activate T2 = ACTION Deactivate

Sets
Books

Metaphor Object: Time line
Ordered list of advertisements to be broadcast.

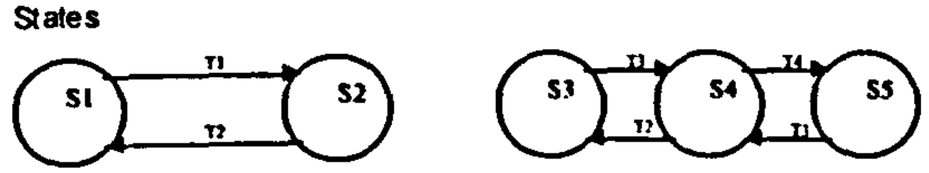
HTA links
5.1-5.4

Efferent Actions

Name	Source(s)	Subordinate focus object(s)	Consequence(s)
Select	Book	Advert	None
Add	Book	Advert	Added item
Remove	Book	Advert	Removed item

Attributes
INTEGER advertTotal

States



S1 = Empty S2 = Non-empty S3 = No Advert selected S4 = Unselect last Advert S5 = Selected Advert	T1 = ACTION Add T2 = ACTION Remove T3 = ACTION Select T4 = Transient

Sets
AdvertItems

Metaphor Object: Book
Contains Advert objects and the Timeline

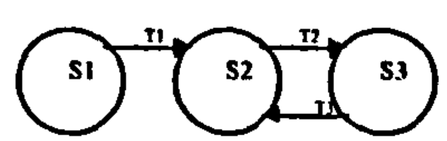
HTA links
5.1-5.4

Efferent Actions

Name	Source(s)	Subordinate focus object(s)	Consequence(s)
Select	DJ	Advert Time line	None
Add	DJ	Time line	None
Remove	DJ	Time line	None

Attributes
None specified

States



S1 = No Advert selected S2 = Unselect last Advert S3 = Selected Advert	T1 = ACTION Select T2 = Transient T3 = ACTION Select

Sets
TimeLine
AdvertItems
LastSelectedAdvert

Metaphor Object: Advert
Entity encapsulating advertisement details

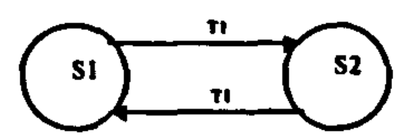
HTA links
Unspecified

Efferent Actions

Name	Source(s)	Subordinate focus object(s)	Consequence(s)
Select	Book Time line	None	None

Attributes
STRING advertDescription

States



S1 = Activated S2 = Non-Active	T1 = ACTION Select

Sets
None

In their advertisement model, a bookshelf maintains a collection of books – each book contains a 'time line' object (in set 'time line') and a selection of 'advert' objects (stored in the 'advert items' set). The specification of the book is very similar to that of the play list group: the book behaves like the inventory list whilst the time-line mimics the play list. However, subtle differences can be found between these two models. The MP3 rack acts as a parent object to the two lists but does not specify any further relationship between the two. However, the book is effectively a list type object that itself contains a further list-like object: the time-line (both book and time-line maintain a list of advert objects).

Figure 56 USC Group 4 producer model

Metaphor Object: DJ Object / DJ Profile
A representation of the DJ (linked with the DJ object)

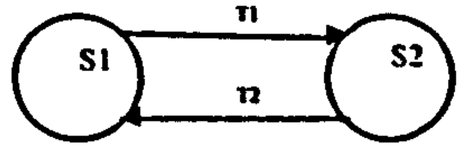
HTA links
4.2,5.1-5.2,7.2, 5.1,5.3,6.1-6.3,7.2

Efferent Actions

Name	Source(s)	Subordinate focus object(s)	Consequence(s)
Pick style	DJ	unspecified	Changed Style
Change name	DJ	unspecified	Changed Text
Volume change	Master Mixer	none	Modified Power
Put On Air	Microphone	none	Switched On Air
Put Off Air	Microphone Microphone stand	none	Switched Off Air
Drag 2D	Producer	none	Modified XY Position

Attributes
STRING name
INTEGER xPosition
INTEGER yPosition
INTEGER volume
STRING style

States



S1 = On Air
S2 = Off Air

T1 = Efferent Action Put Off Air
T2 = Efferent Action Put On Air

Sets
None

Metaphor Object: Master Mixer (Producer)
HTA links
None specified

Efferent Actions

Name	Source(s)	Subordinate focus object(s)	Consequence(s)
Drag Vertical	Producer	Volume Slider	Modified Power

Attributes
INTEGER volume

States
None

Sets
mixerSlider

Metaphor Object: Microphone
Used by the Producer to pass broadcast focus between DJs.

HTA links
None specified

Efferent Actions

Name	Source(s)	Subordinate focus object(s)	Consequence(s)
Drag 2D	Producer	None	Modified XY Position

Attributes
INTEGER xPosition
INTEGER yPosition

States
None

Sets
None

Metaphor Object: Microphone stand
Dock for microphone, taking all DJs off air.

HTA links
None specified

Efferent Actions

Name	Source(s)	Subordinate focus object(s)	Consequence(s)
All Off Air	Microphone	none	All off air

Attributes
None

States
None

Sets
studioDJs

During the metaphor model elicitation, group 4 enthusiastically pursued the producer's metaphor model (Figure 56).

The four objects in this group interact with each other to provide the producer with basic audio control facilities, however without recourse to the qualitative analysis the summary does not reveal sufficient detail to understand the model. Each DJ interacting with the USC prototype has an associated metaphorical DJ object representing their presence within the virtual broadcast environment (not explicitly stated in the documentation). Also within this environment is a microphone and stand object; removing the microphone from the stand and giving it to a particular DJ gives him/her 'the air' to the exclusion of all other DJs. Replacing the microphone on the stand removes all access to the air from all DJs. The summary for this model partially describes this: the stand maintains a set of all DJs in the studio and receives 'all off air' actions from the microphone. A user acting as a producer may drag the microphone around the environment.

During elicitation, group 4 identifies not only changes in attribute properties, but also consequences of actions for the DJ, microphone and stand that can be later translated into state changes, mapping-constraints sets (and operations on them). However, the detail of these mechanisms through which the placing on and taking off of DJs from the air through the microphone is unclear and requires refinement. The main reason for this uncertainty is that only the 'efferent' actions are summarised for each object and so only provides a partial description of the complete model. Later discussion in the elicitation for the interactor model makes this mechanism (also used in a similar fashion by the master mixer, although not explicitly captured) significantly clearer, see section 2.4.

2.3 Devices and components

Relatively little time was spent focused on the details regarding the device and component parts of ISML since these details can be extracted from the initial project proposal and implemented prototypes respectively. The expected hardware for the USC prototype was an Intel compatible, networked, multimedia PC running Microsoft Windows™. No special input or output devices were required and in both groups' design of the DJ/Producer environment, only a standard keyboard and single button mouse are needed for user input. For this reason, the screen and mouse device previously defined in chapter 4 will be re-used for USC and to it, added a keyboard device:

```
<DEPipeDeclaration Name="keyboardDevice">
        <PipeAttrCaps>
                <AttrCap Name="Key">
                        <Type Type="STRING"/>
                        <Access Type="RO"/>
                </AttrCap>
        </PipeAttrCaps>
        <PipeFuncCaps>
                <FuncCap Caps="PIPE_KEYS" FuncName="GetKeyInfo"/>
        </PipeFuncCaps>
</DEPipeDeclaration>
```

The use of input devices was verified during the interactor elicitation process (see section 2.4). Having established the supporting devices for the USC prototype, a set of compatible components must be created to deliver the appropriate 'look and feel' for each group's design. An examination of the technical implementation of the graphical components used to implement those objects covered in the elicitation follows to isolate the requirements.

Figure 57 shows examples from both groups' prototype systems. To a limited extent, the interactive objects graphically reflect the metaphorical domain developed during the design process. ISML components only partially complete the metaphor specification as a whole (mostly in the visual sense) and so it is the graphical requirements of the interactive elements that are of principal concern here.

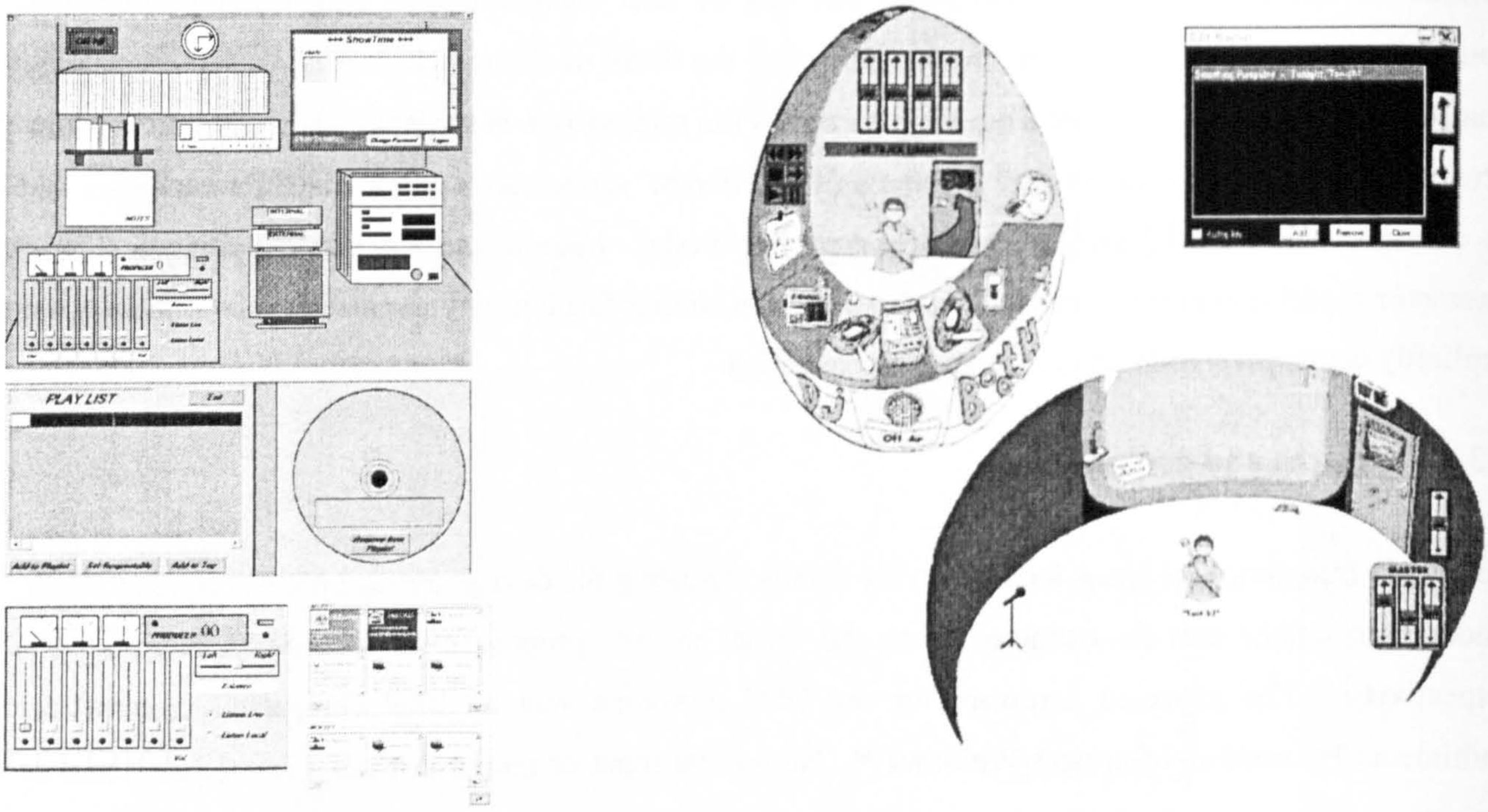


Figure 57 USC Prototype system screenshots

It is clear from both image sets that basic bitmap drawing capabilities are required. In addition to projecting passive graphical images, a series of components must be created that mimic the standard Microsoft Windows™ controls, including:

- Buttons (flat or radio or bitmapped)
- Text boxes
- Scroll bars

Simplest of all is the button, which has three basic states (armed, unarmed and in-focus) and may be qualified to appear as a radio button through the use of a Boolean flag. The rendering of text is more complex since it requires an algorithm to translate alphanumeric data into typographic imagery, displayed within a constrained two-dimensional box. The appearance of the scroll bar may be automatically generated from parameters determining its orientation, minimum and maximum extent and current position.

2.4 Interactor

This section will focus on the interactor implementation of objects already discussed by each group in previous meta-object elicitation stage. Preliminary screenshots were used as the basis for verification of the implementation of the metaphor objects elicited from previous meetings (see

Figure 58 for examples). In conducting this exercise, the interactive parts of the interface screenshots were identified and mapped to both the metaphor and task parts of the ISML framework. The known behaviours of the interface widgets used in each prototype implementation (Visual Basic or Borland C++ Builder) provide specific details regarding how the abstract metaphor design would be actualised. This review is to be used as a guide for the design of the interactor part of the ISML specification, rather than as an inventory of Microsoft Windows interface controls to be mimicked. It is worth noting that a considerable proportion of the screen space used in both prototypes is given over to the presentation of passive graphics that are intended to illustrate the virtual environment to the user.

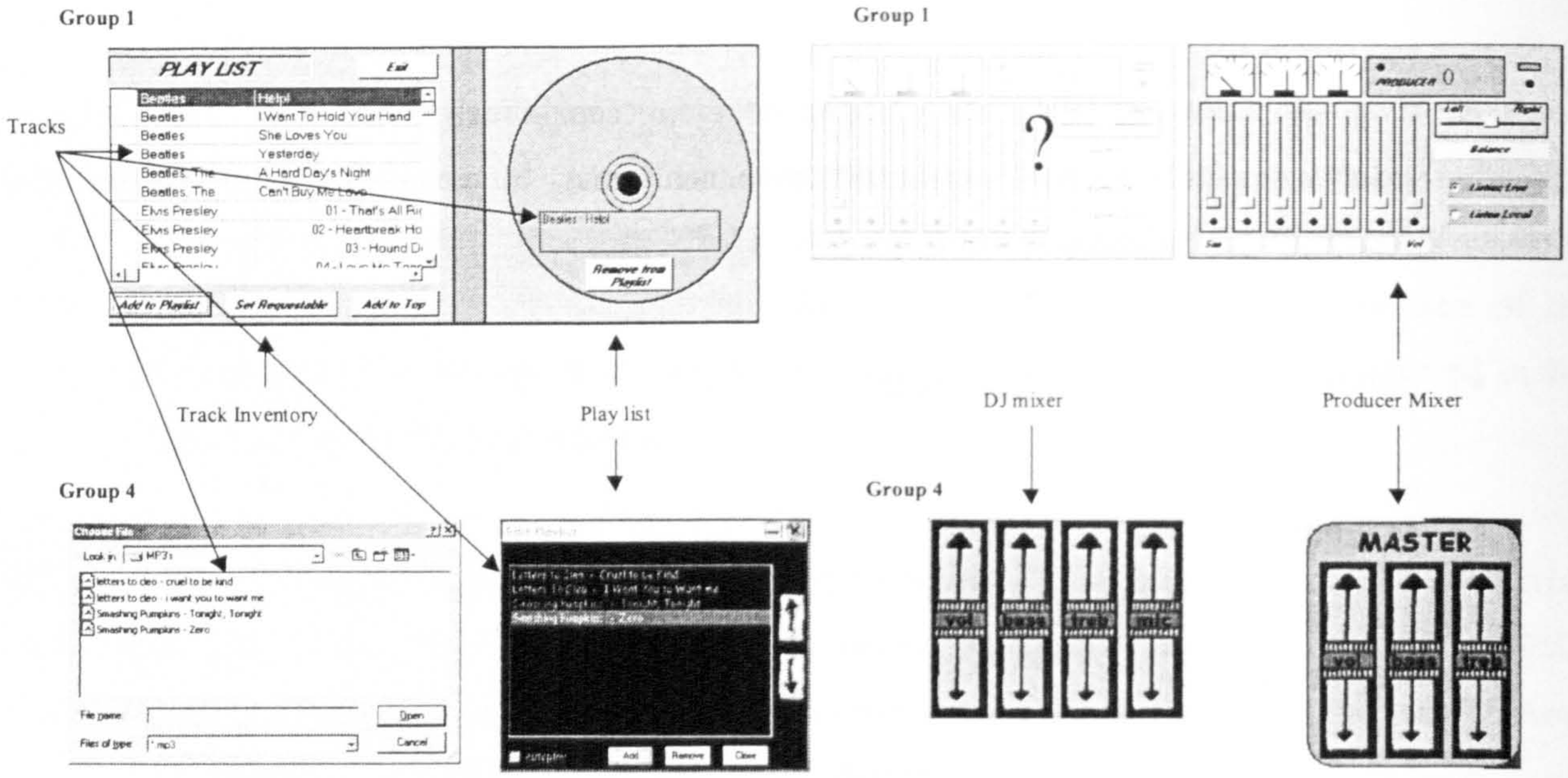


Figure 58 USC Play list and mixer interactors

The implementation of the underlying metaphor reveals the greatest amount of design divergence between the two design groups. Only two common meta-objects remain relatively consistent and comparable between groups – the play list and the sliders used on the mixers. However, even here disparities can be seen in both the implementation and their interaction styles. Track objects appear in all cases as text boxes in higher level container and dialogue components. The arrangement of text boxes in these various grids is a considerable challenge to the overall USC specification since the highly specific features of these components (largely particular to the Microsoft Windows environment) do not fit well with the USC metaphor.

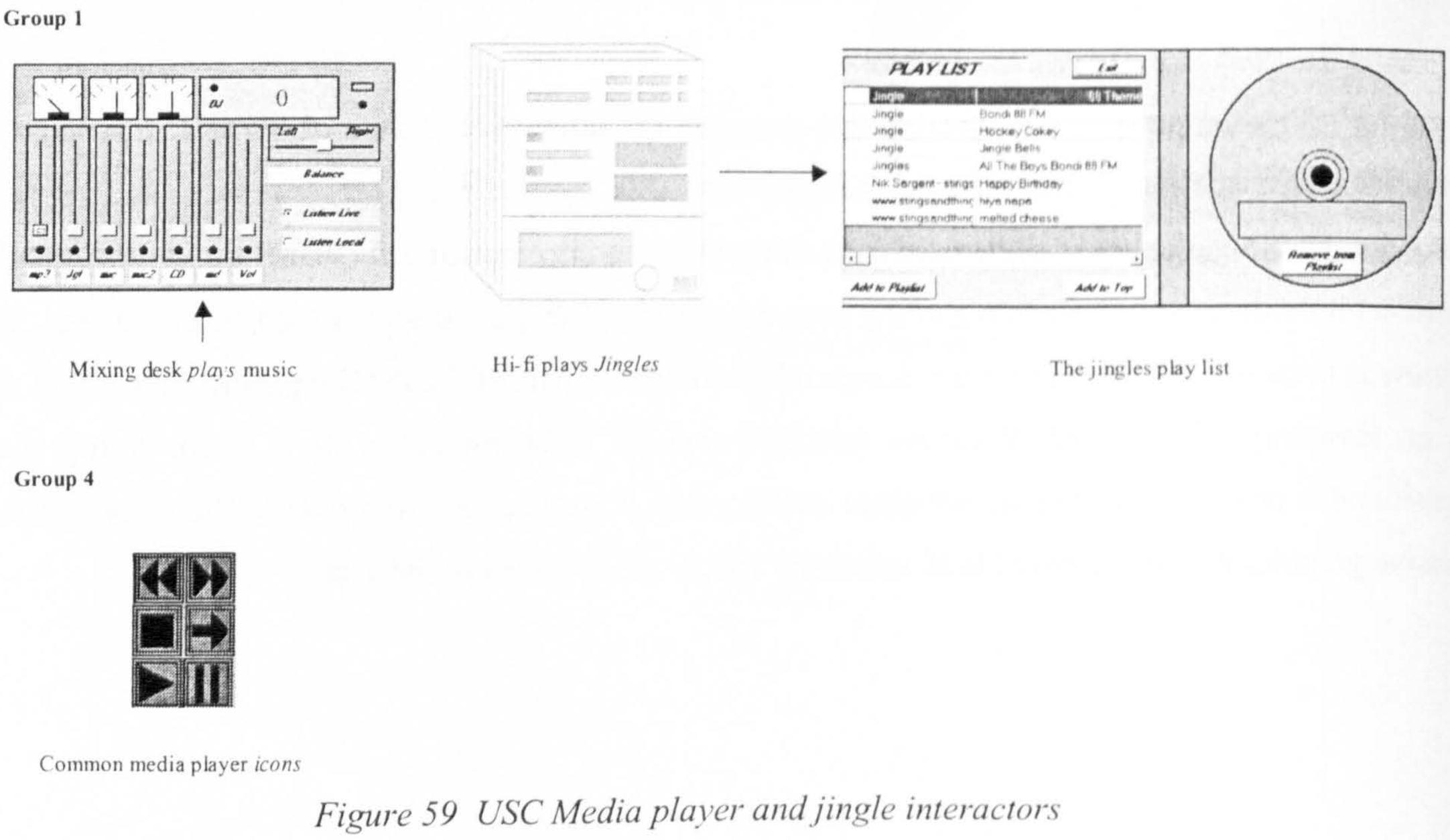
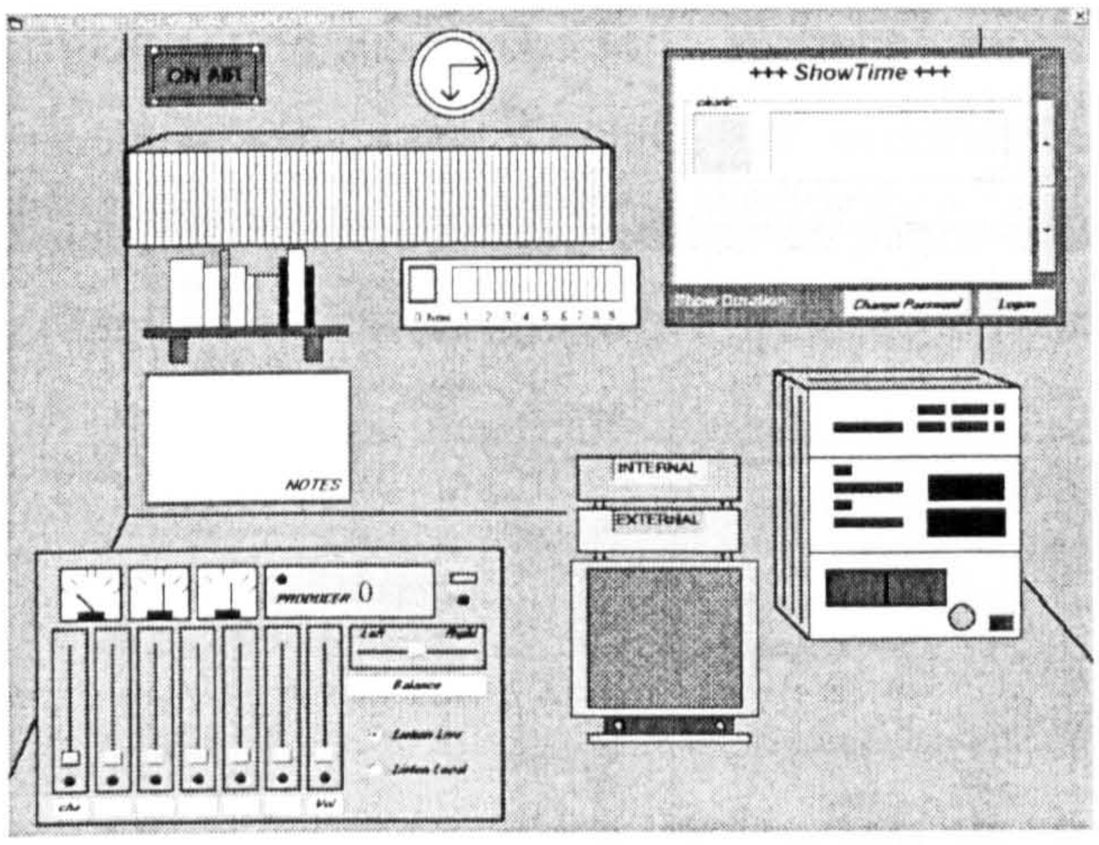


Figure 59 USC Media player and jingle interactors

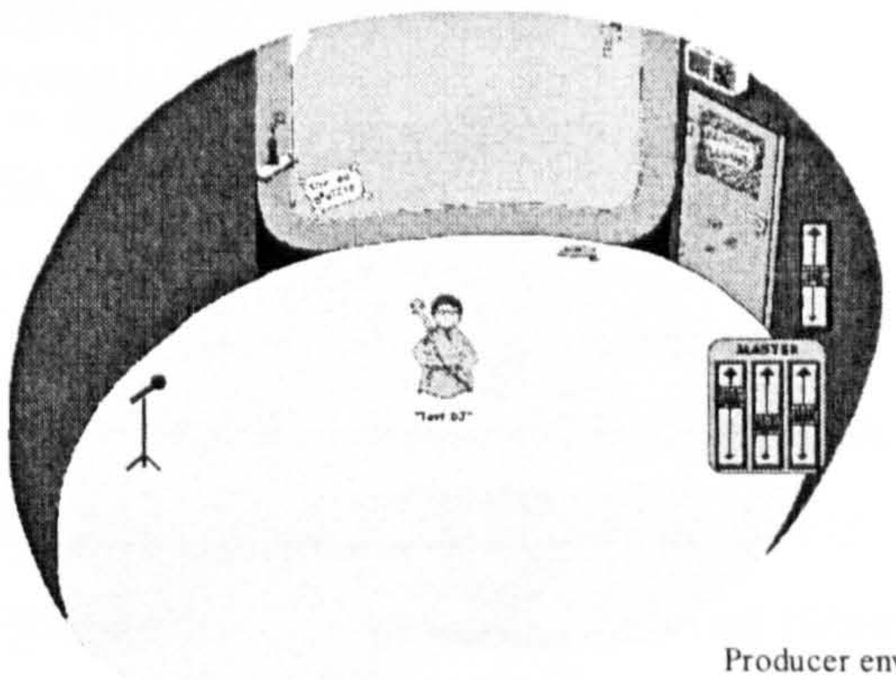
Buttons and slider bars are common implementation features of the mixing desk (see Figure 59); both groups modify the behaviour of standard button controls made available to them in their respective development environments. However, whilst group 4 implement smooth slide movements for all their sliders, group 1 only allow this type of interaction for the master volume slider – all other sliders effectively act as switches (clicking on the button moves the slider to maximum or minimum directly).



DJ/Producer environment

Group 1

DJ environment



Producer environment

Group 4

Figure 60 USC Producer environments

For both prototypes, media playing *operations* are very simple and only functional for MP3 files registered within the system. Group 1 uses the binary function of sliders to play and stop either music tracks or jingles (the two are exclusive – flipping one slider up drops the other down) from their independent play lists. Other potential audio sources are indicated on the mixer desk, but are non-functional. As well as media playing operations, a numeric counter indicates elapsed track time; a left/right audio balance changes the stereo reproduction of sound[27] and a radio button couple control DJ broadcast signal to the producer. A small circular button labelled 'DJ' in Figure 60 changes the context

[27] Arguably, this could be described as mixing but only in a very weak sense.

from a DJ to a producing role. In contrast, group 4 does not present the user with controls for multiple sources but instead uses a standard set of buttons, with familiar icons, to interact with an otherwise metaphorically 'invisible' media player.

Changing the environment from the DJ to the producer reveals further interactive objects. The mechanism for this change of context for group 1 is the clicking of a small button labelled 'DJ' on the mixer desk, the effect of which is to change the desk into a producer's mixer. A change in context in the prototype developed by group 4 actually changes the entire interface – the user clicks on the door to 'walk' into another room. Group 4 introduces the interactive 'DJ Object'[28] within the producer's environment - a bitmap graphic that is used in conjunction with a microphone and slider bar which both 'snap' to the DJ object in order to effect broadcasting ability and a change to the volume of their output.

Figure 61 USC Group 1 Advertisement book interactor

Group 1 implements a yellow book form to support advertisement management, see Figure 61. A 3x2 array of embedded windows containing a bitmap and short text description acts as the source of adverts. Each advert image may be dragged from its position over to the time-line (a form group labelled 'new set'), the effect of which is to place the advert at the end of the line. Standard buttons allow the user to navigate between pages, causing changes in the array or timeline as 'pages' are turned or the view along the line shifts.

In the tables below, each group's core meta-objects are listed against the interactor objects used to implement them.

MetaObject	Potential interactors
Track	Text box
Play list	Bitmap, text box, grid layout, button
Media player[29]	Bitmaps, text box, button (standard and radio)
Sliders	Bitmap, button
Book[30]	Bitmap, text box, button, grid layout
Advert	Bitmap, text box
Time line	Text box, grid layout

Group 1 Interactors

MetaObject	Potential interactors
Track	Text box
Play list	File dialogue, form dialogue, button
Media player	Button
Mixer	Bitmap, button
Sliders	Bitmap, button
DJ Object	Bitmap
Microphone	Bitmap
Stand	Bitmap

Group 4 Interactors

Table 17 USC Interactor summary

Whilst all interactive objects used to implement the USC prototype used API-specific 'forms' to project bitmaps and contain other elements, the additional functionality normally associated with the form component[31] was hidden from the user. Other redundant complexity can also be found in various dialogue boxes used by group 4; comparatively few of the interactive parts that make up these dialogues were traced back to the metaphor design. For this reason, in refining the USC model, simple interactors that only support the interactions required for the design will be specified.

---

[28] In fact, this same graphic is also visible in the DJ's room, however it is a non-interactive object.
[29] Appears as a mixing desk
[30] Scroll bars appear in this implementation - these were in fact not desired by the design team and so will not be included here.
[31] Including form decorations and implicit menus

2.5 Summary

The piece-wise examination of data collected identifies those design features that could be expressed within the ISML framework and those that could not. This comparison is based on the design features documented in the ISML data and a) the features discussed during the design meetings and b) the final USC implementations.

	Captured	Missing
Devices	Keyboard and mouse	None
Components	Bitmaps, text boxes, buttons, scroll bars	None
Metaphor	Partial features of core objects, using hierarchical or composite views. Some mappings and constraints to describe relationships between objects. Action-events affected by the DJ.	Track abstraction does not relate to media objects Some play list operations Explicit support for object re-use DJ-to-air mechanism (afferent actions missing)
Interactors	Basic equivalence of meta-objects to interactors User interactions with interactors	Detailed display and controller part definitions Interface technology specific interactor WIMP appearance and behaviours
Tasks	Hierarchical view of tasks Basic task objects A few conditions for task execution	Some task objects not included Complex concrete and abstract features of the DJ environment Communication between DJs

Table 18 ISML Design capture summary

The specification of both devices and components was relatively simple since both groups' implementations did not use complex user interface technologies.

Two distinct styles of metaphor construction emerged between teams. Group 1 chose to pursue objects and structures that were analogous to real-world counter-parts, whilst group 4 'broke' real-world concepts and synthesised them into new designs. In both cases, the specification framework was capable of expressing the basic features of each design and each group was able to specify actions and a few mappings and constraints. Problems with the abstract nature of the track object (already discussed in chapter 5) are reflected in the specification of the metaphor. A track object is a data file in reality, and this important distinction finds no place in the metaphor model. For the same reason, group 4's specification does not address the actions of file retrieval from a dialogue box that is essential for the play list operation. There is evidence for the need for an explicit re-use mechanism in ISML in the similarities between group 1's play list and advertisement book – this does not exist. Finally, although the groups identify many of the core objects, structures, mappings and action-events, there is a lack of afferent action

detail. This is an important omission, particularly with respect to inter-object communication. Group 4's air model is a clear example of this, in which a mechanism for placing of DJs to and from the air cannot be specified without further communication between the DJ, the microphone and the stand.

An enumeration of objects and interactions through the use of preliminary screen-shots provides general mappings between meta-objects and actions to interactor equivalents. Explicit details regarding mappings between specific meta-object attributes and their implementation as display parts was not elicited due to time constraints. Conversely, the details peculiar to the components used for implementation are not captured and so cannot be mapped to the metaphor model[32].

The most serious omissions are to be found in the task model, which include the absence of important real-world objects, concepts and interactions. It is clear that ISML is very weak in this area and that a rich description of a real radio broadcast environment is far beyond that which an ISML specification can express. Specifically, these problems occur in describing abstract or non-concrete concepts and communication behaviours. At present, the data collected from either group consists only of a task hierarchy, a simple enumeration of objects (and associated task actions), coupled with a handful of stop-iterate conditions.

3. The unified USC meta-object model

Having examined each group's model data, the following sections attempt to unify the core tasks addressed by both teams through the synthesis of a media, play list, mixer and air model. This will be achieved by looking to each team's design strengths and addressing, if possible, missing aspects of the meta-object model through recourse to the design meeting transcriptions.

Overall, group 1's modelling strategy took a broad and hierarchical approach, which echoes a conventional understanding of the radio broadcasting environment. However, whilst their model had greater coverage than group 4's, with respect to the number of objects included, it suffers from weak internal modelling and redundancy. Conversely, group 4 describes a more compact model that focuses primarily on the playing of tracks, play list assembly and air management. This narrowness of design sacrifices the broader perspective on the metaphor however, leaving incomplete or wholly missing supporting conceptual models; the incomplete play list meta-object and missing media player task object

[32] However, since these are largely superfluous to the metaphor, this is not a major concern.

are examples of this. Both groups were able to use concepts known to them from the real world and creatively generate a metaphor model used to support DJ activities in USC.

	Group 1	Group 4
Media model	Real-world media player hierarchy	-
Play list model	Inventory and schedule list	-
Mixer model	Mixer desk hierarchy	Mixer desk hierarchy
Air model	-	Microphone and stand
Room model		DJ model

Table 19 USC unified meta-object features

The synthesis of the USC meta-models is summarised above; in addition to the four main models, a 'room' model is added to improve completeness (this is discussed in section 3.2.5). For a complete specification of the unified meta-object model, see appendix H.

3.1 Unified task model

The limitations of the data describing tasks result in a relatively static and high-level model. Objects used within the task model are mostly derived from group 1's task data, which have a broader (but not detailed) range of objects (see Figure 62). All actions are afferent and executed by the DJ, however, no underlying state model has been specified to support their execution. A simple container-class mapping-constraint describes subordinate objects in media objects, the media player, mixer desk and play list.

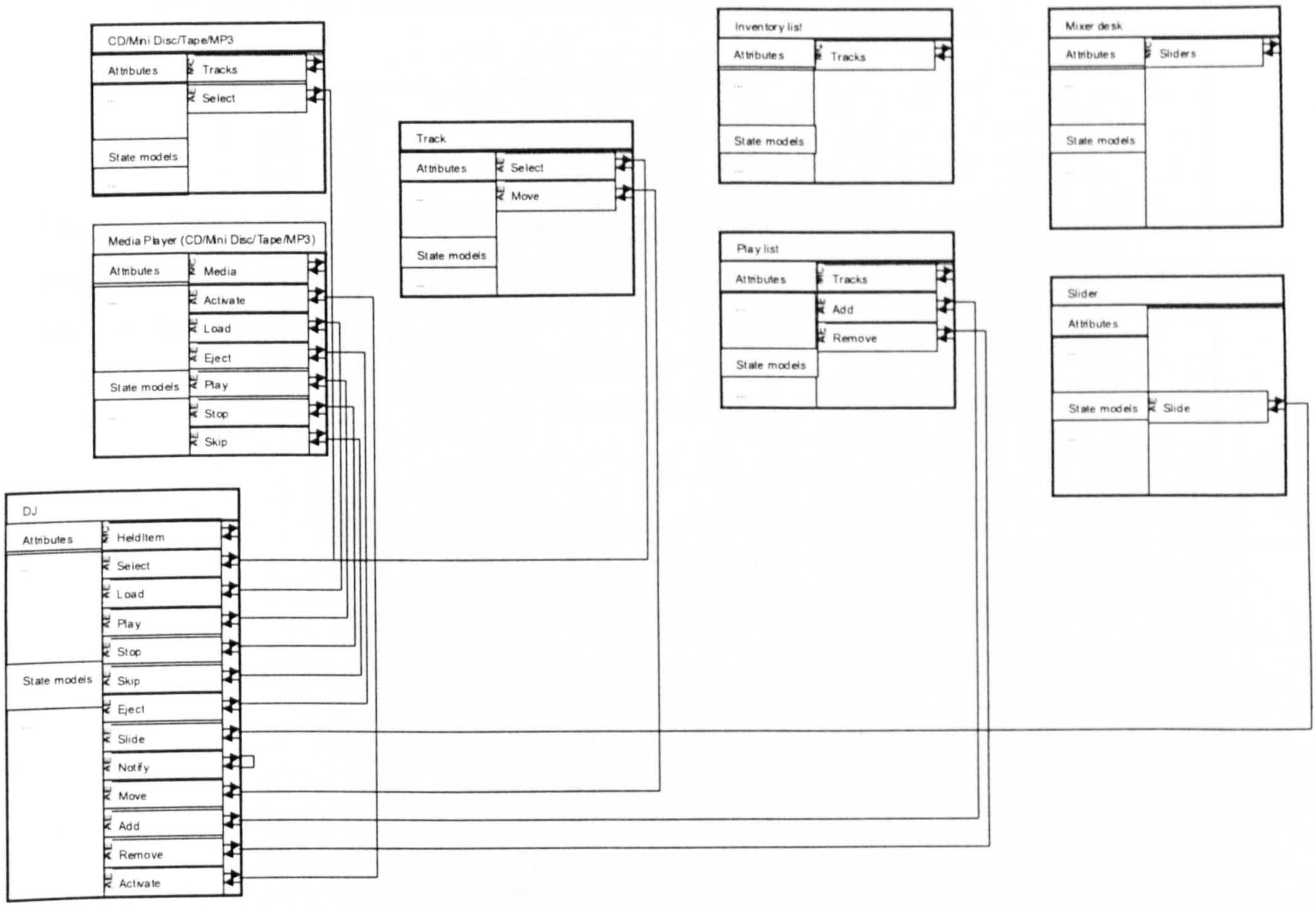


Figure 62 USC unified task meta-object model

The final task hierarchy, in Figure 63, synthesises task views from both USC design groups.

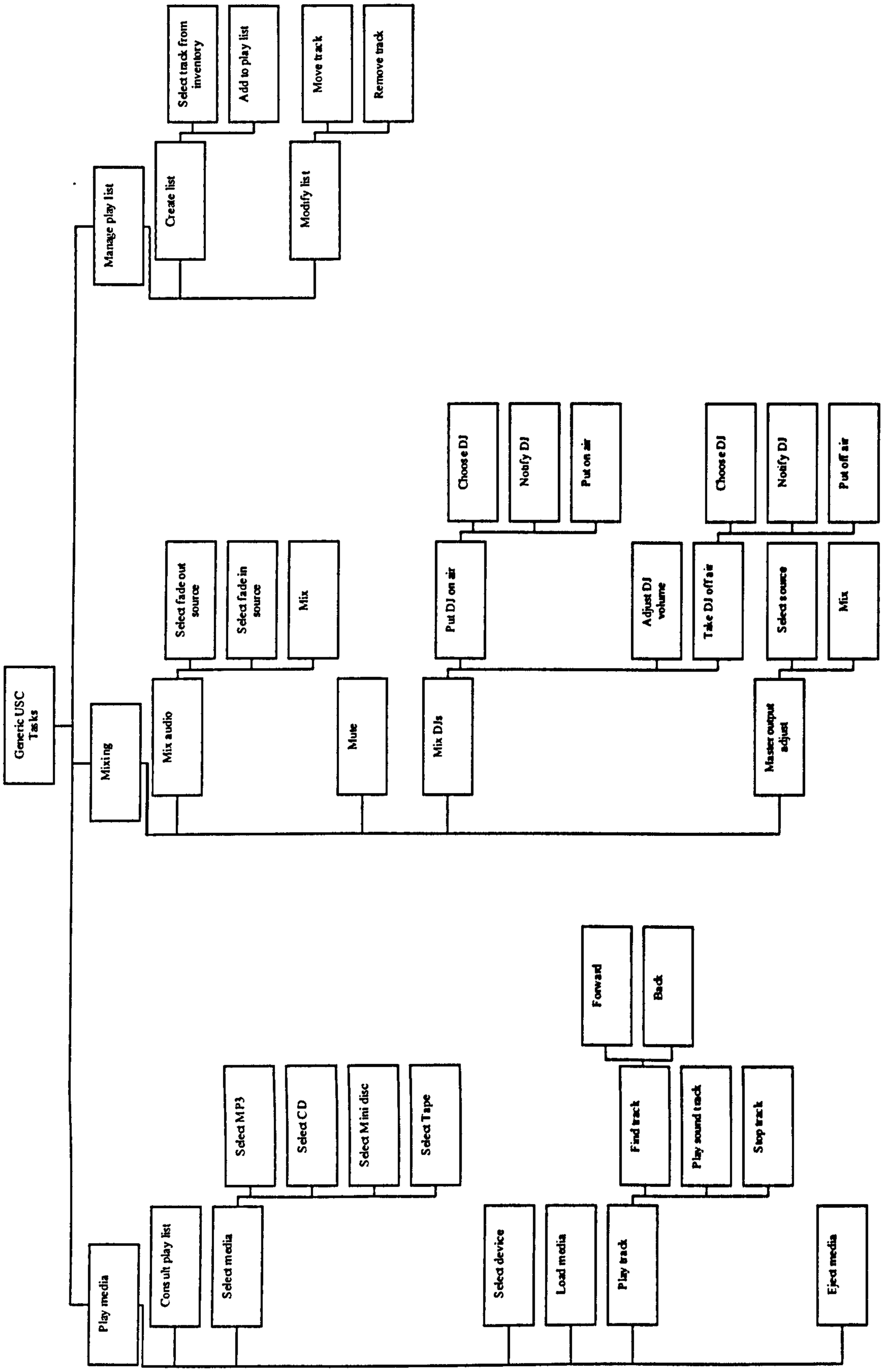


Figure 63 USC unified task model

3.2 Unified Core Meta-Objects

In this model outline, a number of simple graphical conventions are used to depict relationships between objects. What follows are a number of model 'views' in which the active meta-objects are displayed with the attributes, state models[33], mapping-constraints and action-events that are pertinent to that view. Solid triangles next to attributes indicate accessibility:

▷ = write

◁ = read

whilst arrows on MCs and AEs show afferent and efferent status:

← = efferent action

→ = afferent action

Lines connecting object boxes indicate the transmission and reception of AEs. A summary of the communication systems used by the USC meta-objects is outlined first since they support the semantic basis for the over-all design.

[33] Only the state model name is given, the complete model itself can be found in appendix H.

MC instances	MC description
MC Equipment MC Mixers MC Lists MC Sliders MC Media MC CurrentDJs MC Devices	**CONTAINER** A simple set used to maintain objects bounded with a 2D rectangle. $s_x < t_x < s_x + s_w$ $s_y < t_y < s_y + s_h$
MC Floor Space MC Current DJ MC Owner MC Environment MC Air Holder MC Current Track MC Connects MC Tracks	**RELATIONAL** A simple set. Used to maintain non-spatial relationships, no mappings or constraints.
MC Map Volume MC Map Treble MC Map Bass	**AUDIO MAP** Three mappings to specific attributes of an object (*volume*, *treble* and *bass*) $s_{volume} = (s_{scale} t_{value}) + s_{offset}$ $s_{treble} = (s_{scale} t_{value}) + s_{offset}$ $s_{bass} = (s_{scale} t_{value}) + s_{offset}$
MC Holding MC Attached	**DM (Direct Manipulation)** A set of mappings that displace the (x,y) position of the target with some source offset. $t_x = s_x + s_{offset}$ $t_y = s_y + s_{offset}$
MC ConstrainedButton	**SLIDE** A set of mappings that constrain the x position such that the target may move vertically along a constrained range. $t_x = s_x + s_{x_{offset}}$ $s_{y\,min} < t_y < s_{y\,max}$

Figure 64 USC Unified mapping-constraint summary

Five basic mapping-constraint types used in the unified model are described in

Figure 64; boxes on the left indicate specific MC instances used by the objects in the specification.

Unlike mapping-constraints, objects may only use one instance of an action-event (even though their specification name may be renamed). For this reason, the models summarised here only use the definition boxes shown in Figure 65. Action-events without a sender parameter are those events that are called during system initialisation and do not have any 'source' with respect to other model entities.

AE Pick	(SET sender, INT x,INT y)
AE PickCopy	(SET sender, INT x, INT y)
AE Drop	(SET sender, SET holding, INT x, INT y)
AE Own	(SET sender, SET toOwn)
AE Release	(SET sender, SET toRelease)
AE Select	(SET sender, INT x, INT y)
AE Load	(SET sender, SET holding, INT x, INT y)
AE Eject	(SET sender, INT x, INT y)
AE Play	(SET sender, INT x, INT y)
AE Stop	(SET sender, INT x, INT y)
AE Forward	(SET sender, INT x, INT y)
AE Back	(SET sender, INT x, INT y)
AE Slide	(SET sender, INT x, INT y)
AE SetToAir	(SET sender, SET dj)
AE AddTrack	(SET track)
AE RequestTracks	(SET sender)
AE CopyTracks	(SET sender, SET tracks)
AE *AddMixer*	*(SET mixer)*
AE TryDoor	(SET sender, STR roomType)
AE Enter Room	(SET sender, SET equipment)
AE Leave Room	(SET sender)
AE ConnectRoom	(SET room)
AE RegisterDJ	(SET sender, SET dj)
AE UnRegisterDJ	(SET sender, SET dj)

Action-Event *AC parameters*

Figure 65 USC unified action-event summary

3.2.1 UNIFIED MEDIA MODEL

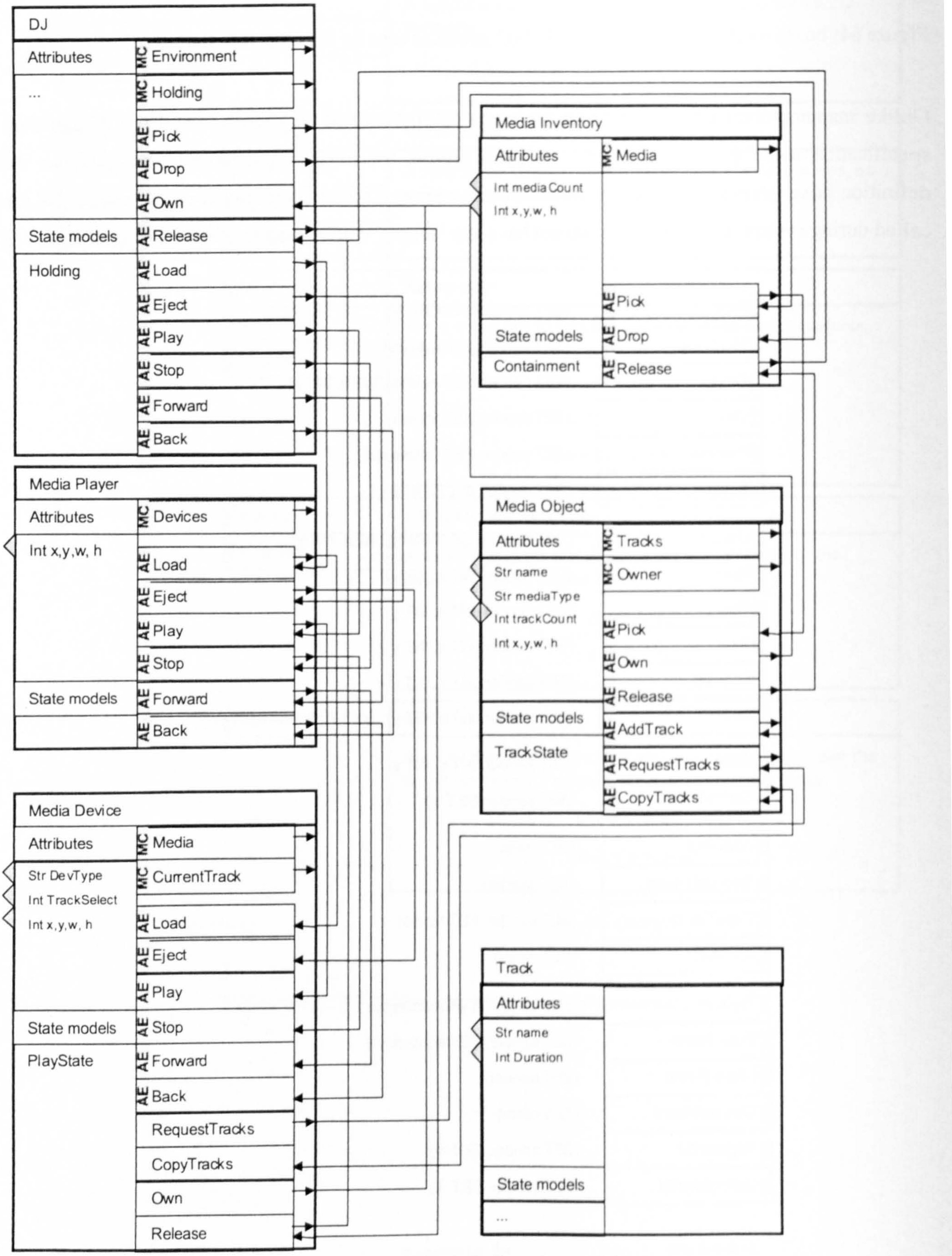


Figure 66 USC unified media model

The media model presented in Figure 66 expresses much of the hierarchical model initially described by group 1 before subsequent 'mangling' whilst at the same time adding two new entities: the media object and the media inventory object. This new inventory object should not be confused with the inventory list, which is specified in section 3.2.2. During system initialisation, tracks are added to media objects via the 'add track' action; each fully populated media object is then 'released' into the media inventory[34].

Every DJ maintains an MC environment that contains objects for his/her use – in this case a media player object and media inventory. The former contains specific player devices, whilst the latter contains media objects. During interaction, the DJ may hold a media object through the successful exchange of pick, drop and own AEs (the basic mechanism for which is described in chapter 4). Once held, the media object can be loaded into the appropriate media device (cascaded from the media player). Attempts to load a media object into an inappropriate device result in a release action, returning the object to the DJ; this release mechanism is used for eject actions. Play, stop, forward and back media operations are cascaded to the appropriate device in the same way. Upon successfully loading a media object, track objects are copied to the player device (maintained in the 'media' MC) whilst the currently selected track is contained in the 'current track' MC. All tracks are 'flushed' upon an eject action.

[34] Technical note: here, the sender is specified as 'NULL'

3.2.2 UNIFIED PLAY LIST MODEL

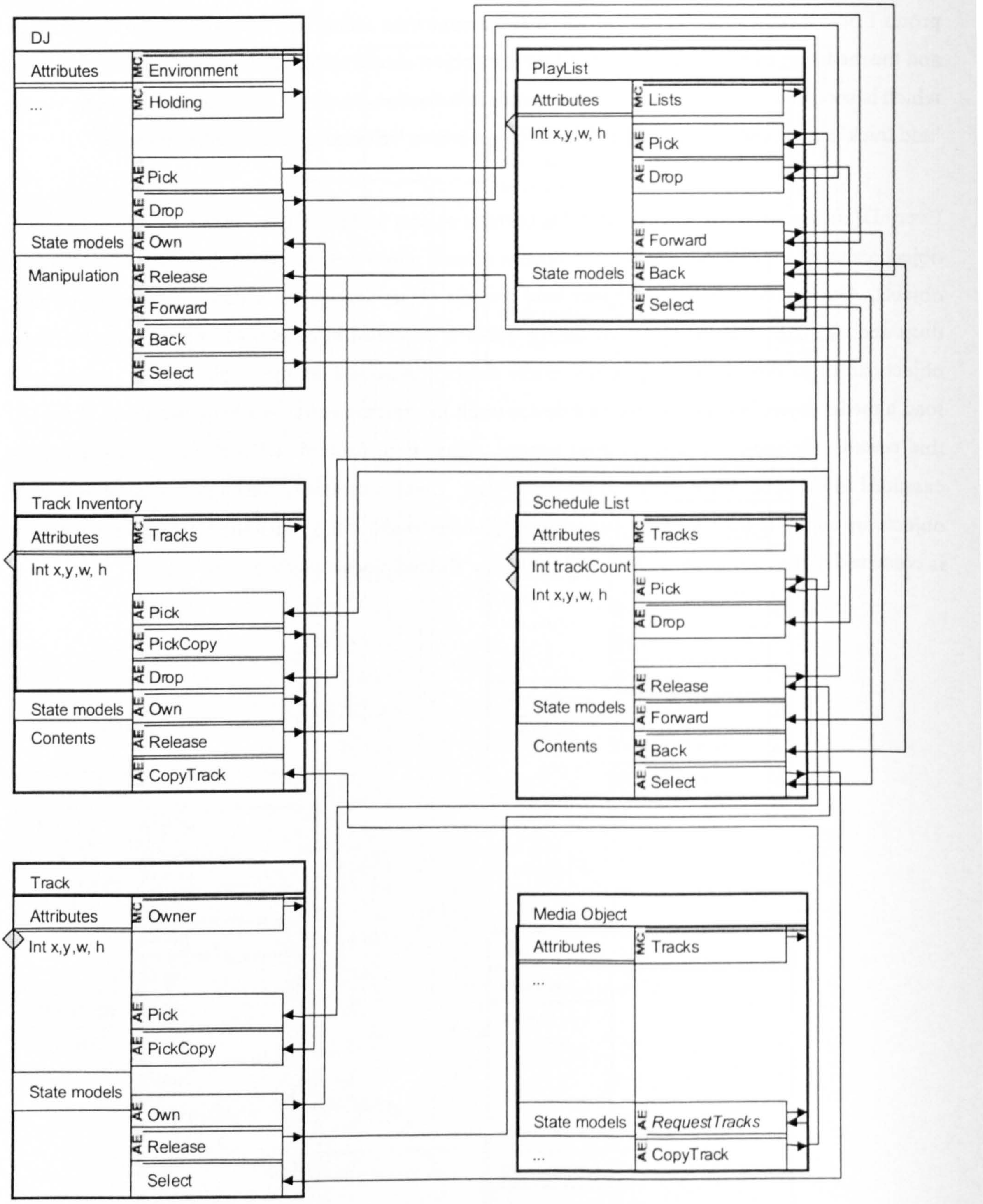


Figure 67 USC unified play list model

For both the sake of clarity and also as an acknowledgement of group 1's inventory model, the play list object is a composite of two list-like objects (the track inventory and schedule) – see Figure 67. During initialisation, all media objects copy their tracks into the track inventory. In this way, the extended media model (see section 3.2.1) can co-exist with the strong track and play list associations developed by both USC groups. The principal difference between the unified USC model and those developed by each group is that the unified model only uses the play list as a guide, rather than as a media playing device in its own right. Implementation of the latter model would only require two minor changes: 1) only one media object need be created in which all tracks reside and 2) a single media device would copy tracks from the schedule list, rather than from the loaded media object (see section 3.2.1). However, the unified model is proposed since it provides a more comprehensible metaphor.

In creating the schedule, the DJ executes a pick action that is then translated into a 'pick copy' action by the track inventory, thus ensuring that the inventory remains static. Tracks are dropped onto the schedule list using the pick-drop-own model and manipulated through a focusing action called 'select' followed by forward and backward actions.

3.2.3 UNIFIED MIXER MODEL

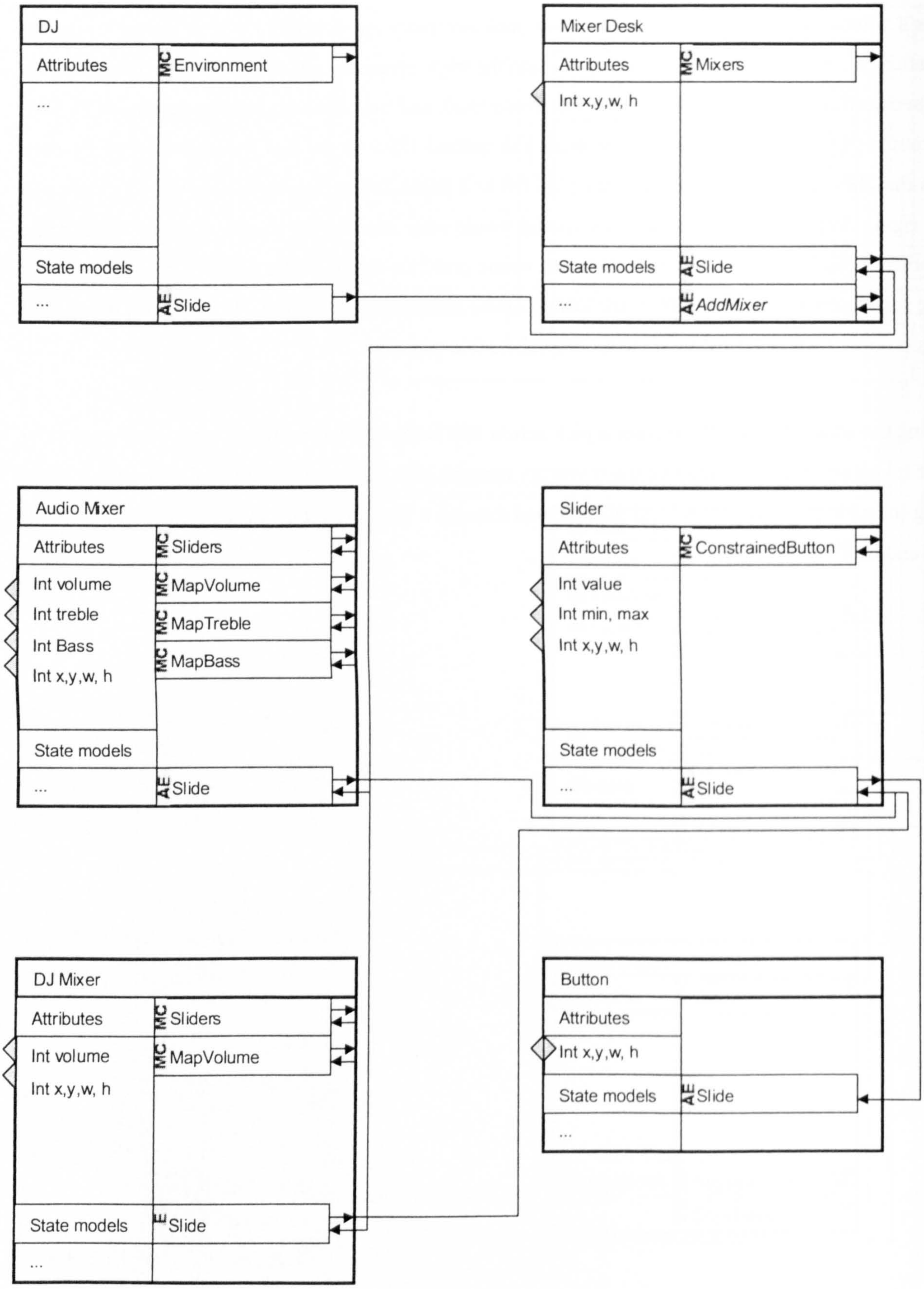


Figure 68 USC unified mixer model

Compared to other views, the mixer model is very simple, see Figure 68. System initialisation creates and hands over management of slider objects via the 'add mixer' action-event. Slide actions are cascaded through the object hierarchy (mixer desk, audio mixer or DJ mixer, slider and finally button). Values for the appropriate audio properties are mapped through mapping-constraints.

3.2.4 UNIFIED AIR MODEL

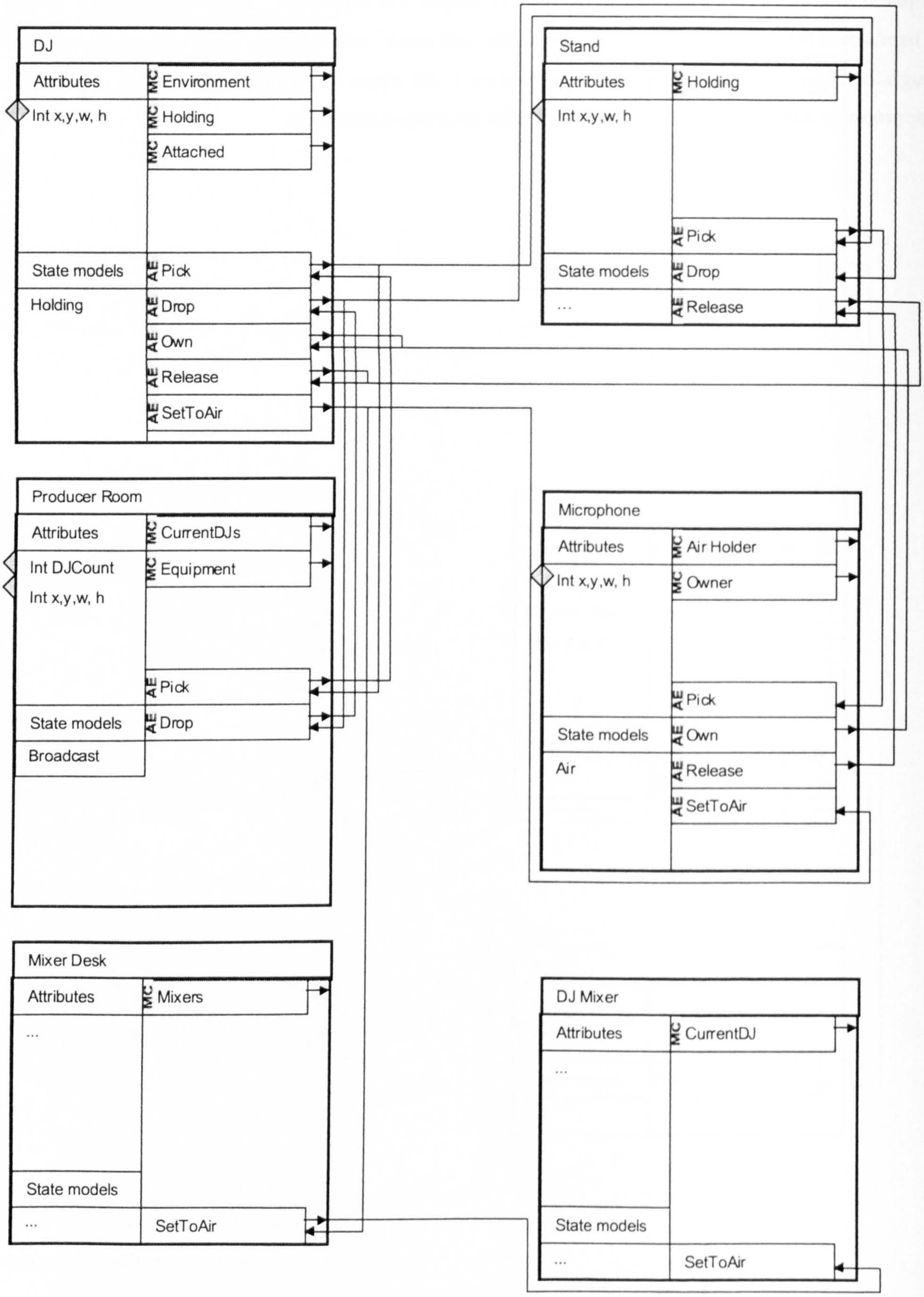


Figure 69 USC Unified Air model

The unified air model (Figure 69) appears to indicate recursion in the pick-drop-own action model, however, this is not the case but rather the result of a compact representation of both the actions of the DJs and the producer acting within the producer's environment. To understand this model, it is necessary to be reminded that a) the actions of the DJ discussed here are viewed as those carried out by the user acting as a producer and b) other DJs sharing this environment also send and receive actions (but these are not the actual actions of said user).

From the producer's point of view, the environment contains a stand and mixer desk (the microphone is initially held by the stand). The producer is located within the producer's room (via the 'connects' MC, not detailed here, see section 3.2.5) through which 'pick' and 'drop' actions may be effected to reach the currently connected DJs. Picking up and replacing the microphone from the stand are achieved using the pick-drop-own mechanism (see section 3.2.4). Placing a DJ on air means picking up the microphone and attaching it to one of the available DJ objects – these actions are passed through the 'producer room' object to the DJs contained in 'current DJs'. A DJ receiving a microphone sends a release action to the producer object, that then sends a 'set to air' action (referring to the DJ as a parameter) to the mixer desk[35], which passes on this information to the DJ mixing object.

[35] The unified model does not attach mixer desks to DJs, but rather keeps them static in the room.

3.2.5 UNIFIED ROOM MODEL

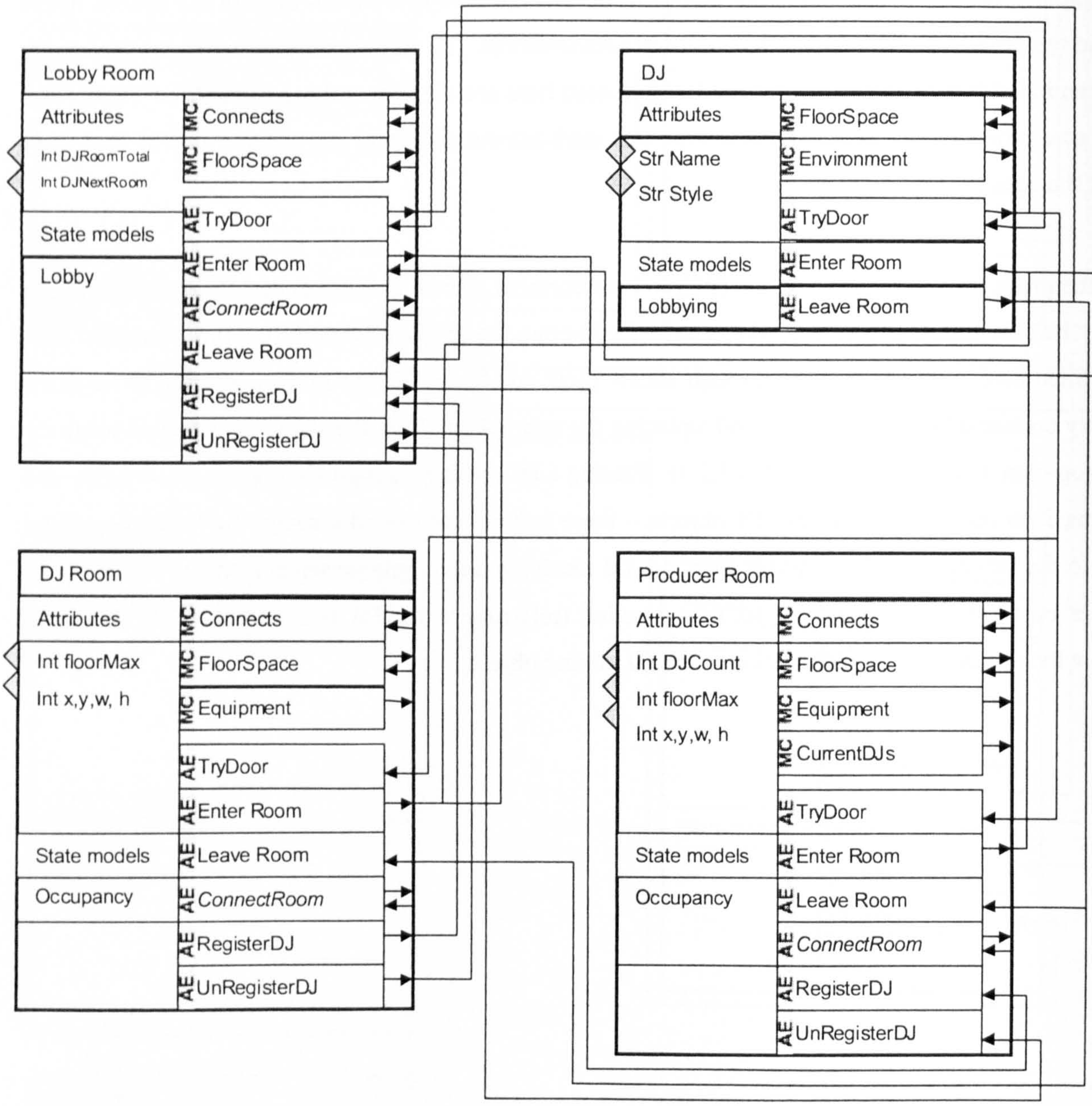


Figure 70 USC Unified room model

The notion of rooms with DJs resident is an implicit and tacit assumption evident in both USC group prototypes although this aspect of the design was not explored. However, rooms are an important part of the meta-object model since whether explicitly presented in the design (such as with group 4) or not, DJs operate within rooms that are private.

In developing the room model (Figure 70), the behaviour of both USC prototype 'log-in' systems was examined and combined with a commonly used 'lobby room' metaphor (this is explicitly modelled in

Microsoft's multi-user network layer, *DirectPlay*). At initialisation, all potential DJs and rooms are created – DJs are contained within the room's 'floor space' whilst all rooms are maintained within the 'connects' mapping-constraint. DJs may then attempt to join the shared environment by trying a door (the parameter of the 'try door' AE determines whether it is a producer's room or a DJ room). If a free door is available, the free abode sends an 'enter room' AE back to the lobby, which in turn passes it to the DJ. Additionally, if the newly occupied room is a DJ habitat, a 'register DJ' action is sent to the lobby which is then passed on to the producer room such that a list of connected DJs can be maintained. If the DJ wishes to leave any room, this action is passed to the room he/she is currently inhabiting (maintained using the 'floor space' MC) and a similar 'un-register' AE is cascaded through the system, eventually completely with the DJ returning to the lobby floor space.

3.3 Interactor layers for the unified model

In the following sections, potential mappings to interactors for each of the meta-object abstractions defined are described. In each case, all DJ actions are implemented using mouse button clicks, this is discussed further in section 3.3.6.

3.3.1 MEDIA PLAYER IMPLEMENTATION

The unified meta-object media playing model was extended to reinstate the media object and player concepts recognised by both groups, but subsequently dropped due to implementation concerns (see chapter 5). This model needs only to be partially used in either group's realisation of the media player: only one media object is required, containing all the available tracks (effectively, this represents all the MP3 files on the PC hard disk).

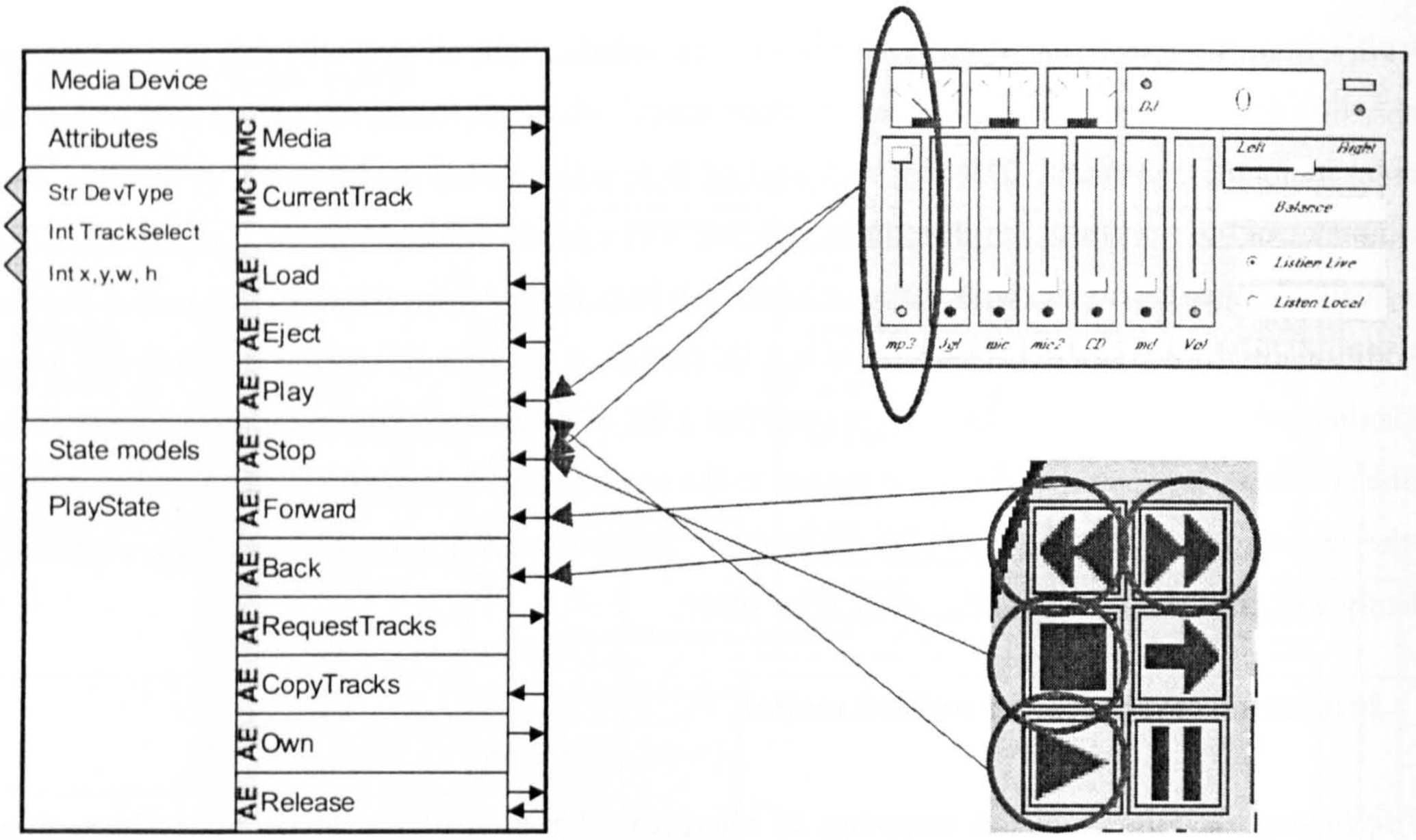


Figure 71 Media player implementations

In Figure 71, the interactor for media player is derived from the media device meta-object – all other objects from the metaphor abstraction whilst instantiated are not graphically represented in either design.

Group 1 implements their media player to appear and behave like a mixing desk - this is confusing. It is important to note that the superficial appearance and behaviour of the graphical object is *mapped to the underlying media device*. This group's interactor implements two display parts – one to display the background slider and light, and the second to display the position of the slider button. This interactor behaves as a glorified switch, calling the *play* AE when thrown up or calling the *stop* and *forward* AEs when thrown down (in accordance to prototype behaviour).

Group 4's player implements six display parts, although, in their prototype, the forward and back buttons are not functional. Here, the mapping to meta-object action is simpler: each button calls the appropriate action-event.

3.3.2 PLAY LIST IMPLEMENTATION

The realisation of the play list presents difficult challenges for the separation of metaphor from implementation for a number of reasons. Strong WIMP designs can be found in both groups' prototype designs which 'overload' the metaphor abstraction of copying a track item from a source to a target. In

addition to this, group 1 re-uses the play list model for their advertisement book, in which a direct-manipulation model that more explicitly represents these actions is applied.

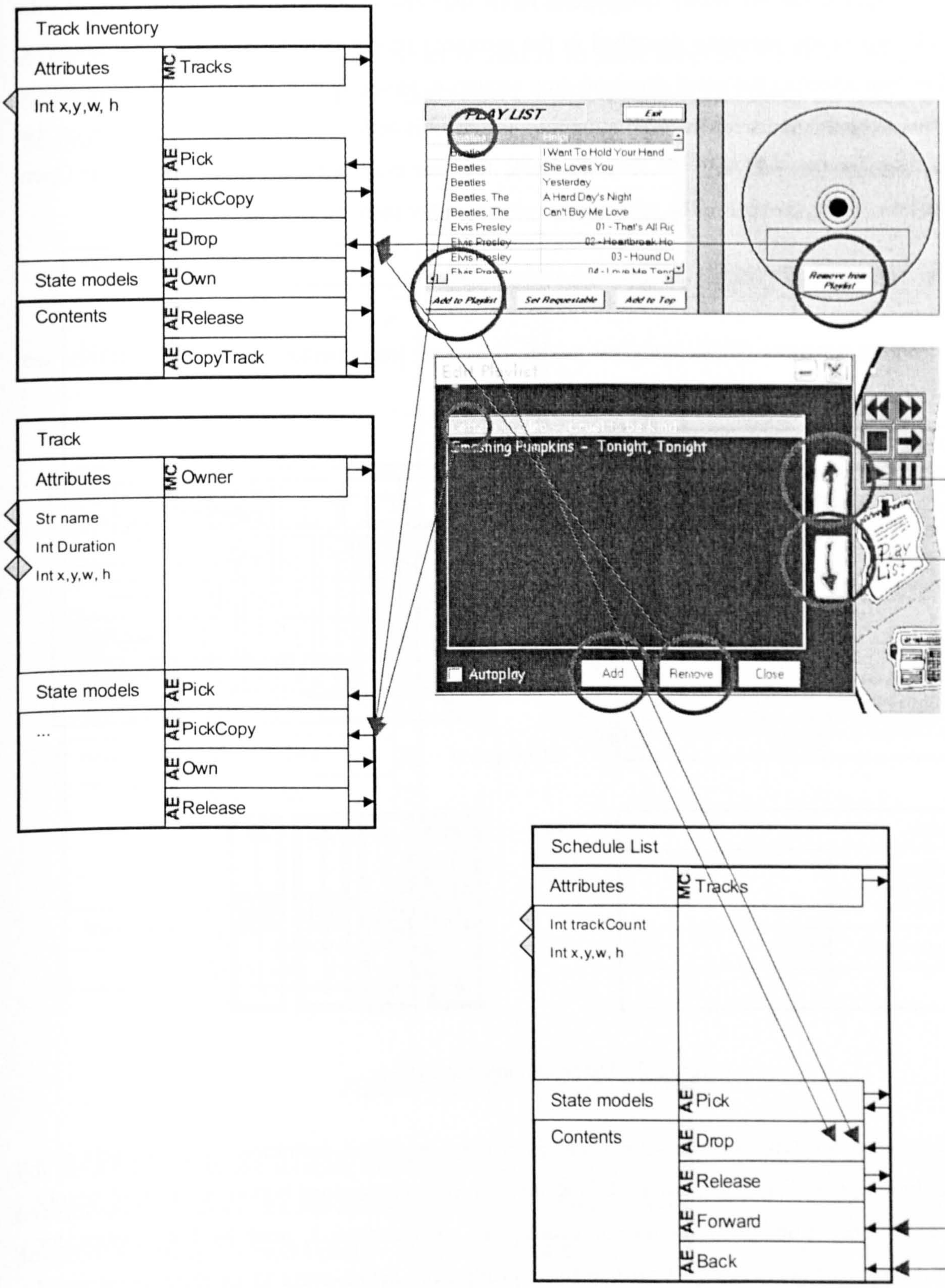


Figure 72 Play list implementations

Linking these designs to the underlying metaphor model is inelegant and problematic; the metaphor actions must be split across the WIMP components rather than enacted using direct manipulation. To do this, the pick-drop action sequence described in the metaphor model must be broken at the interactor level. Rather than affecting the usual drag-and-drop sequence: *mouse button down*, *mouse move*, *mouse button up*, two separate mouse clicks are required to perform this action. The first, a mouse click on the track object executes the *pick* AE. To *drop* the track, the user must click the Add (group 1) or Open (group 4) buttons, which are part of the schedule list object, rather than the play list.

3.3.3 MIXER IMPLEMENTATION

Of all the concrete instances of the metaphor model, the mixer implementation is the most direct, see Figure 73.

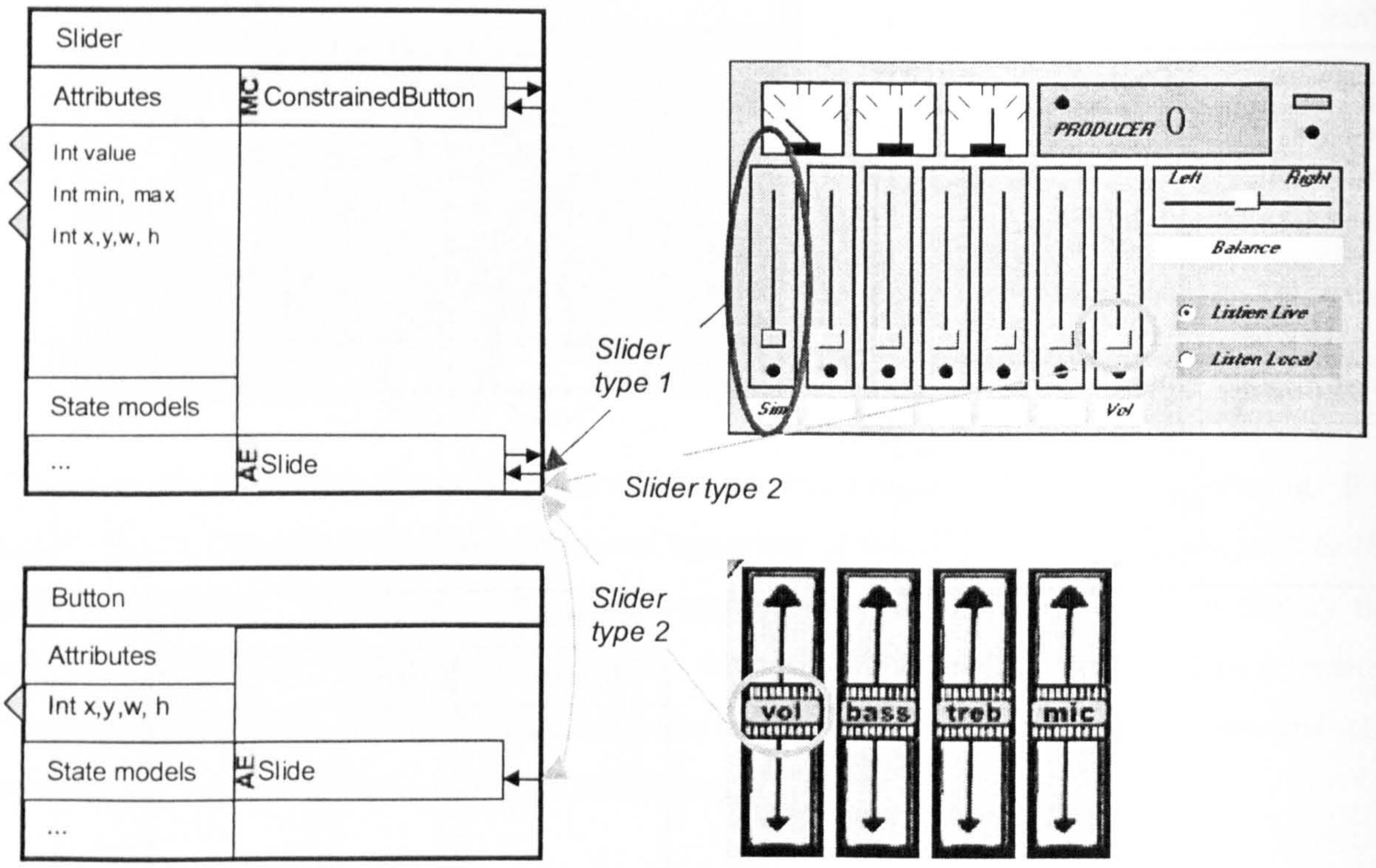


Figure 73 Mixer implementatiations

Group 1 must derive two types of interactor slider from the meta-object definition – one that acts as a switch (slider type 1) that does not forward slide actions on to its constrained button, but instead simply changes its appearance from down to up (or visa-versa). Slider type 2, used by both groups for continuous movement of audio parameters *does* pass on the *slide* action-event to its child, constrained button, which alters its position accordingly.

3.3.4 Air implementation

In this implementation, the same air model is used in its most simplistic terms for group 1, and in a sophisticated sense for group 4. The former group's design represents DJs within the producer's room as sliders on a mixing desk.

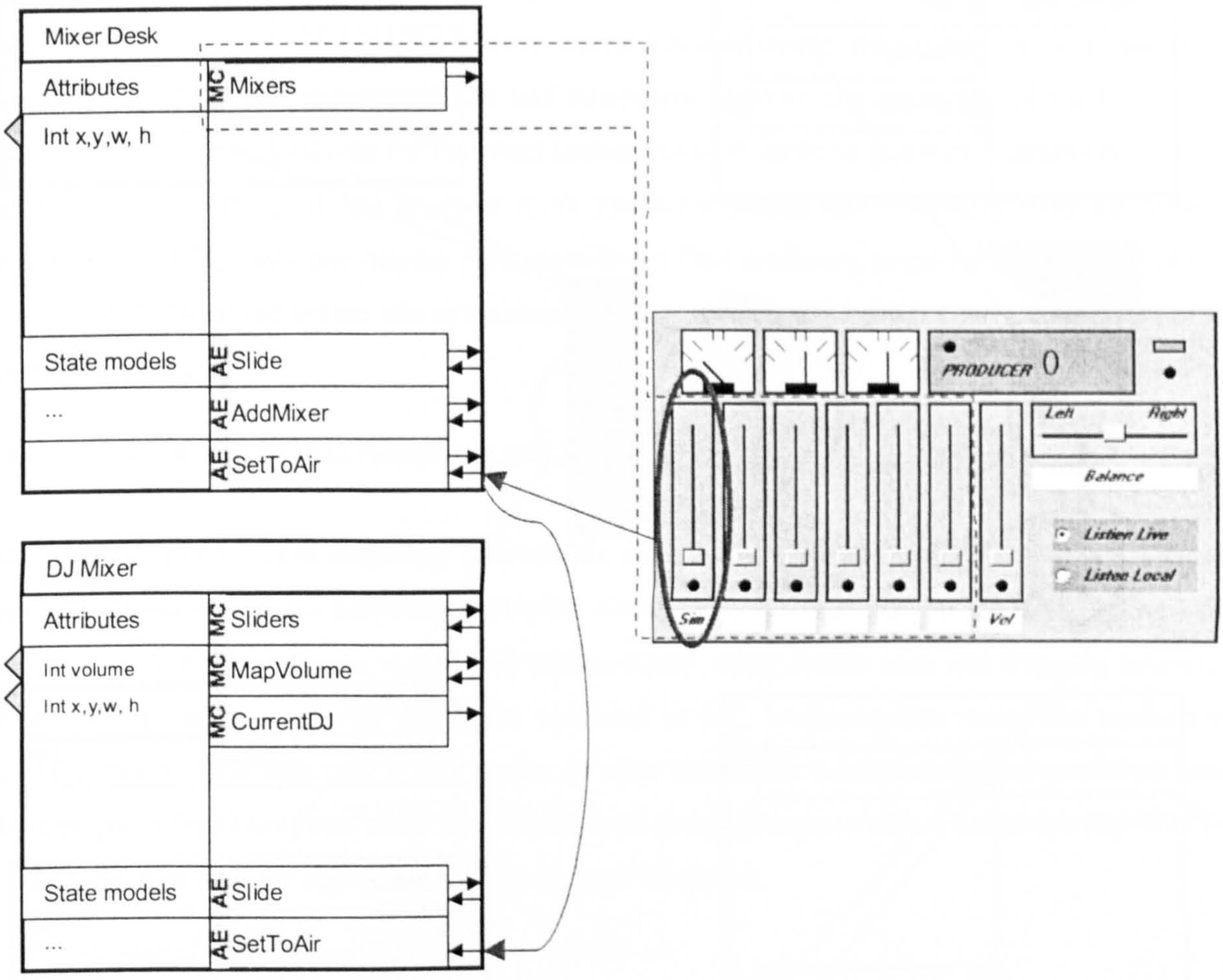


Figure 74 Group 1 Air implementation

For the group 1 interactor solution, much of the underlying air metaphor model is unused. The DJ object (*represented* as a slider, but *not* derived from a meta-object slider[36]) behaves in the same binary fashion described for group 1's media player. However rather than issuing play AEs, the set-to-air AE is called on a mouse click event, placing a reference to the current DJ object in the DJ mixer (see section 3.2.5).

---

[36] This is an important distinction; the DJ object for group 1 looks like, but does not engage in slider behaviour.

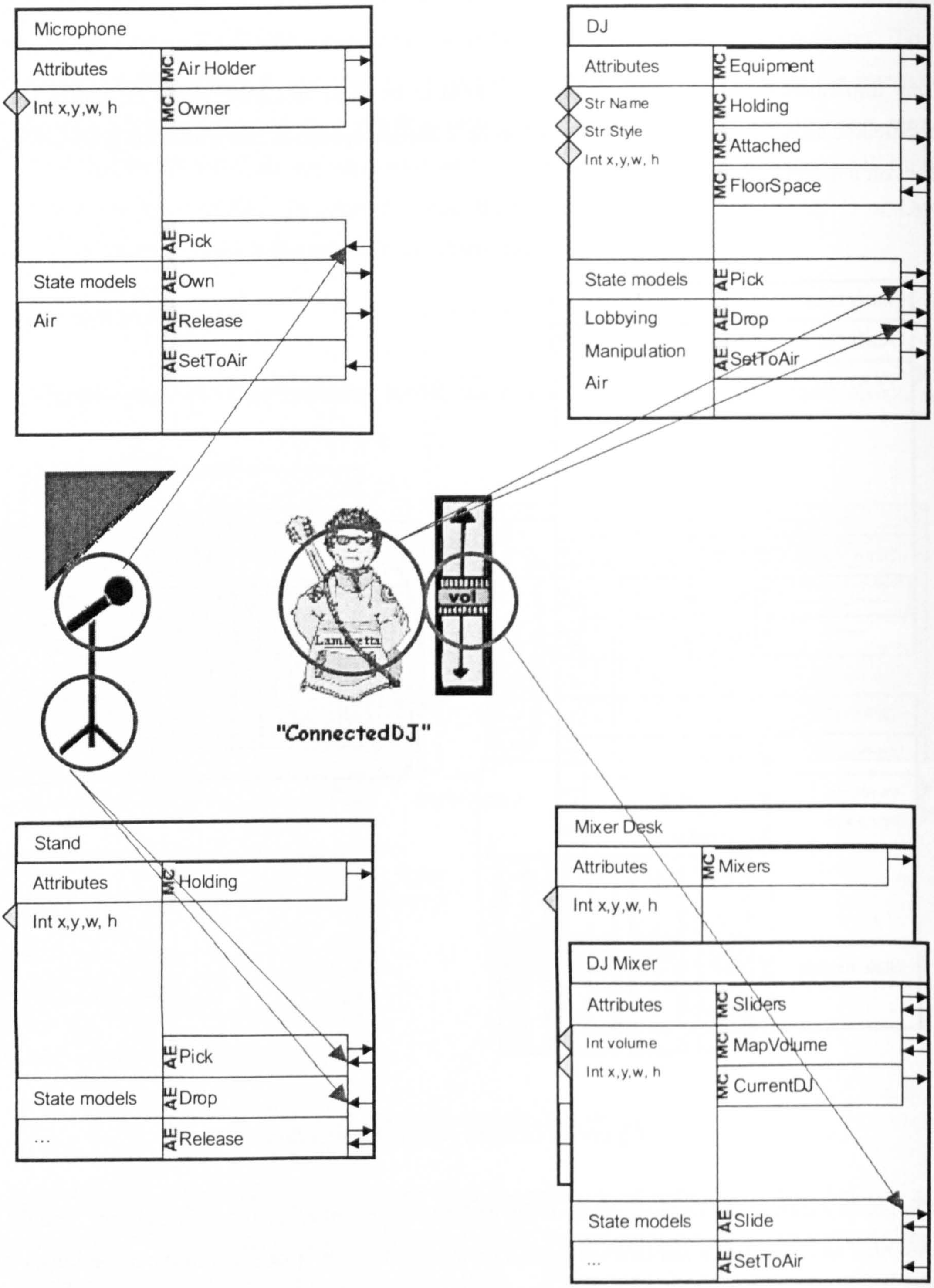


Figure 75 Group 4 air implementation

The unified air model was developed to support group 4's more interesting microphone manipulation. Here, an interactor for each meta-object is directly derived, implementing simple display parts throughout (see Figure 75, noting producer actions are shown). Mouse up and down events initiate the pick, drop and slide action-events.

3.3.5 ROOM IMPLEMENTATION

A view of the room implementation is exceptional in that it is not generated from the USC case study data. Despite this, there is a need for a room model since a) rooms are implied by both groups during discussions and b) rooms that contain DJs and equipment improve the coherency of the USC metaphor model. In fact, the principal role for the room interactors is to serve as graphical containers for the other interactive objects. Both DJ and producer room interactor objects derive directly from the meta-object abstraction. Each has only one display part, providing a final rendering target for all the other objects, so that they are all displayed within one environment (see chapter 4, section 6.5.1 for a description of the re-targeting mechanism).

3.3.6 SEMANTIC DETERMINATION PROBLEM

It has been possible to show mappings between the metaphor and interactor layers. However, a problem arises in determining which particular metaphor action is to be executed by the user, acting as a DJ. Interaction in the USC prototype is primarily mouse-based, using simple click and dragging actions. This results in the problem that, with the model specified so far, no mechanism exists that will determine whether a mouse click executes a *pick* action or *play* action, for example. To solve this problem, an additional pair of action-events must be included with every interactor object definition that will furnish the DJ abstraction with the appropriate action to send, these are:

```
GetSemantic(SET sender, INTEGER x, INTEGER y)
and
ReturnSemantic(SET sender, STR context)
```

The *GetSemantic* AE must be cascaded through the object hierarchy until the focus of a mouse click is determined, where upon the target interactor returns a string value to the DJ (the sender), detailing the appropriate associated metaphor action. Once established, the normal cascade of meta-object action-events can then proceed.

3.4 Summary

The mapping of the metaphor model to various interactor designs achieves varying degrees of success. It is clear that where interactions enact the actions of the metaphor, the translation to implementation is reasonably simple. Problems arise where components signify these actions rather than allowing the user to enact them – the play list implementation exemplifies this problem. The semantic determination problem is a 'fly-in-the-ointment' that the USC case study has brought into focus. Whilst this problem is not catastrophic to the explicit separation and treatment of a metaphor model, it does highlight that, at present, only a partial de-coupling is possible. The implication of this is that specifying metaphor abstractions using ISML seems to lead, as the USC case study has already shown, to implementation considerations where it is undesirable to do so.

4. Discussion

The findings from the examination of the model data from both groups and the application of a unified metaphor model to potential interactor designs shows that the application of the ISML framework to the USC project has experienced successes and failures in specific areas. For the specification of the unified metaphor model, interactor examples and task hierarchy, see appendix H.

4.1 Analysis question 2, part 1

Part 1 asked the question: What aspects of design did ISML capture and what was missed? Core actions performed by the user are all mouse-based actions requiring only a single button so in this respect, ISML captures the required devices easily. However, this is also a weakness in the evaluation since nothing can be said regarding problems in which other input devices are required. Specifying bitmap, button and text-box components is also relatively trivial, since they only need provide basic graphical display services to the bound display parts.

Perhaps as a consequence of the progressively narrowing focus on the design of the system as the case study progressed, many of salient features of the core metaphor model could be documented. However, two metaphor complexity problems, either a) difficulty of expression or b) difficulty in management occur for both groups' specification. The abstract nature of the track and play list illustrates this problem most clearly within the case study. Although not as serious, currently ISML does not provide any mechanism for re-use of common objects and behaviours, resulting in unnecessary additional modelling.

Many of the interactors used by each group were instances of relatively simple components (see above) that are easily implemented within an interactor abstraction. However, a small number of more complex, API-specific components would require a significant library of additional interactor designs to mimic the behaviour of the Microsoft Windows™ environment. Whilst these extra behaviours are largely irrelevant to the behaviour of the system, they do represent aspects of the final design which are missing from the specification. This raises a serious scalability issue for ISML; a large proportion of software applications run using Microsoft Windows™ components (or those like them). For ISML to become useful to the broader software engineering community, large libraries of such components would have to be created and managed – at present there is no support for this.

The task models described in this chapter, even after refinement, remain somewhat limited. Inherent complexities within the radio broadcasting domain did not translate well into a formal model: both the elicitation constraints and ISML framework are significantly under-powered to capture this information satisfactorily. Consequently, the unified task model is a static assignment of basic objects and their permissible actions towards one another coupled with a relatively inflexible, hierarchical task tree.

4.2 Analysis question 2, part 2

Part 2 asked the question: To what extent can the ISML abstract the USC metaphor? A partial separation of a metaphor model from a final implementation has been possible under the current ISML framework. It has been demonstrated that in implementations of the metaphor where there is a relatively direct correspondence of user interaction with metaphor behaviour, mappings are achievable. Play list implementations by both groups demanded that the interactor layer subvert the underlying meta-object model in order to achieve the same effect.

This finding may go some way to explain some of the design behaviours explored in chapter 5 in the following respect: each team was constrained, to a degree, by what was technically possible both in terms of the GUI components available to them and with respect to their engineering capabilities. As a result, the enactment of metaphor-level actions on objects was very difficult for them to engineer for their implementation (indeed, group 4 mentions such technical constraints; see chapter 5, section 5.2.6.2). To ameliorate this problem, each group reduced the sophistication of their design.

Finally, the semantic determination problem demonstrates that some of the properties of the metaphor model must be retained within the interactor solution. Both design groups were asked to specify their designs within the ISML framework without knowing that this would be an important problem to solve and it is perhaps for this reason that they found describing the abstract metaphor difficult

5. Summary and conclusions

This chapter has analysed the high-level ISML models generated by each group for the USC case study and proposed a unified meta-object specification that partially supports both groups' design views. Analysis of the model data from each group revealed aspects of design that ISML could capture as well as those that it could not. Specifically, the design aspects captured can be characterised as those that are visual and direct: devices, the USC metaphor objects (and some of their actions) along with their representation at the user interface. Missing design aspects were predominantly those that were either too large or complex to be easily managed, or those that were difficult to express. Examples of these include non-concrete abstractions such as the track (and its relationships), the full specification of vendor-specific graphical components, and the expression of complex interactions within the task model.

In deriving a unified model USC meta-object model, it has been possible to show that an explicit, abstract metaphor model can support two different design implementations. In attempting to separate metaphor from implementation views, two important lessons have been learned. Firstly, the WIMP-based solutions used by the groups have both led to aspects of the metaphor being 'hidden' from the user. Secondly, the emergence of the 'semantic determination' problem has provided one possible reason for the problems the groups encountered whilst trying to separate metaphor from implementation.

CHAPTER 7 Conclusions

1. Introduction

In this chapter a summary of the research carried out and its findings is presented. The research aim and associated three objectives are reviewed, followed by a brief outline of the activities that supported them at each stage. A discussion of the results of the USC case study points to a number of design issues specific to ISML and suggests possible reasons for their occurrence and the implications for the wider, MB-UID community. An evaluation of the research process and the decisions made during its execution discusses the current position of ISML research. The chapter concludes with directions for further work and the contribution this work has made to model-based, user interface design research.

2. Summary of research

This research has developed a novel specification framework to support metaphor models that can be integrated with other user interface design views. At present, little work exists that makes an abstract metaphor design explict and integrated with other model-based design notations or tools. Toward achieving this aim, three research objectives were identified:

1. Identify extant HCI design models that might be extended to support metaphor abstractions

2. Develop a language that supports metaphor abstractions and integrates with models found in (1)

3. Evaluate the language developed by (2) with user interface designers/software engineers to assess the application of an abstracted metaphor layer to the design of a GUI prototype

To identify potentially useful models for objective 1, a review of the theoretical design frameworks and model-based views on user interface design was presented. This contrasted the perspectives on development from a number of different methodologies found within the HCI research community. It was also important to gain some understanding of the technical foundations that supported these various design perspectives. A review of the literature revealed varying degrees of formal and tool-based support, but very little that could be said to directly support metaphor abstractions. However, the range of design views and architectural abstractions found in the literature had plenty to offer for extension to support of metaphor design.

The ISML framework was then developed based on a limited range of model-based abstractions and computational concepts; this limitation was necessary since the range of design views addressed by the MB-UID community is broad. After the framework had been realised at a high level, it was considered important to capture these concepts in a formal language so that it might lend itself to some form of machine support. The first inception of ISML, based on a *Lex* and *Yacc* grammar Levine et al. (1992) was abandoned due to the excessive code generation required to parse the language. Subsequent migration to XML proved more successful since tools already existed to automatically verify and validate the language.

A small-scale model was developed using ISML to make some initial explorations with the language; this revealed numerous syntactic aberrations (subsequently corrected) as well as some insights into the use of the framework to specify metaphor. A larger case study was then conducted to determine, in a more realistic development scenario, how the framework might be used to specify novel, metaphorical, graphical user interfaces. It was considered important to gather evidence for or against its application to a real design problem; this would help validate (or not) the concept of a metaphor abstraction and the mappings to other design views as prescribed in ISML. For this reason a qualitative approach was selected, as it seemed likely to produce the richest data set for this question (for a discussion on selection of methodology, see chapter 5, section 2).

The qualitative analysis focused on the elicitation of an ISML specification, interactively constructed by the author and two, independent design groups. In fact, design meetings with each group were all transcribed – however the length of the complete transcript set was too large to analyse within the scope of the research time scale. A grounded-theory Glaser and Strauss (1967) approach was adopted since this was the most flexible and generative method for analysis in the face of novel social behaviour (i.e., software engineers specifying designs using a new specification framework). Findings from the analysis revealed a number of interesting design behaviours and responses from the teams that could be compared with respect to the description of tasks, the metaphor model and the interactor implementation. This prompted further analysis of the specification data produced by both groups to identify a) which aspects of the design ISML had captured (or missed) and b) to what extent the metaphor abstraction could be separated from other user interface design views.

3. Summary of findings

3.1 Objective 1

Carefully designed metaphors in user interface design are widely recognised as useful tools to help users interact with a computer system. An initial review of the literature indicated that whilst interface designers could look to psychological or formal accounts of metaphor, the effective use of these accounts would require special training and so would not be easily accessible to many development teams. Guidelines and case studies on the application of metaphor to UID potentially invest designers with useful insights, but relatively little support can be found to explicitly specify and map metaphorical concepts to other design views.

In order to raise the profile of metaphor design and relate it to other interface design considerations, a review of the broader HCI literature then took place. Since this is a broad and inter-disciplinary area, it was important to execute a focused and directed examination. To this end, it was decided that an examination of some of the common notations and models within the HCI literature would be a good approach for two reasons. The first, to identify potential abstractions to both extend or relate a metaphor model and secondly, it would place this research within the context of the broader research community without getting lost in numerous theories and methodologies.

The literature review outlined some of the design views that enjoy some form of model-based support, including task, presentation, dialogue and domain oriented frameworks. It became clear that within this field, a number of approaches to development and abstraction were shared between design views. The concepts of communicating objects, abstracting small aspects of a larger design view was prevalent in much of the literature. If an explicit metaphor model were to be useful, it would be necessary to map it to existing formalisms, and it was on this basis that ISML was created.

3.2 Objective 2

The design views and architectural developments evaluated in chapters 2 and 3 led to the adoption of a small selection of concepts and computational models that could be extended to support a metaphor abstraction. Specifically, these were event-based communications and mappings between objects that maintain 'abstract' and 'concrete' parts (variously applied in the literature) synthesised to address five design views within the ISML framework:

1. Devices (simple, high-level abstractions of input-output hardware)

2. Components (collections of input 'collectors' and output presentation units)

3. Meta-objects (includes syntactic and semantic definitions for a metaphor model)

4. Interactors (implementation of (3), realised with mappings to (2))

5. Tasks (simple, hierarchical task descriptions with mappings to (4) and subsequently (3))

A syntax and grammar for the ISML framework was expressed using XML. The small-scale specification example given in chapter 4 highlights the features of the language and also some of its inherent limitations, leading to predictions of scalability and metaphor separation problems for larger scale projects.

3.2 Objective 3

The six-month USC case study was executed with two, independent software engineering teams of four members each. Some high-level structure was needed to aid the analysis of ISML, and to this end, two principal analysis questions were raised:

1. What are the reactions of developers to the use of ISML?

2. To what extent does ISML capture a design?

Data gathered for question one was a sub-set of the transcription of the seven design meetings that took place over the course of the case study. An analysis of the transcripts resulted in comparable accounts of

how both teams specified their USC prototype designs using the ISML framework. Subsequent post-project open-ended questions generated group reflections that show their perceptions of the utility and practicality of the language. The ISML specifications generated by both teams were compared and contrasted to evaluate the extent to which it reflected the USC design problem. A unified metaphor model was then derived from within the scope of the concepts shared by both teams and potential mappings to each team's implementation demonstrated.

3.2.1 QUESTION 1 : WHAT ARE THE REACTIONS OF DEVELOPERS TO THE USE OF ISML?

Analysis of question one is further sub-divided into two parts:

Part 1: How is a user interface metaphor developed within the ISML framework?

Part 2: What is the perceived utility and practicality of the application of ISML to design?

Open coding of the design meetings was followed by a structured axial coding method, examining in turn each of the 3 elicitation phases (task, meta-object and interactor) conducted during the case study. An examination of the treatment of each of the phases highlighted the effects that the ISML framework had on design expression (part 1). Group reflections on the specification process shed some light on the use of the ISML framework during the case study (part 2).

3.2.1.1 The development of user interface metaphors within the ISML framework

Analysis of the case study design meeting transcriptions revealed a number of design behaviours:

- Design reduction
- Non-concrete concepts
- Implementation bias
- Metaphor mangling
- Common models and re-use

The progressive reduction in design complexity and the 'mangling of metaphor' that occurred as both groups migrated from task descriptions to actual interactor solutions suggested that both groups undertook strategies to manage the complexity and mappings between design views. In each case, there are examples of a reduction (or elimination) of objects and their roles in order that they can be managed in the implemented design.

Few software engineers would dispute that most software engineering projects will undergo some degree of scope limitation of the problem in order to make a solution tenable Jackson (2001). Indeed, the relatively short life span of the USC project (6 months) may partially account for this behaviour. Where the complexity of the metaphor model became unmanageable, at least one of two strategies was taken: either a) a return to conventional WIMP based solutions and/or b) the mangling of metaphor. The strongest evidence for the former behaviour can be seen in the play list design and the reflections of group 4 explicitly state the complexity of the problem as their reasons for this recourse to conventional design. Both groups also use iconic representation: metaphors are substituted for symbols to indicate, but not enact, metaphorical concepts.

Metaphor mangling was an unexpected artefact of the mapping problem, resulting in both distorted and serendipitous designs. Distortions are plainly evident in group 1's prototype; here the expected behaviour of one object was substituted for another. Ironically, whilst group 1 exhibited strong analogies to real-world structures in their metaphor model which is subsequently mangled, group 4 did not adopt such a strong analogical approach but managed to produce a more coherent implementation. This shows that a de-coupled metaphor model does not necessarily lead to an interface design that reflects it; the potential reasons for this are discussed in sections 3.2.2.2.

Further descriptive problems emerge as 'non-concrete concepts' – aspects of the environment which are properly part of the domain but cannot be easily expressed using concrete metaphors. Each group found it difficult to describe abstract properties of a metaphorical environment (such as the track or the scheduling properties of the play list or show) within the ISML framework. Part of this problem was almost certainly due to the novelty of ISML for the USC development teams. However, in developing metaphors both groups sought, to varying degrees, to adopt objects and behaviours from the real world: these do not always map to the features of the intended system. Alty and Knott (1999) refer to this as a 'S+M-V+' condition or an instance of *metaphor inconsistency*. This is a problem for the ISML framework: conceptual abstracts appear to play at least as important part in the metaphor model as do the existence of real world objects. It is possible that yet another form of abstraction is needed here to fill this gap. Furtado et al. (2001) introduce an ontology-based model in which concepts, relations and attributes are mapped to task, user and domain models. A higher-level of abstraction such as this might be a suitable augmentation to ISML, expanding its expressive capability.

3.2.1.2 The perceived utility and practicality of the application of ISML to design

The reflections from both groups (conducted independently) were elicited using five open-ended questions intended to allow each group to offer their views on the use of ISML as a generative framework for discussion and also as a practical or useful tool for design. Responses from the groups reiterate some

of the problems experienced during development (outlined in part 1) and suggest that the ISML design process is one that has expression limitations, but that can also reveal important features in the design of metaphor.

Upon reflecting on issues such as the impact on their design and the practicality of the application of ISML, the groups made three main points. Firstly, both groups suggested that their 'kernel' ideas would remain more or less the same but that the ISML process would help to 'unlock' design ideas and reduce repetition in the description of object behaviour. Stepping through the ISML specification process was beneficial in an analytical sense and at the same time, this realisation also points to a potential improvement in the ISML framework through the introduction of a more object-oriented approach to object specification.

Secondly, it was difficult for the groups to separate the metaphor abstraction from implementation concerns, which made ISML model building difficult – this is discussed further in section 3.2.2.2. The USC groups' reflections on this matter also suggest improvements in the specification elicitation method could be made by more clearly delimiting the scope of the metaphor model and expanding the expressive power of the framework. Whilst the former might be addressed by improving the teams' understanding of the ISML philosophy through extra training, the latter is a harder problem. Some of the metaphorical relationships derived from the elicitation (such as 'containment' and relational mapping-constraints) are very similar to *image schemas* identified by Lakoff (1992) in his psychological account of metaphor. Benyon and Imaz (1999) argue that many aspects of HCI design to some degree utilise image-schemas to describe ideas – the appliance of these psychological structures to the ISML elicitation process may improve communicability of ideas.

Finally, both groups felt that the time required producing an ISML specification was substantial and that the benefits from doing it would be maximised if it were conducted at the very beginning of the project. This is not surprising since it is inevitable that the decisions made by each group earlier on in the project with respect to the functional requirements of the system clearly have an effect on aspects of the interface design. The challenge of a networked application, capable of playing media and designed with a view to streaming audio[37] in later revisions was not trivial. As such, the technical challenge of the project almost certainly conflicted with the metaphorical designs developed by the teams.

---

[37] Actual audio streaming was not required for the prototype. Nevertheless, group 4 did in fact implement this feature.

3.2.2 QUESTION 2 : TO WHAT EXTENT DOES ISML CAPTURE A DESIGN?

Subsequent to the USC qualitative analysis, analysis of question 2 is also sub-divided into two parts:

Part 1: What aspects of design did ISML capture and what was missed?

Part 2: To what extent can the ISML abstract the USC metaphor?

A critical examination of both group's design models summarised the design features identified within the ISML framework during elicitation and highlighted those aspects which are either not adequately addressed or missing (part 1). An analysis of the devices and components used by each group identified the interactor requirements for the specification (part 1). Finally, a unified USC model was outlined, identifying those aspects of the metaphor model that can be specified independently of any one group's implementation (part 2).

3.2.2.1 Aspects of USC design captured and missed

An analysis of the models from both groups reflects the findings from the qualitative analysis. Much of the richness of the task model is not adequately expressed using current ISML methods. The missing features fall under either objects not considered important in the task model by the design group or highly abstract concepts that are not easily expressed within the current framework. Based on the data from the evaluation, it is clear that task modelling within ISML is under-powered and not fully exploited. If ISML is to be a viable framework for GUI development, this must be addressed since without an adequate task description the value of the model is substantially diminished.

Meta-object models fared better; ISML was able to support hierarchical and composite views of design for both groups, both of which were able to make use of attributes, state models and action-events. However, some of the action-event models were under-specified since only efferent actions were documented. This points to a review of the elicitation process and the need to include steps to capture and verify the communication mechanisms between meta-objects; indeed, some tool-based support for this process would be highly desirable since it could potentially diagnose problems such as these.

The eventual components used by each group to implement their design challenged the ISML framework in two important ways. The first challenge is that of scalability – this was predicted from the small-scale desktop model specified earlier. Whilst many of the components used by the groups were relatively simple to model at the abstract, interactor level, the potential number of 'concrete' instances of the object

was very large simply because of the many software APIs available. This represents a scalability problem for ISML since it demands an explicit rendering mechanism, in a similar fashion to the AUI model Schneider and Cordy (2001). At present, the alternative is to delegate the rendering details to component abstractions, such as in Luyten and Coninx (2001) and Mueller et al. (2001). Whilst this alternative removes the scalability problem, it also reduces the expressive power of ISML by implicitly restricting the range, appearance and behaviour of components to those dictated by the implementation target. There is clearly a need for some kind of library abstraction here, in which sets of ISML components can be independently developed and pooled.

A corollary of the first, the second challenge is that of metaphor complexity and scalability. Both teams were forced to fall back to the ubiquitous desktop metaphor in some aspects of the implementation. More complex objects (such as file dialogues) introduced the need to specify yet further models to support concepts from the desktop and office. Again, ISML does not provide support for multiple metaphors (or indeed recognise them as such) and so there is a clear need for management of concepts in this regard. Again, some form of software-based support to manage the complexity of this problem is required – an ISML elicitation and specification toolkit beckons.

3.2.2.2 The extent that ISML abstracts the USC metaphor

The unified model necessarily introduced additional design features (such as the DJ booth) to create a coherent and self-contained model. Mapping the metaphor abstraction to the core implementation features of each group was troublesome. Difficulties arose when the interactor-based solution did not allow the user to enact actions upon objects contained within the metaphor. Media player and mixer solutions resolve to relatively simple mappings, since their mappings are direct. However, where some objects and actions were only partially visible or completely hidden in the implemented system, mapping problems occurred. These resulted in an interactor model circumventing the underlying action and event sequences in order to achieve the desired effect in the underlying metaphor model.

A further problem arises in linking the metaphor abstraction to the implementation layer, referred to as 'semantic determination'. The USC case study revealed this hitherto unrecognised problem because it includes a wider range of action and objects within the scope of the specification. At present, it is necessary to include additional logic within the interactor layer in order to determine the 'meaning' of a user action as it is directed toward an object represented at the interface.

These findings suggests two possible reason for some of the design behaviours exhibited by both groups:

1. Implementation constraints reduce metaphor 'visibility'

2. The current mappings between ISML meta-objects and interactors are problematic

The groups' choice of GUI component technology resulted in the hiding of some aspects of the underlying metaphor – objects and actions were implicitly enacted 'behind the scenes'. In other words, the capabilities of the target hardware and software can be seen to affect the expressiveness of the design; this problem is also identified in the construction of virtual environments Smith et al. (2000). Target platforms that implement very constrained forms of interaction, such as the command line or menu system, are likely to both poorly represent an underlying metaphor and also enact the actions associated with it on behalf of the user.

The 'semantic determination' problem (see chapter 6, section 3.3.6) could also be one of the reasons for the groups' difficulty in making distinctions between the metaphor abstraction and an interactor solution – the expression of a variety of actions are enacted only using the mouse. It is therefore not surprising that group members often talked about their design in implementation terms.

Whilst these problems were solvable, this has shown that it is not entirely possible to separate implementation details from the metaphor abstraction. Of course, there may be situations in which this might be desirable (such as the use of batch information processing systems). Batch or automated systems not withstanding, in situations where a user is unfamiliar with the operation of a system the implementation of even the most effective underlying metaphor is likely to be of limited use since many of the concepts are hidden from the user. Given this, it is not unreasonable to predict that ISML will probably be more effective when used with technologies that provide greater degrees of freedom with respect to user input and output technologies.

Finally, the experiences gained from the use of ISML offer lessons to the wider MB-UID community. ISML is at an early stage of development – other model-based enterprises are considerably more advanced. The case study in this research has shown that the complexity of the ISML framework causes problems not only due to scalability issues but also because the mappings between different design views are not always obvious or simple. ISML does not guarantee good metaphor design - simply being able to

describe a large problem space is not enough. A deeper understanding of the relationships between different design views, in a *real user interface design context,* is also important.

4. Evaluation of research process

In this section an evaluation of this research is presented, outlining a number of the challenges that faced its execution and the impact that they made on the eventual direction and position of the research. The broad and inter-disciplinary nature of HCI presented this research with the problem of delimiting the scope of HCI design methods and tools within which to search for a suitable platform to develop an explicit metaphor model. Important design views such as psychological and organisational perspectives were not included in ISML. Even within the scope of the model-based design methods limitations had to be specified in an attempt to make the research tenable. Evidence from the ISML evaluation has highlighted the problems that result as a consequence of these absences. Without these necessary limitations however, it is likely that the mapping problems already discussed above would have been exacerbated still further. Identification and management of mappings between design views is a large and complex challenge that faces the MB-UID community Puerta and Eisenstein (1999), Vanderdonckt and Berquin (1999) and it is clear that ISML is no exception to this problem. The case study has shown that the mappings between the metaphor and interaction abstraction are difficult and not properly understood by the design groups.

Similar decisions were made with respect to the architectures and technologies used to support ISML. The tension between the high levels of abstraction and the practicality of a software-based toolkit proved difficult to resolve for a number of reasons. On the one hand, a highly abstract formalism offers the possibility of analysing metaphor models using mathematical methods, such as in Kuhn and Frank (1991). However, as already stated, this is a hard exercise and difficult to communicate to designers. On the other hand, forging ahead with the development of a complete software tool to support the ISML framework would require a substantial engineering effort without proof-of-concept. Arguably, without a proof-of-concept, a lot of design and engineering effort could have been wasted if, when eventually released, developers found the concepts it embodied to be either difficult to work with, or worse, useless.

Therefore a middle ground was sought in the form of a specification language that encapsulated the ISML framework. Originally, an extension of the C syntax to include notations for meta-objects, interactors and tasks was developed. Although attractive because the initial grammar was relatively elegant and already included logical and functional expressions (see appendix O) this was later abandoned because of the

considerable development overhead required to produce a multi-stage parser. The subsequent XML alternative proved to be more successful because, although more verbose and less elegant[38], a number of parsers and partial validation tools (such as Altova's XML Spy) already exist to support the specification process.

Writing an ISML specification using the XML formalism has a number of advantages and disadvantages. Positive features of this approach include some automatic verification and validation of the specification and the potential for later transformation into other logical forms using the XSLT processor. These translations may be either to other model-based specifications or to potential source code. Negative aspects include very long specification documents that are laborious and difficult to read. In addition to this, the logic of the model must be checked manually and it is for this reason that an executable tool is highly desirable.

The evaluation of ISML therefore had to be suitable for the Ph.D. research, i.e., an examination of the proof-of-concept of the framework. For this reason, a modestly sized case study with a qualitative analysis method was chosen in order to assess how the design of a strongly metaphor-orientated project might be captured using ISML. Working with two teams of software engineering degree students presented many difficulties and confounding effects on the development of the USC prototypes. Both teams had severe limitations upon their time both to the project itself and ISML project meetings. Ideally, such a case study might better be conducted in a context where the designers were only focused on the USC system and had more time during which to develop it. The realities of practical research dictated otherwise, although it is arguable that there are few engineering projects where pressure on development time and effort isn't going to be a problem.

Results from the evaluation yielded important insights into the treatment of an explicit metaphor model and its relationships with other views of user interface design. The richness of the data collected in the transcriptions was both the evaluation's saving grace and Achilles' heel. Identifying those aspects of the design that ISML did and did not capture was possible with recourse to the transcripts, followed by a comparison to each group's models. However, due to both the complexity of the problem and the length of the transcripts covering the six-month project, only a limited data set could be realistically chosen from within which to examine the ideas expressed by each group. It can be argued, therefore, that different data and conclusions might have been drawn from the case study had the focus been elsewhere. Whilst this argument affords some credence, ultimately the research had to concentrate on data from within the

---

[38] The mark-up text makes XML verbose and difficult to read

ISML elicitation rather than casting a broader gaze over the more general features of a metaphorical user interface prototype.

Overall, this research has been lead by necessary limitations and trade-offs in order to gain some insights into one possible way of explicitly specifying the operation of a metaphor model within a graphical user interface design. Along the way design and evaluation decisions that rejected some approaches have had noticeable effects on the outcome of this research. This is not wholly unexpected and, arguably, it is likely that if other trade-offs had been made other problems would have occurred as result of different omissions. As such, the position of this research is by no means a panacea, but instead a useful starting point from which to address the issues associated with the ISML design framework and grounding for further research.

5. Further work

In many respects, the interface specification meta-language is still in its infancy. This research has highlighted a number of problems not only in the assembly of the framework itself, but also in its reception with USC design groups. Potential revisions to the language are many. The provision of re-use through class inheritance for devices, components, meta-objects and interactors might well reduce specification size. A re-working of the action-event mechanism such that calls return values and include caller identification would also simplify design. More importantly, the provision of some degree of data abstraction to support non-concrete aspects of the metaphor model and re-work of the framework to solve the semantic determination problem would seem essential.

As a model-based user interface specification language, ISML is relatively broad and as a result, some of the 'periphery' elements are not well developed, particularly task modelling. This is a disappointing but inevitable limitation of the scope of this work. The task layer in ISML is basic, expressively weak and has no predictive power. As such, it can only really be used as a 'documentary' device for linking idealised task forms and their execution at the interface (through an interface metaphor). However, the ISML task framework does reflect a number of important properties found in task modelling literature (hierarchy, actions, objects, states and constraints) and so may well be amenable to extension and refinement.

For both of the USC design groups, separating the metaphor model from its implementation was not easy. During the ISML elicitation, a number of important aspects of the metaphor were clarified when the group discussed the visual aspects of their prototype system. Additionally, an ISML specification expressed in XML is both very large and takes a long time to write. It would therefore be desirable to provide tool-based support for ISML. At present, an ISML run-time kernel is under construction (the

meta-object kernel has already been partially constructed, see appendix I) that will support ISML designs. In addition to this, a tool kit that automatically translates ISML into compilable code, based on XSLT transformations, is also planned (see appendix J for a sample of work in progress).

6. Contribution to knowledge

Currently, the MB-UID community is challenged with the integration of a large number of design views for the development of interactive systems for a wide range of computing platforms. Not surprisingly, this is an enormous undertaking and one that is likely only to be accomplished in small, incremental steps by the community as a whole. This research sought to develop an explicit metaphor model that could be integrated into model-based user interface methods. The ISML framework, its application in the USC case study and subsequent evaluation has shown one particular approach to the problem and demonstrated some success in capturing metaphorical aspects of a novel user interface design. In addition, this research has also uncovered important lessons with respect to the effect that the separation of metaphor from implementation issues can have on user interface design and its wider implications for the model-based, user interface design community.

Appendices

Appendix A – The USC project proposal

Appendix B – ISML elicitation programme

Appendix C – Open coding frequency chart

Appendix D – Axial coding charts

Appendix E – USC Analysis data

CD-ROM Contents

Appendix F – The XML expression of ISML

Appendix G – A small ISML specification

Appendix H – Unified USC specification

Appendix I – Basic meta-object kernel (in development)

Appendix J – Sample XSLT transformation

Appendix K – USC Transcripts

Appendix L – USC model summaries

Appendix M – Group 1 USC Prototype

Appendix N – Group 4 USC Prototype

Appendix O – Early ISML grammar

Appendix P – Atlas.ti data files

BLANK IN ORIGINAL

SEM Final Year Group Project 2001-2002

Company: Media Cool Inc.

Product: Urban Shout Cast

Customers: Simon Crowle, Jim Craven

Monday, 24 September 2001
PRODUCT BRIEF

Broadband Internet connectivity will soon be an affordable reality for many households in the UK. Combined with powerful, multimedia home PCs, there will soon exist an unprecedented opportunity for individuals to broadcast their own media shows to a wider Internet community. Media Cool is a multimedia innovation company that seeks to develop and license powerful, *easy-to-use* multimedia broadcast solutions.

Our first step is to design a virtual radio broadcasting room and develop a prototype that can be used to test proof-of-concept. If successful, this design will be used as the basis for the development of the product proper by our own in-house software engineers. Basic broadcasting and streaming technologies already exist (WinAmp/ShoutCast; Microsoft NetShow technologies) but Media Cool envisages much more:

- Innovative and intuitive GUI for server and client
- Multiple, remotely linked DJs sharing the virtual broadcast environment
- Virtual mixing desk
- Advertisement management
- Client music request service

We expect the prototype to run on current, entry-level, Intel-compatible computers running Microsoft Windows 9x, ME, 2000 or XP over an ADSL/Cable or other LAN Internet connection. Developers of the prototype will work closely with Media Cool's own software lead (Simon Crowle) during the requirements, specification and design stages. The successful prototype development team will be imaginative and innovative, producing a prototype that will appeal to the anticipated early adopters of this technology.

BLANK IN ORIGINAL

Appendix B – ISML elicitation programme

Introduction
Thanks
This is not a test of any kind!
Not a design assessment exercise
Do ask questions or add comments at any stage during the meeting

TASK

Real world task description
Based on a hierarchical task model (logical breaking down of tasks)
May include 'plan' parts
Stop description when we get to an action-object reference

HTA
Create task hierarchy; top down

Iteration condition
Per task, identify any iteration condition
If one exists, list condition **[create MC]**

ENUMERATE TASK ACTIONS
Create task actions (referring from HTA)
Per action, link actions with HTA
List actions

ENUMERATE TASK OBJECTS
Create task objects (referring from HTA)
Per object, verify efferent task actions
Per object, link with HTA
List objects

MC Verification
Per object, verify MCs
Per MC, validate conditional parts (attributes, states) against objects
List MCs

Products:
Action List
Object List
MC List
Hierarchical task model
Iteration stop conditions (mapping-constraints)
Task objects (summary form only)
Task actions (summary form only)

META-OBJECT

Abstract description of the virtual/metaphorical environment
Looking for descriptions of the objects, their behaviours and actions
Not looking for physical descriptions of interactions (like mouse clicks)

HIGH-LEVEL OBJECT ENUMERATION

List all objects
Per object, create brief object description **[create Object]**
Per object, suggest potential metaphor links with task HTA

EFFERENT ACTION ENUMERATION

Per object, specify all efferent actions
Per efferent action, **[create Action]**
Per object/action, specify any focus on sub-ordinate object parts
Per object/action, specify source of action
Per object/action, specify any consequences **[create MC]**
Per object/action, add MC conditional parts to object description

ACTION VERIFICATION

List all actions
Per action, verify sources and targets
Per action, suggest potential metaphor links with task HTA
Consequences of action
Per action, consider consequences for source **[create MC]**
Per action, verify consequences for all targets

OBJECT ENUMERATION PART II

Per object, verify attributes/states
Per object, verify afferent/efferent actions
Sub-ordinate relationships
Create brief object description **[create Object]**
Per sub-object, specify relationship ('is contained by' etc) with super-object **[create MC]**
Per sub-object, specify focused efferent actions from super-object
Per sub-object, specify consequences of efferent actions **[create MC]**

MC CLEAN-UP

List all MCs

Products:
MO Object list
MO Action list
MO MC list
MO Object descriptions
MO Action descriptions
MO Mapping-constraint descriptions

INTERACTOR

This section describes the implementation descriptions of the virtual/metaphor environment
Don't worry if this is incomplete!

INPUT/OUTPUT DEVICES

Informal device enumeration
Per input device, describe the number of data inputs and their types
Per output device, describe data output types and (informal) rendering capabilities
List devices

HIGH-LEVEL INTERACTOR OBJECT ENUMERATION

Per MO object, **[create super interactor]**
List interactors

HIGH-LEVEL INTERACTOR ACTION ENUMERATION

Per MO action, specify input device 'actions' **[create device action description]**
List input device action descriptions

SUPER INTERACTOR DIFFERENTIATION

Efferent action focusing
Per input device action description, identify sub-interactor focal parts **[create sub-interactor]**
Link sub-interactors to their super-interactor
Add sub-interactor to interactor list

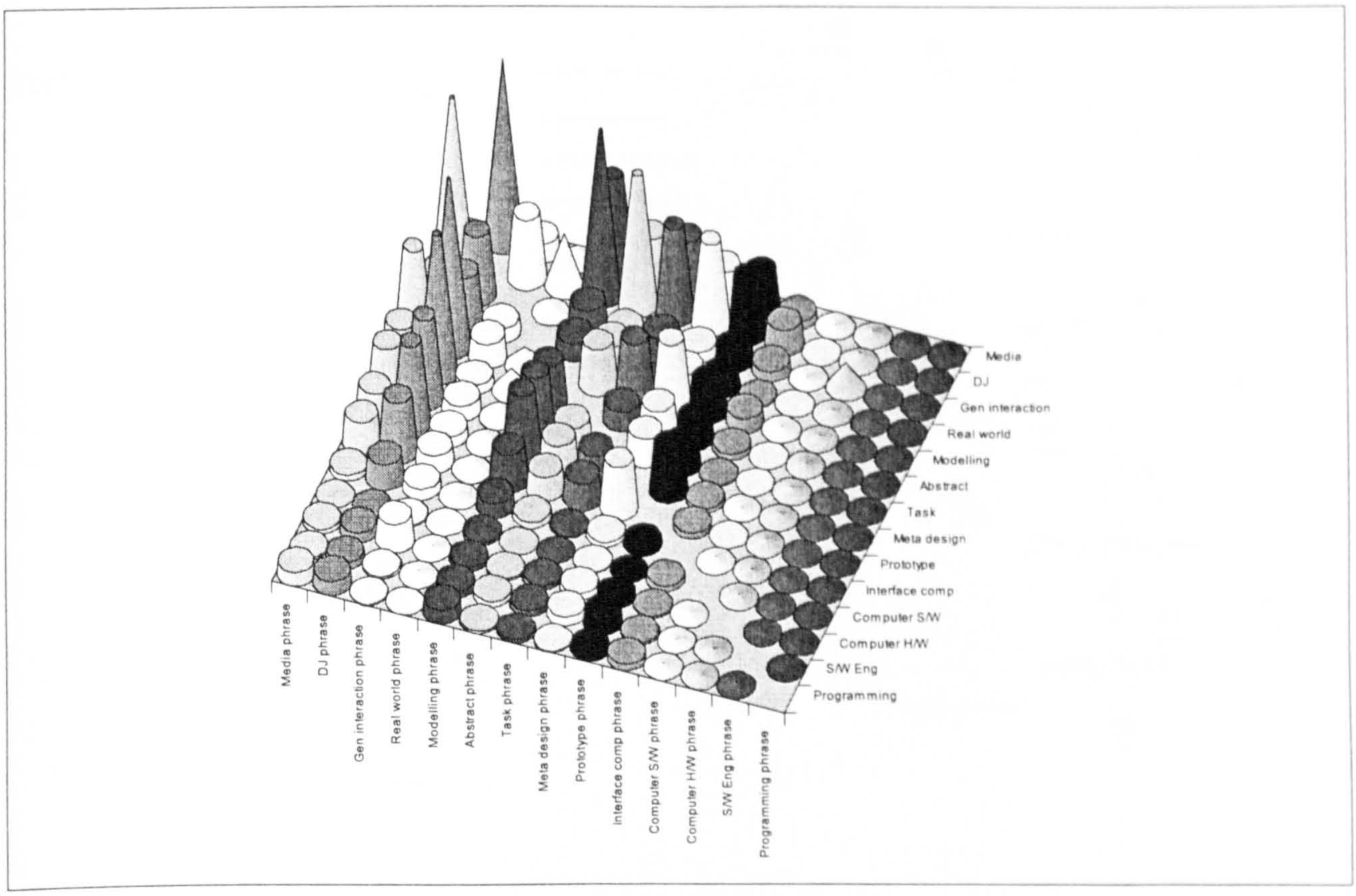
Products:
Device List
Interactor list
Device action list
Device descriptions
Interactor descriptions
Device action descriptions

DE-BRIEFING

Many thanks
Honest feedback
Questionnaire

BLANK IN ORIGINAL

Appendix C – Open coding frequency chart



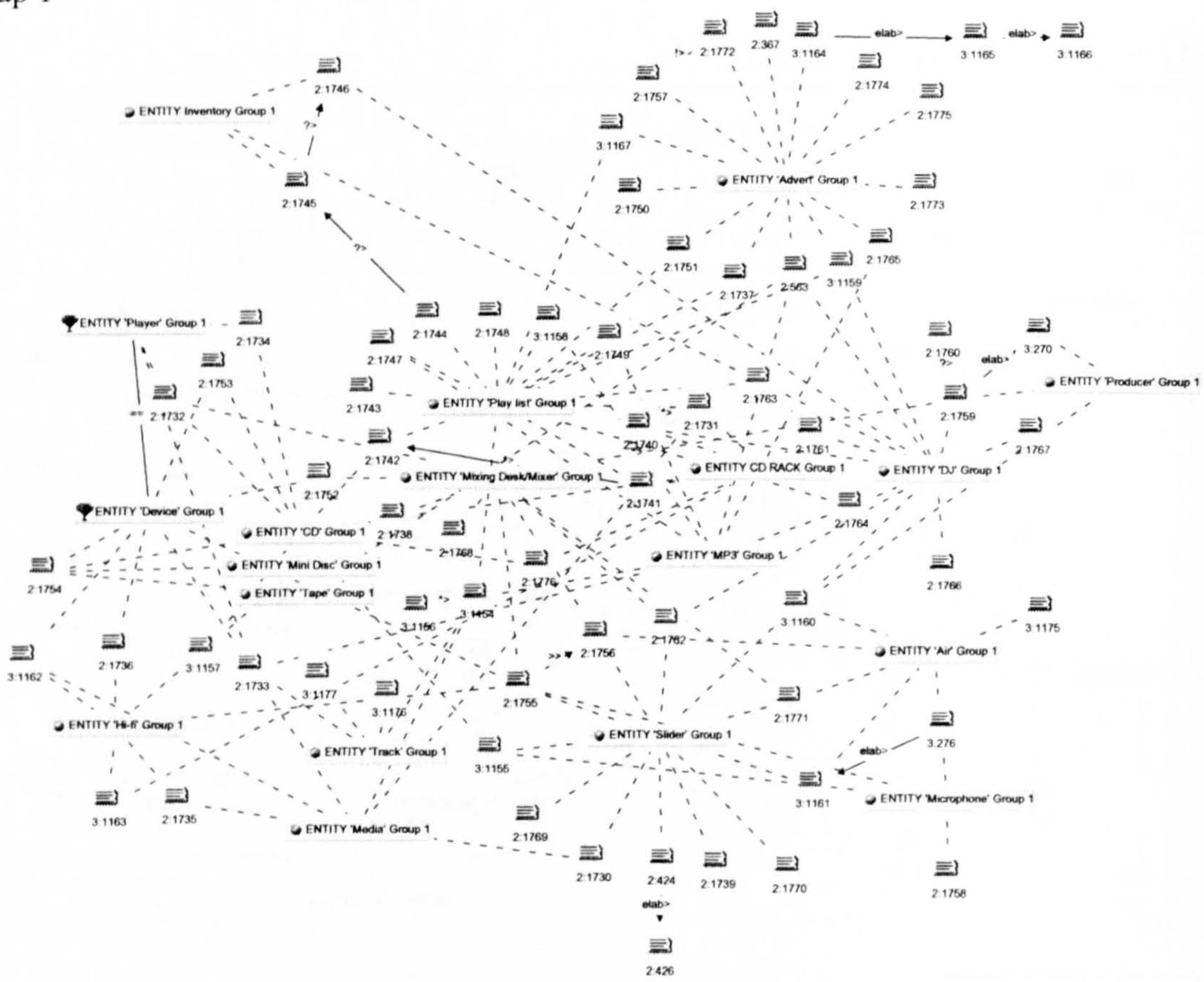
The graph above indicates co-occurrences of noun groups with noun phrases, and visa-versa. In the following quotation (8:1019):

> "But the **track** itself, no, it's not. When you play the **track**, that's just reading some data from a file, you know, it's not the actual WAV file itself, so no, it's not."

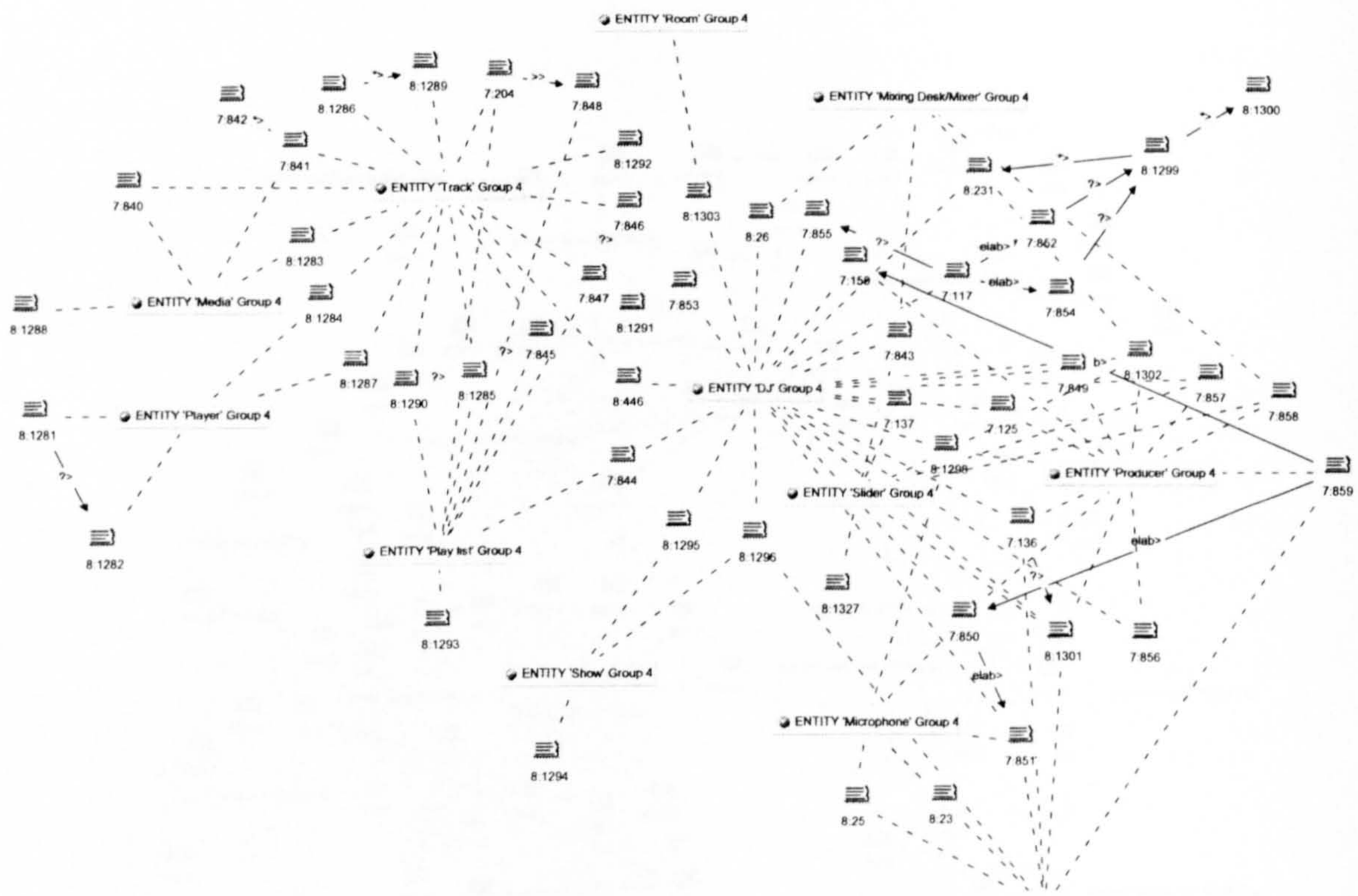
There are three *programming nouns* (underlined) and two *media nouns* (emphasised). The quotation is classed as a *media phrase*, since it primarily discusses a media concept.

PAGE NUMBERING AS ORIGINAL

Group 1



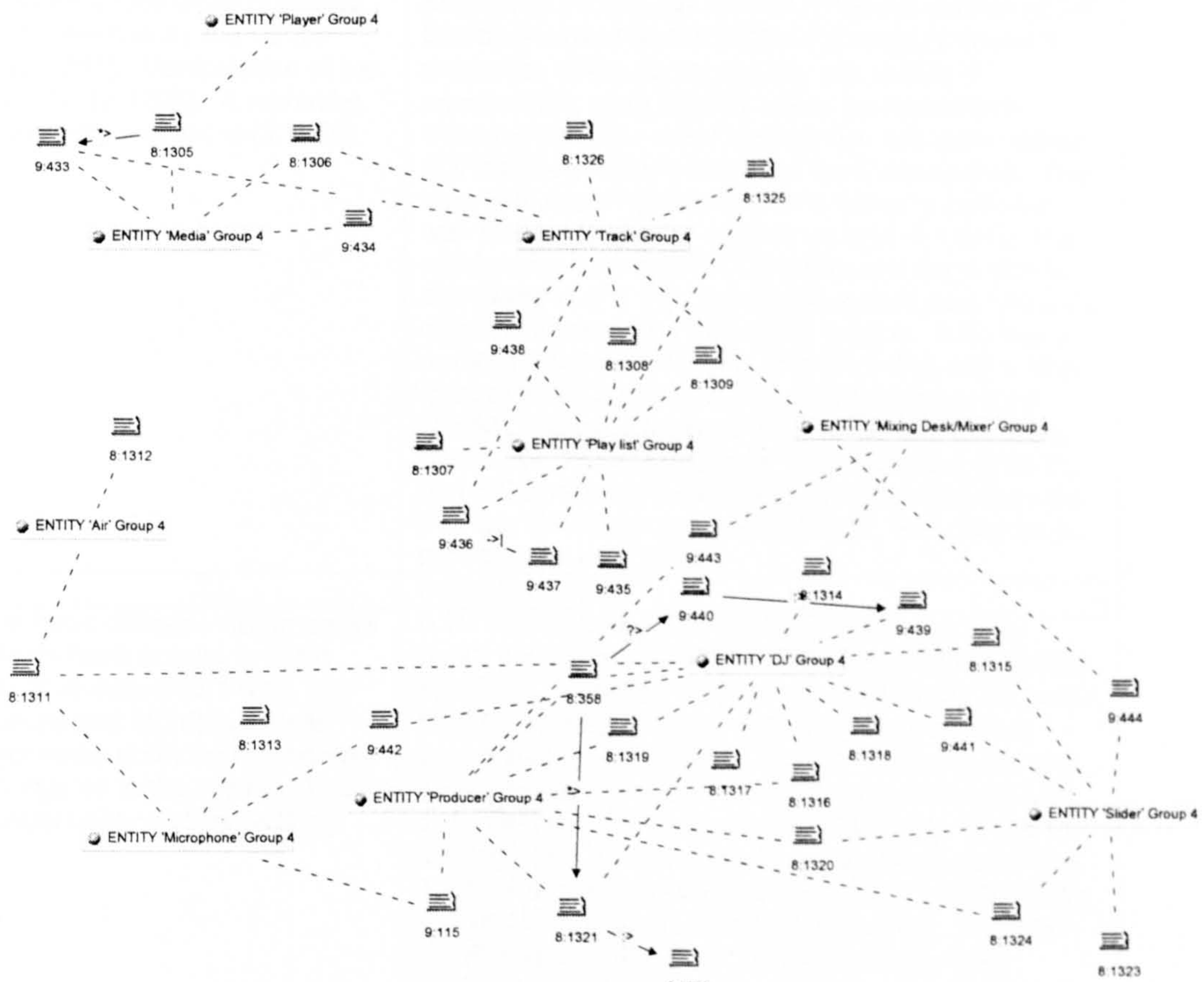
Group 4



Meta-Object nodes

Group 1

Group 4



Interactor nodes

BLANK IN ORIGINAL

Appendix E – USC Analysis data

THE DJ

Group 1	Group 4
Task phase	*Task phase*
The DJ directs the microphone 'channel' (1:1330), this is done through a slider (1:1323). The DJ is also responsible for adverts. A stop-iterate condition is identified for the playing of media – an action of the DJ. The playlist is also established as a guide for the playing of media for the DJ (of which there may be different types of play list) - (1:1304).	Suggested by the interviwer: the DJ manipulates audio sources in the real world (5:1439) - agreed by the group, but this is not part of their design. More analogical descriptions regarding the media and playlist found in (5:1438) (6:564). A vague stop-iterate condition is considered by the group with regard to DJ behaviour (5:230). DJs may be put on and off 'air' and have volume (5:1434) (5:1436) and has a certain amount of time related to the showlist (5:1428). The DJ sits within a hierarchy (5:404) that may dictate some of the properties of his 'output'. There is some discussion regarding whether or not the DJ may be able to play music to him/herself, but the play list is taken rather literally (6:564)
Meta-object phase	*Meta-object phase*
The DJ (and producer) have different types of mixer (2:1759) (2:1762). DJ messaging not really covered - not part of the mixing desk (2:1767) and only hinted at through the appearance of letters (2:1766). Media items playable by the DJ are held in the CD rack (2:1761). Manipulation of the play list is done by the DJ (2:1763). A hierarchy of music to air responsibility is given in (3:1160).	A DJ object 'is rather complex' (7:855) and is manipulable by the producer such that he/she can be placed on or off air and manipulate their audio properties. ALSO, the DJ has access to a mixer (8:26) that allows him/her the ability to change some audio properties of the sound output. His 'profile' is manipulable, such that his 'visual representation' changes (7:159) - some tech design talk here - some of the actions initially identified were not verified. The idea of a visual representation is futher re-inforced with the description of volume change (8:1301). The relationship between the DJ object and the profile is discussed in (7:117) - the visual presentation of the DJ object is informed by the profile (8:231). A DJ has a name (7:854) which is part of their profile, and a 'style' (part of their profile) which is selectable from a list (7:853). Group 4 suggest that the DJ object and the profile could conceptually be the same thing (7:852) (8:1299 in detail) and it is only at implementation time that this distinction is made (8:1300). A DJ can be a producer as well (7:125).
Interactor phase	*Interactor phase*
The DJ and Producer have different mixing desks (4:960). Listener clients have access to some tracks found in the DJ's inventory (3:1174). An additional design feature (not MO'd) is a 'note' with which the DJ can make notes for him/herself (4:959). When a DJ logs on to the system, their name appears in a label under a slider (4:957).	A DJ object appears interactively in the producer booth and 'passively' in the DJ booth (8:1316) (8:358) (8:1321) - however, it "does not exist" in the DJ booth found in (9:440) it is only a 'representation'. More contention on this subject - (8:1317) – the profile and DJ object are combined. This DJ object forms a part of an abstract (MO) visual queue (8:1319). Whilst in the producer, a red box provides feedback specifying whether an object is droppable (8:1311) – once on air, his name tums red to indicate he is on-air. A similar effect is described for the floating volume slider (8:1314), strongly in 'windows' terms. The 'profile' of the DJ is accessed via an icon (9:439) - it's manipulation is discussed in (8:1318). The DJ has sliders that he/she may use to manipulate audio (8:1315).

Group 1	Group 4
Task phase	*Task phase*
	The producer has a 'browser' for playlists from other DJs (5:1424), which have slots in time (5:1441). The producer puts DJs on and off air (5:1435)(5:497) - a mechanism of notification is discussed in (5:1437) and (5:1443) for this. The producer 'recruits' DJs for his show (5:1444). The producer is above the DJ in a hierarchy - the audio properties of the DJ are superceded by the producer (5:404). Communication between producer and DJ identified as possible gestures (5:1001) and suggests that the group have a model of DJs being physically separated in different rooms (5:1440). A producer creates a list for his show consisting of DJs (6:565).
Meta-object phase	*Meta-object phase*
	A producer may also be a DJ (7:125). The producer has two mixing 'boards' (7:137), one for master output (7:858) and another which he uses to change the audio output of DJ objects (representing the DJs) (7:136) - this includes volume (7:159) as 'output' from the DJ (7:856) as opposed to a property of the DJ him/herself (8:1301 in detail). This is manipulated using a slider (8:1302) in the producer's environment. The producer puts DJs objects on and off air (7:859) using the microphone (7:850) (8:1298).
Interactor phase	*Interactor phase*
	If a DJ is a producer as well, two rooms are presented to the user - a DJ booth and a producer booth - which are navigable (6:358) via a door and are considered 'virtual' (8:1322). DJ objects appear in the producer's booth (8:1316). In discussing the appearance of DJ objects in the room (MO discussion), a 'visual' queue is considered to represent the order in which DJs are to appear on air (8:1319) - the use of 'space' in metaphor is interesting here. The producer 'edits' the DJ's volume via his/her slider (8:1320) which 'floats' and attaches to the DJ.

MEDIA AND MEDIA OBJECTS

Group 1	Group 4
Task phase	*Task phase*
Media is selectable from a 'collection' - two types of selection are given - from a media object or from the playlist (1:16). Actions include play, stop and choose. Actions such as 'stop' are the playing devices affecting the media (1:1320). A track is something that can be found in a certain 'position' on a media device (1:1302); this is resovled later in discussing the role of choosing media with a device (1:1316).	Is playable, stoppable and pausable (5:1409, 5:1412); the media can be ejected - the CD object is used as an example (5:1411) this implies a playing device. Media is said to be included in a list (as tracks) which cannot be deviated from (6:115). Playing the media only occurs in tasks if it appears in the play list. (6:561). Selecting media (such as a record) is closely tied to the creation of the play list (5:1423) - your selection is recorded on the playlist (5:1423). Selecting media is choosing a media object and it's tracks (5:1410)(6:558) - the media object is a collection of tracks (6:560). In playing media, the actual focus is on playing, stopping etc the track (6:104) – but does this lead to problems with the play list? Where does the track belong? In the task world, it belongs to the media object (6:92).
Meta-object phase	*Meta-object phase*
Media can be listened to locally (using the Hi-Fi, (2:1732)) or sent out to the producer through the use of the mixer desk (2:1730). Media are confirmed as objects such as CDs during the selection and playing of media (2:1734), however the role of these objects diminishes as design discussion continues (2:1735). The CD object is used to represent the playlist (2:1742) both as an 'array' of CDs and also as a visual article (this is interesting). This gets developed into a 'CD RACK' object (2:1738) - although group 1 suggests it is difficult to 'align' the metaphor. This is hinted at further in (2:1776) since picking up CDs from a rack isn't what is going on - it's looking at the play list really.	Media is 'folded' into the track object (8:1288) (8:1283)
Interactor phase	*Interactor phase*
MAPPING – monitor to playing media, "not important", but in implementation (4:951). MAPPING task actions to media associated sliders (push not used as no button implemented) (4:948) see also (4:946) (4:950). MAPPING - playing media is associated with the mixer desk (4:952) and also MAPPING the play list (4:953), although only as selection of MP3s (4:950). ON THE 'CD' A CD is used to represent all the tracks as a playlist (!!) (3:1173).	A virtual media player is identified (8:1305) - these are icons - MAPPING from media player MO to icons (9:433). No 'actual' media player exists in their design claim group 4 (8:1306) - rather there is just the track operations by the virtual media player.

THE PLAY LIST

Group 1	Group 4
Task phase	*Task phase*
Actions associated with the playlist include creation of the list, adding and removing songs; these are suggested as write and erase actions although the analogy is not very strong here (1:1296). 'Shuffling' is broken down into separate write and erase actions. One team member prefers to resort to considering the implementation design. This view is emphasised by the initial definition of 'displaying MP3s' (1:1304). The play list is considered as an object - the term 'script' is suggested as a means of anthropomorphising. Time attributes are discussed - the play list is considered to relate to the 'show'. The 'voice' of the DJ is not something that could easily be accomodated (1:1308). Other assertions are made that the play list is a list of MP3s (1:1303) or a device itself (1:1298). Creating the play list is considered to be associated with 'planning' the play list (1:1305). Some 'stop-iterate' conditions are explored here in describing the use of the play list in the task model (1:1310 - 1:1311). *ON THE INVENTORY* The "inventory list" noun name is suggested by the interviewer (2;1745) since group 4 have difficulty finding a noun to describe the source of MP3 objects to choose from (although this object is referred to as "the actual list of songs in 2:1744 and emerges as a concept in (2:1763)) - this is because the MP3 object belongs in the computer domain already (they suggest). Interviewer then suggests 'possession' by the DJ, in effect holding (2:1746) - the group goes along with this, but does not add to the model.	An initial definition of the playlist is a schedule for the DJ (5:1422). An anlogical explanation can be found as the playlist as a 'box' of media objects (5:1438). Later: this list contains is a compiled list of 'tracks' (contained by 'media') (6:115) and the tracks appearing there are the ones that are played (6:561). Playing media and the use of the playlist are suggested to 'over lap' (5:1408). During the creation of the playlist, media is selected and made into a 'collection' for the show (5:302) (5:1423) (5:1413) - in fact it's the tracks that are selected, not the media itself (6:558) (6:560). Creation of the playlist has nothing to do with selecting media objects, but selecting 'tracks' (6:562). How does this 'track object' relate to the playlist (it belongs to the media object (6:92) - it (not the name) it's a property that gets 'transferred' (6:566) - this sounds like implementation talk. Tracks can be shuffled or moved around in the playlist (5:1447). Tracks in the playlist are ordered (5:312). The playlist seemed to be an object to which constraints could be applied to (5:1420) (5:1430) (5:451) (5:1429) (5:1427) (5:1428). Removing an entity from the playlist is not considered - the play list is a guide (5:1414).

Meta-object phase	*Meta-object phase*
The playlist begins to develop along two separate lines – in it's own terms and also as a 'CD rack' (2:1738) (2:1731) (2:1741)(2:1742). It is suggested that parts of the playlist are 'copied' to the monitor object (3:1176) (3:1154), but it was realised that this "wouldn't work". Objects can be selected, added and removed from this object (3:1159) - but with qualifications (2:1763) - an inventory list concept emerges. Difficulty with 'dynamic' aspects of the show (sources of media) are discussed in relation to the playlist and other media objects (3:1154) – is the playlist just becoming a holder for MP3 objects? I think so. Songs in the play list have order (2:1743) (2:1748) and may be removed (2:1747) but not from the inventory (2:1744). MP3 objects are 'copied' from the inventory to the playlist (3:1158) - this is done abstractly but not 'represented' as a copy (see context). Some consequences of action are suggested by the group here (2;1749), but they are ambiguous and not embedded in the metaphor model. *ON THE CD RACK* The development of the CD RACK (2:1776) (2:1738)(2:1731)(2:1742). A description of the rack is very similar to the playlist (2:1761) (2:1740). The rack object is used as a metaphorical 'representation' (2:1741) - and is associated with the playlist; it's graphical nature is described in terms of implementation. The rack comprises of two objects the inventory list and the play list (2:1749).	The primary role of the play list is to encapsulate tracks the DJ will be playing in an order (7:844). Tracks can be added and removed from the play list (7:847) as well as moved and possibly edited (7:204) (7:848). Actually adding tracks to the playlist is a problem since the metaphor design breaks down (7:845) (8:1285). 'Files' are selected from the windows dialogue and represented as tracks in the playlist (8:1290)(8:1287). Buttons are said to move tracks up and down the play list in order (7:847). Adding, removing and moving are 'editing' actions (8:1291). A playlist was not considered a 'moveable' object (8:1293) although it is as such in the implementation.
Interactor phase	*Interactor phase*
Clicking on the hi-fi object brings up the play list (4:967). This has a playlist and inventory list (3:1168) with various buttons to add and remove songs (3:1170) (3:1169) (3:1172). Although a play list can be activated/deactivated, group 1 suggest little interaction other than putting it back on the shelf (?!) (3:1173) *quick trip back to meta-objects suggests object representations here*. The mixer desk is 'connected' to the monitor which displays the currently playing track (4:962), which is the top track, *removed* from the playlist. The play list is associated with selection of MP3 objects to be played (4:950) rather than actual playing. A 'hide' action is suggested for the MO (3:1172). ON THE HI-FI The hi-fi is no longer associated with playing media (4:494) task model, and used instead to play jingles (adverts) - the reasons for this are given in (3:1171) – implementation problems. The hi-fi brings up the 'play list' - *reused* to display jingles (4:967)(4:949) and MAPPING is only used for selecting MP3s (4:950). The disappearance of the playing media function of the hi-fi is discussed in (4:954) as a 'massive change'.	The playlist is 'viewable' via an icon (8:1307). Tracks can be found in the play list (8:1308) and are added to the playlist via a file dialogue (8:1325). MAPPING adding/removing/moving tracks (9:435) from the tasks (9:438) - some difficulty, only a few parts mapped to tasks. A play list is 'movable' but only because it has been developed within the MS windowing system (8:1309).

PLAYER DEVICES

Group 1	Group 4
Task phase	*Task phase*
Devices are media players (include MP3 and CD as examples); the user chooses a device which may contain media objects (1:1301). In discussing the selection of media and the role of the device, some clarity is resolved in that a device selects the 'position' on the tape that is of interest (1:1316, 1:1302, 1:1320) - the action for doing this is 'skip' (1:1317) which allows the user to find the track they are interested in. A device may be 'activated', played (1:924, 1:1320). Devices also considered as part of delivering adverts (1:1300).	Very little is mentioned of devices with regard to media objects. These are either implied (5:1411)(6:560)(6:92) or suggested by the interviewer (5:1432).
Meta-object phase	*Meta-object phase*
We see the start of the disambiguation between media (and specialisation of media types such as CDs) and media playing devices in (2:1733). The 'device' object (a media player object) is associated with the mixer desk and an implied slider action (2:1752). The choice of media object (in playing media) determines which device will be 'activated' (2:1754). *ON THE 'HI-FI'* The 'player' turns into a Hi-Fi (2:1732) since it plays various media objects - the refinement of this partial model is discussed (2:1736). The term 'device' and 'player' are used inter-changeably when discussing player objects (2:1753). The implied association with the mixer and slider is introduced in (2:1755) in discussing the operation of the Hi-fi - the slider determines whether sound is output to air. Panels are established as having panels (for interaction) for the appropriate devices (3:1162) - these are activated (see context for broader discussion and 2:1754). IMPORTANT (?) (3:1157) - the actions on these panels for playing devices are linked to physical hardware on the computer, however whilst these design ideas are recognised they are rejected as not a part of the implementation (2:1735) (3:1163).	Group 4 argue that the media player was only ever conceived to play 'soft media' types (8:1282). (8:1287) the media player does not really exist in their metaphor design - the tracks 'play themselves', although a graphical part of the prototype is identified as part of it - the current track information (8:1284) and also as a 'trigger' for the track object to play itself (8:1287) - an underlying functional part?
Interactor phase	*Interactor phase*
The mixing desk now assumes this function (4:950).	A virtual media player is identified (8:1305) - these are icons - MAPPING from media player MO to icons (9:433). No 'actual' media player exists in their design claim group 4 (8:1306) - rather there is just the track operations by the virtual media player.

Group 1	Group 4
Task phase	*Task phase*
ON MP3s These are media objects 'stored' in the playlist (1:1299) - a technical point of view. MP3s are also considered as tracks or playlists (1:1303) - here there is some confusion and a need to clarify is identified.	A track is contained in media (6:115) (6:563) (6:121) (6:560) and is contained within the playlist and is movable. This is discussed a great deal with respect to playing media objects. Tracks are put on the playlist, not media objects (6:558) (6:562).
Meta-object phase	*Meta-object phase*
Track information is sourced from different media objects (CD, minidisc, MP3) and appears in different places too (monitor, the playlist)(3:1176) (3:1154). The nature of the track is identified as problematic. Jingles are "mini disc" tracks.	The nature of track is discussed as an entity (8:1289) – information parts of it (also described as a profile, like the DJ 'profile' 8:446) are visible in the environment but not easily pinned down as one single object. Track information can be edited (7:204) (7:848) whilst in the play list. A richer metaphor picture was discussed in (8:1292) but shelved because of Flash technology problems. The track is an entity (8:1283) that can be played, stopped, paused, hi-lighted and added and removed from the play list. The source of tracks is a problem and not consided by Group 4 (7:846) - in (7:845) the group resort to referring to their desktop implementation to explain track location, as 'stored media' on the PC. Although they recognise that this should not be important to the user. (8:1285) (7:842) (7:840) - the group explain they have to resort to the desktop metaphor - work effort. 'Files' are selected from the windows dialogue and represented as tracks in the playlist (8:1290) - these are lines of text (8:1286). The problem of the track as an important yet intangible object is explained by group 4 in (7:841) – the play list was created to display descriptive information about the invisible 'track' objects. During manipulation in the play list, tracks are 'high-lighted (7:847) indicating 'seleciton'. Tracks also 'play' themselves (8:1287) – no media player really exists - this is explained as in terms of code.
Interactor phase	*Interactor phase*
Selecting tracks is in effect selecting an MP3 object (4:953) (see also context after 3:1171). Tracks are displayed on the monitor (4:962) - playing a track involves 'flicking up' the slider, which removes the track from the playlist and 'dumps' the details into the monitor. Tracks are found by selecting an MP3, added and removed (songs), whilst separate in the MO, tracks and jingle playing have been joined in implementation (4:334). ON MP3 The play list is associated with *selection* of MP3 objects to be played (4:950) (4:953) (4:954) rather than actual playing. An MP3 and 'track' are more or less synominous (4:334) (4:950).	Tracks are 'manipulated' (actions like play, stop etc identified in implementation in (9:436)) - a focus of this originally was the media player (8:1306). Tracks exist in a list in the playlist (8:1308) can be selected (8:1326) and moved with UP/DOWN arrows. Tracks are added to the playlist via a file dialogue (8:1325).

THE MIXER OBJECT

Group 1	Group 4
Task phase	*Task phase*
The mixer desk controls the audio output to 'air' (an 'up' slider determines broadcast, 1:246). The mixer object uses sliders to control the mic (multiple instances) output going out to air (1:1323, 1:1333 and 1:1322), a number of sliders belong to a number of different DJs in different 'rooms'. Buttons are considered as operators here too. The slider model is then challenged with choosing the device to use to play the media (1:1318) - a qualifier to the action. The moving of sliders is considered as the mixing of audio 'sources' (1:1315), each of which is related to a player device (1:376). However, some hint to implementation is given in (1:1322) where the sliders literally operate the 'play' and 'stop' actions of the media devices.	Has buttons (5:1431) and is the source of audio for mixing (5:1445) - in the 'real world' these have 'hundreds' of parameters (5:1446).
Meta-object phase	*Meta-object phase*
Two types of mixer desk exist (2:1768) one for the DJ and one for the producer (2:1759). Media can be listened to locally or sent out to the producer through the use of the mixer desk (2:1730) - this is a select and (slide up action - implicit) (2:1752)(2:1755) – more explicitly stated in as sliders for particular playing devices in (2:1756), slide and push actions suggested for this (2:1762); clarified in (2:1771) - buttons used to determine whether the audio is going to air (ambiguous before). 'Audio channels' and chatting not a part of the mixing desk (2:1767). *ON THE SLIDER* The slider develops as both an indicator of a particular media device (including microphones and MP3s [3:1155)), and as a attenuator of volume for that device (2:1755), but not air space (2:1756) (2:1771), CONTRADICTED (2:1730) and later in (3:1160). The DJ operates these objects (2:1762). The state of the slider determines what goes on air and also what gets displayed on the monitor (3:1161). An interesting XOR model of the slider behaviour on the mixer desk is described in (2:1770) - some state modelling occurs here and also in (2:1739) in describing slide actions. Another function for the button (and 'light') to indicate queued audio (2:424).	A 'DJ mixer' allows the DJ to change his/her audio *output* (8:26) (8:1327 : interesting - see context - the group identify constraints with the parameters of the slider) - through the manipulation of an associated slider (7:843). The producer has two types of mixing board, one of which is used in a 'drag-and-drop' sense to control the output of a DJ object (7:137) - see (8:1302) to explain manipulating the DJ's volume - the slider is in fact 'attached' to the DJ (7:857). ON THE SLIDER Sliders are used to manipulate the audio properties of a DJ (7:843) - this object can be attached to the DJ (7:857) (8:1302). It is the property of the audio ouput of the DJ (8:1301) which is changed here, this is a separate effect to the master volume slider (7:137) (7:858). Constraints of the slider object are identified by group 4 (8:1327).

THE MIXER OBJECT (continued)

Interactor phase	*Interactor phase*
MAPPING – playing media is associated with the mixer desk (4:952) but no 'mixing' as such is implemented, sliders are associated with DJs from the producer's point of view (4:960). The mixer desk is 'connected' to the monitor which displays the currently playing track (4:962), which is the top track, *removed* from the playlist. Only one slider may be up for a media source at any one time (4:947) - each source is labelled (4:960). DJ labelling occurs on the producer desk when a new DJ logs in (4:957). A button on the mixing desk changes it from a DJ desk to a producer desk (4:956). ON THE SLIDER MAPPING task actions to media associated sliders (push not used as no button implemented) (4:948). Sliders are 'clicked' and they "pop up to the top" (a binary state system (4:960)) (4:946) (4:962) - with the mouse (4:947) - only one slider may be up at any one time. MAPPING: slide action used for slider (4:948). Sliders associate with media player objects and volume (4:963); only one of the media sliders may be up at any one time. The volume slider DOES have sliding, continuous motion (4:964), MAPPING as drag action (4:965). MAPPING: fading in and out is really a switch (4:966) when adjusting sliders.	Sliders vary in use. The DJ has some sliders for manipulating his/her audio (8:1315) whilst the producer has the same, and a volume manipulator for the audio output from the DJ (8:1314) (9:441)(9:443). Sliders are MAPPED to mixing (9:444). Their implementation is component based and re-used for different audio properties (8:1323) and used across booths (8:1324)(8:1320) – other than master audio.

THE MICROPHONE AND AIR

Group 1	Group 4
Task phase	*Task phase*
The microphone is considered as either on or off air, with a volume value (adjustable on the mixer desk, 1:1333) and as a 'source' (1:1315). The output of the microphone is a 'channel' which is directed (1:1325, 1:1326). These actions are associated with the slider and the mixer desk (1:1323). Sources are identified as 'input' to the mixer desk (1:1334).	*ON MIC* The microphone is a media 'source' (5:325), which has volume (5:358) and that can be on or off (5:113). *ON AIR* May have DJs 'put on' and 'taken off' it and considered part of 'mixing' task done by the producer (5:497)(5:1442)(5:1443) - the procedure for this is discussed in 5:1437.
Meta-object phase	*Meta-object phase*
ON MIC It's output to air has volume (2:1758), an associated slider (3:1155) and can be seen through the monitor (3:276). *ON AIR* Sound output to 'air' is determined by the associated slider object on the mixer desk (2:1755) and can be viewed on the monitor (3:1175) (3:276) and is determined by the button or slider (2:1771) (2:1756) (3:1161) - this is contentious - this is controlled by the DJ or producer	ON MIC A producer 'drags and drops' the microphone onto the DJ to put them on air. (8:1298) (also referred as 'activating' (7:849)) - the microphone is 'attached' to the DJ (7:850), putting him on air (8:23). The states of DJ objects are linked with microphone manipulation and the use of the stand identified as taking all DJs off air (8:25) (7:851). ON AIR DJs may be on or off air (7:136) (7:859). Placing the mic on the stand removes all DJs from the air (7:851) (8:25).
Interactor phase	*Interactor phase*
	ON MIC Snapping behaviour of microphone to DJs and stand discussed in (8:1311) (includes visual feedback) - the microphone stand is a 'holder'. Task MAPPING (9:442) for microphone drag and drop - used by the producer (9:115). ON AIR Light reference is a button and status graphic determining whether the DJ is on air (8:1312). The interaction mechanism for placing a DJ on air (including feedback) detailed in 8:1311 - 'snapping' of microphone to either stand or DJ discussed.

THE ADVERT

Group 1
Task phase
Considered as audible and visual. These entities belong to media objects and playing an advert is part of choosing a media object and appropriate player (discussed in 1:310 and refined in 1:1300) - done by the DJ (1:341).
Meta-object phase
Early inspection of the actions for the advert suggest an advert inventory (2:367) (2:1774) and later (3:1167). Adverts are contained within a book object (3:1164) (3:1167) and can be added to a 'queue'. In the 'real world' these adverts would appear on tapes (2:1757) (2:1772) - but this was not considered further. The book sits of a shelf (3:1165), can be activated, and also contains a 'timeline' (3:1165) which appears both in the book and on the wall at the same time (some reference to implementation here). The timeline also acts as a point of delivery for the DJ to send an advert (3:1166). Some rather odd references to adverts being a part of the bookshelf, rather than the book (2:1775 and 2:1773 and 2:1765) {possible interview influence here}. In discussing the placing of adverts from the inventory to the timeline, the group associate the mouse with the metaphor of a hand (3:1159) - but it's NOT important to the design. Is similar to the play list (2:563) (2:1737) (3:1167) (2:1751) - actions for the advert are mapped back to the play list. Objects can be selected, added and removed from this object (3:1159). The 'copying' model for the adverts from the inventory to the time line is mapped back to the playlist (2:1751). It was difficult for group 1 to think of a metaphor for the time line, although they agreed that with expansion, it could behave like a calendar or diary (2:1750).
Interactor phase
{The advert is somewhat under developed in the MO, but comes out in detail here}... Adverts are draggable (although no interaction is claimed in (4:978)) from an inventory part to a 'timeline' in a 'book' (4:975) (4:972) - detailed in (4:970) and (4:969). Adverts exist in pages which can be navigated through via buttons (4:976), although this was not implemented (4:974). Duplicity in the advert/timeline development is identified by group 1 (4:973). MAPPING: tasks for the advert and time line identified as being analogous to the playlist (4:979). Some MO discussion of the advert/time-line is found in (4:971).

THE SHOW

Group 4
Task phase
The show is considered to consist of a list of DJs (5:1424) (5:376) (5:1428) very much like the playlist and also be associated with a collection of 'media' (5:302). An analogy of the 'show list' is that of a schedule (5:1441) in which DJs are assigned a slot and a time. Some discussion for a stop-iterate condition in the showlist with regard to DJ presence (5:1421) - difficult to model since the mechanics of the show are not fully worked out. The creation of the playlist is suggested first as working with paper, and then moved onto computer support (6:565) - expanded more in 5:1425 - DJs can be ordered.
Meta-object phase
In fact the show list is under-developed and has little relevance to group 4 (8:1294) but a brief description is given...The 'show list' is like a playlist in which DJs are ordered in turn of being placed on air (8:1296) – at least "that's the plan". The producer organises the list - DJs appear one after another in order of connection - this mechanism is virtually identical to the playlist and also "like a note board" - DJ profile information is manipulated (8:1295). DJ objects arranged in a "visual queue" suggest an alternative show list model (8:1319).

THE ROOM

Group 1	Group 4
Task phase	*Task phase*
DJs are physically located in separate rooms (1:1286)	DJs are physically separated by rooms (5:1440).
Meta-object phase	*Meta-object phase*
	The DJ has a 'booth' (8:1303)

BLANK IN ORIGINAL

References

ACCOT, J., CHATTY, S., MAURY, S. AND PALANQUE, P., *1998.* Formal Transducers: Models of Devices and Building Bricks for the Design of Highly Interactive Systems. *In: 4th Eurographics Workshop on Design, Specification and Verification of Interactive Systems,* Markopoulos, P. AND Johnson, P., *Eds*. Granada, Spain, Springer-Verlag, pp 143-159

ACCOT, J., CHATTY, S. AND PALANQUE, P., 1996. A Formal Description of Low Level Interaction and its Application to Multimodal Interactive Systems. *In: 3rd International Eurographics Workshop on Design, Specification, and Verification of Interactive Systems,* Bodart, F. AND Vanderdonckt, J., *Eds*. Namur, Belgium, Springer-Verlag, pp 92-104

ADAMS, S. S., 1988. MetaMethods: The MVC Paradigm.Hoopla!, 1 (4).

AKOUMIANAKIS, D. AND STEPHANIDIS, C., 2000. Multiple Metaphor Environments: Architectural premises for continuous interactions. *In: "Continuity in Human Computer Interaction", Workshop in the context of the Conference on Human Factors in Computing Systems (CHI 2000 - The Future is Here),,* Turner, T., Szwillus, G. AND Czerwinkski, M., *Eds*. The Hague, The Netherlands, ACM Press, pp 6

ALENCAR, P. S. C., COWAN, D. D., LUCENA, C. J. P. AND NOVA, L. C. M., 1995. Specification of Application and Interface Objects for Reuse. University of Waterloo

ALTY, J. L. AND KNOTT, R. P., 1999. Metaphor and human computer interaction: a model based approach. *In: Proceedings of Computation for Metaphors, Analogy and Agents: An International Workshop,* Nehaniv, C. L., *Eds*. Springer-Verlag, pp 307-321

ALTY, J. L., KNOTT, R. P., ANDERSON, B. AND SMYTH, M., 2000. A framework for engineering metaphor at the user interface. *Interacting with Computers,* 13 (2), pp 301-322

ANNETT, J. AND DUNCAN, K., 1967. Task Analysis and Training Design. *Occupational Psychology,* 41 pp 211-227

ARK, W., DRYER, D. C., SELKER, T. AND ZHAI, S., 1998. Representation Matters: the Effect of 3D Objects and a Spatial Metaphor in a Graphical User Interface. *In: Proceedings of HCI 98, the Conference on Human-Computer Interaction,* Johnson, H., Nigay, L. AND Roast, C., *Eds*. Springer, pp 209-219

ATKINSON, B., 1987. HyperCard *Apple Computer*

AVISON, D., LAU, F., MYERS, M. AND NIELSEN, P. A., 1999. Action research. *Communications of the Acm,* 42 (1), pp 94-97

BARNARD, P. J. AND MAY, J., 1999. Representing cognitive activity in complex tasks. *Human-Computer Interaction,* 14 (1-2), pp 93-158

BASS, A., ASPINALL, J., WALTERS, G. AND STANTON, N., 1995. A Software Toolkit for Hierarchical Task-Analysis. *Applied Ergonomics,* 26 (2), pp 147-151

BASTIDE, R. AND PALANQUE, P., 1999. A visual and formal glue between application and interaction. *Journal of Visual Languages and Computing,* 10 (5), pp 481-507

BASTIDE, R., PALANQUE, P., LE, D. AND MUÑOZ, J., 1998. Integrating rendering specifications into a formalism for the design of interactive systems. *In: DSV-IS '98: 5th International Eurographics Workshop on Design, Specification and Verification of Interactive Systems,* Markopoulos, P. AND Johnson, P., *Eds*. Abingdon, UK, Springer, pp 171-190

BAUMEISTER, L. K., JOHN, B. E. AND BYRNE, M. D., 2000. A Comparison of Tools for Building GOMS models. *In: Proceedings of CHI 2000 Conference on Human Factors in Computing Systems,* Turner, T., Szwillus, G. AND Czerwinkski, M., *Eds*. The Hague, The Netherlands, pp 502-509

BEARD, D. V., ENTRIKIN, S., CONROY, P., WINGERT, N. C., SCHOU, C. D., SMITH, D. K. AND DENELSBECK, K. M., 1997. Quick GOMS: A Visual Software Engineering Tool for Simple Rapid Time-Motion Modeling. *Interactions,* 4 (3), pp 31-36

BENYON, D., 1996 Domain Models for User Interface Design. *In:* Critical Issues in User Interface Systems Engineering, Benyon, D. AND Palanque, P. Eds. Springer-Verlag, pp 3-20.

BENYON, D. AND IMAZ, M., 1999. Metaphors and Models: Conceptual Foundations of Representations in Interactive Systems Development. *Human-Computer Interaction,* 14 pp 159-189

BENYON, D. AND MACAULAY, C., 2002. Scenarios and the HCI-SE design problem. *Interacting with Computers,* 14 (4), pp 397-405

BLANDFORD, A. E. AND DUKE, D. J., 1997. Integrating user and computer system concerns in the design of interactive systems. *International Journal of Human-Computer Studies,* 46 (5), pp 653-679

BODART, F., HENNEBERT, A., LEHEUREUX, J., PROVOT, I. AND VANDERDONCKT, J., 1994. A Model-Based Approach to Presentation: A Continuum from Task Analysis to Prototype. *In: Eurographics Workshop on Design, Specification, and Verification of Interactive Systems, Eds*. Bocca di Magra, Italy, pp 25-39

BOLOGNESI, T. AND BRINKSMA, E., 1987. Introduction to the Iso Specification Language Lotos. *Computer Networks and Isdn Systems,* 14 (1), pp 25-59

BORLAND, 2001. C++ Builder 4. *Borland Software Corporation*

BRADLEY, N., 1998. *The XML companion,* 3rd. Addison Wesley

BRAIN, M., 1992. *Motif Programming: The Essentials... and More,* Digital Press

BREEDVELT-SCHOUTEN, I., PATERNÒ, F. AND SEVERIINS, C., 1997. Reusable Structures in Task Models. *In: 4th International Eurographics Workshop on Design, Specification, and Verification of Interactive Systems,* Torres, J. C., *Eds*. Granada, Spain, Springer, pp 225-239

BROWN, J., GRAHAM, T. C. N. AND WRIGHT, T., 1998. The Vista Environment for the Coevoluntionary Design of User Interfaces. *In: CHI '98: ACM SIGCHI Conference on Human Factors in Computing Systems,* Karat, C., Karat, J. AND Horrocks, I., *Eds*. Los Angeles, USA, ACM Press, pp 376-383

BRUN, P. AND BEAUDOUIN-LAFON, M., 1995. A Taxonomy and Evaluation of Formalisms for the Specification of Interactive Systems. *In: Proceedings of the HCI'95 Conference, People and Computers X,* Kirby, M. A. R., Dix, A. J. AND Finlay, J. E., *Eds*. Cambridge University Press, pp 197-212

BUTTERWORTH, R., BLANDFORD, A. AND DUKE, D., 1999. Using Formal Models to Explore Display-Based Usability Issues. *Journal of Visual Languages and Computing,* 10 pp 455-479

CABRERA, M., TORRES, J. C. AND GEA, M., 1999. Towards User Interfaces Prototyping from Algebraic Specification. *In: 6th International Eurographics Workshop on Design, Specification and Verification of Interactive Systems,* Duke, D. J. AND Puerta, A., *Eds*. Braga, Portugal, Springer, pp 67-81

CALVARY, G., COUTAZ, J. AND NIGAY, L., 1997. From Single-User Architectural Design to PAC*: a Generic Software Architecture Model for CSCW. *In: CHI '97: Conference on Human Factors in Computing Systems,* Pemberton, S., *Eds*. Atlanta, Georgia, USA, ACM Press, pp 242-249

CAMPOS, J. C. AND HARRISON, M. D., 1997. Formally Verifying Interactive Systems: A review. *In: 4th International Eurographics Workshop on Design, Specification, and Verification of Interactive Systems,* Harrison, M. D. AND Torres, J. C., *Eds*. Granada, Spain, pp 109-124

CANTER, M., 1988. Director *Macromedia*

CARD, S. K., MACKINLAY, J. D. AND SHNEIDERMAN, B., 1999. *Readings in Information Visualization: Using Vision to Think,* San Francisco, CA, Morgan Kaufmann Publishers

CARD, S. K., MORAN, T. P. AND NEWELL, A., 1983. *The Psychology of Human-Computer Interaction,* New Jersey, Lawrence Erlbaum Associates, Inc.

CARR, D., 1997 Interaction Object Graphs: An Executable Graphical Notation for Specifying User Interfaces. *In:* Formal Methods for Computer-Human Interaction, Palanque, P. AND Paternò, F. Eds. Springer-Verlag, pp 141-156.

CARR, D. A., 1996. Toward more understandable user interface specifications. *In: 3rd International Eurographics Workshop on Design, Specification, and Verification of Interactive Systems,* Bodart, F. AND Vanderdonckt, J., *Eds*. Namur, Belgium, Springer, pp 141-161

CARROLL, J. M., 1997. Human-Computer Interaction: Psychology as a Science of Design. *Human-Computer Studies,* 46 pp 501 - 522

CARROLL, J. M., 2002. Making use is more than a matter of task analysis. *Interacting with Computers,* 14 (5), pp 619-627

CATTELL, R. G. G., BARRY, D. K., BARTELS, D., BERLER, M., EASTMAN, J., GAMERMAN, S., JORDAN, D., SPRINGER, A., STRICKLAND, H. AND WADE, D. 1997 *The Object Database Standard: ODMG 2.0,* Morgan Kaufmann

CHUNG, C. M. AND SHIH, T. K., 1997. On automatic generation of multimedia presentations. *Information Sciences,* 97 (3-4), pp 293-321

COUTAZ, J., NIGAY, L. AND SALBER, D., 1995 Agent-Based Architecture Modelling for Interactive Systems. *In:* Critical Issues in User Interface Engineering, Palanque, P. AND Benyon, D. Eds. Springer-Verlag, pp 191-209.

CROWLE, S. AND HOLE, L., 2001. Seeing the wood for the trees: A framework for the specification of metaphor in interface design. *In: Workshop on Integrating Multimedia, Metaphors and Multimodality , in PC-HCI 2001: Human Computer Interaction 2001, Eds.* Patras, Greece, Typorama Publishers, pp 19-24

CROWLE, S. AND HOLE, L., 2003. An Interface Specification Meta-Language. *In: DSV-IS 2003 : Issues in Designing New-generation Interactive Systems Proceedings of the Tenth Workshop on the Design, Specification and Verification of Interactive Systems,* Jorge, J. A., Nunes, N. J. AND Cunha, J. F., *Eds.* Funchal, Madeira Island, Portugal, Springer, pp 381-396

DA SILVA, P. P., 2000. User interface declarative models and development environments: A survey. *In: Interactive Systems. Design, Specification, and Verification, 7th International Workshop, DSV-IS 2000,* Palanque, P. AND Paternò, F., *Eds.* Limerick, Ireland, Springer, pp 207-226

DA SILVA, P. P., 2001. User interface declarative models and development environments: A survey. *In: Interactive Systems. Design, Specification, and Verification, 8th International Workshop, DSV-IS 2001,* Johnson, C., *Eds.* Glasgow, Scotland, Springer-Verlag Berlin, pp 207-226

DA SILVA, P. P., GRIFFITHS, T. AND PATON, N. W., 2000. Generating User Interface Code in a Model-Based User Interface Development Environment. *In: Advanced Visual Interfaces,* di Gesù, V., Levialdi, S. AND Tarantino, L., *Eds.* Palermo, Italy, ACM Press, pp 155-160

DA SILVA, P. P. AND PATON, N. W., 2000. UMLi: The Unified Modeling Language for Interactive Applications. *Uml 2000 - the Unified Modeling Language, Proceedings,* 1939 pp 117-132

DEARDEN, A. M. AND HARRISON, M. D., 1997. Abstract models for HCI. *International Journal of Human-Computer Studies,* 46 (1), pp 153-178

DEGENER, J. 1995, http://www.lysator.liu.se/c/ANSI-C-grammar-y.html, 18th April, 2003

DIAPER, D. 1989 *Task Analysis for Human-Computer Interaction,* Ellis Horwood Limited

DIAPER, D., 2002. Scenarios and task analysis. *Interacting with Computers,* 14 (4), pp 379-395

DIAPER, D. AND STANTON, N. A. 2003 - in press *The handbook of task analysis for human-computer interaction,* Lawrence Erlbaum Associates

DIEBERGER, A. AND FRANK, A. U., 1998. A city metaphor to support navigation in complex information spaces. *Journal of Visual Languages and Computing,* 9 (6), pp 597-622

DIX, A. J., 1991. *Formal methods for interactive systems,* Academic Press

DIX, A. J., FINLAY, J. E., ABOWD, G. D. AND BEALE, R., 1998. *Human-Computer Interaction,* 2nd. Prentice Hall

DOHERTY, G. AND HARRISON, M. D., 1997. A Representational Approach to the Specification of Presentations. *In: 4th International Eurographics Workshop on Design, Specification, and Verification of Interactive Systems,* Harrison, M. D. AND Torres, J. C., *Eds*. Granada, Spain, Springer, pp 273-290

DOWELL, J. AND LONG, J., 1989. Towards a conception for an engineering discipline of human factors. *Ergonomics,* 32 (11), pp 1513-1535

DU, M. AND ENGLAND, D., 2001. Temporal Patterns for Complex Interaction Design. *In: Design, Specification and Verification of Interactive Systems: 2001,* Johnson, C., *Eds*. Glasgow, Scotland, Springer, pp 114-127

DUKE, D. AND HARRISON, M., 1993. Abstract Interaction Objects. *Computer Graphics Forum,* 12 pp 25-36

DUKE, R., ROSE, G. AND SMITH, G., 1995. Object-Z - a Specification Language Advocated for the Description of Standards. *Computer Standards & Interfaces,* 17 (5-6), pp 511-533

EDMONDS, E., 1992 The Emergence of the Separable User Interface. *In:* The Separable User Interface, Edmonds, E. Eds. Academic Press, pp 5-17.

ELSAID, M. G., FISCHER, G., GAMALELDIN, S. A. AND ZAKI, M., 1997. ADDI: A tool for automating the design of visual interfaces. *Computers & Graphics,* 21 (1), pp 79-87

FAULKNER, X. AND CULWIN, F., 2000. Enter the usability engineer: integrating HCI and software engineering. *In: Annual Joint Conference Integrating Technology into Computer Science Education: Proceedings of the 5th annual SIGCSE/SIGCUE ITiCSEconference on Innovation and technology in computer science education, Eds*. Helsinki, Finland, ACM Press, pp 61-64

FIELDING, N. G. AND LEE, R. M. 1998 *Computer Analysis and Qualitative Research,* Sage Publications

FILHO, D. S. AND LIESENBERG, K. E., 1999. Capturing Computer-Human Interaction Design via the Protagonist Action Notation. *In: Proceedings of the 18th Brazilian Computer Society Annual Conference, Eds*. Belo Horizonte, MG, Brazil, pp 276-296

FINNE, S. AND JONES, S. P., 1995. Pictures: A simple structured graphics model. *In: Glasgow Functional Programming Workshop, Eds*. Ullapool, pp 1-20

FORBRIG, P., 1999. Task- and Object-Oriented Development of Interactive Systems - How many models are necessary? *In: Proceedings of the Eurographics Workshop on Design, Specification and Verification of Interactive Systems '99,* Duke, D. J. AND Puerta, A., *Eds*. Braga, Portugal, Springer, pp 225-237

FOWLER, M. AND SCOTT, K., 2000. *UML Distilled,* 2. Booch, G., Jacobson, I. AND Rumbaugh, J., Addison-Wesley

FRANK, M. AND FOLEY, J. D., 1993. Model-based user interface design by example and by interview. *In: ACM Symposium on User Interface Software and Technology, Eds*. Atlanta, Georgia, pp 129-137

FURTADO, E., FURTADO, J., SILVA, W., RODRIGUEZ, D., TADDEO, L., LIMBOURG, Q. AND VANDERDONCKT, J., 2001. An Ontology-Based Method for Universal Design of User Interfaces. *In: IHM-HCI 2001: Interaction without frontiers, Eds*. Lille, France

GARLAND, A., RYALL, K. AND RICH, C., 2001. Learning Hierarchical Task Models by Defining and Refining Examples. *In: First International Conference on Knowledge Capture K-CAP 2001, Eds.* Victoria, B.C., Cananda, ACM, pp 44-51

GENTNER, D., BOWDLE, B., WOLFF, P. AND BORONAT, C., 2001 Metaphor is like analogy. *In:* The analogical mind: Perspectives from cognitive science, Gentner, D., Holyoak, K. J. AND Kokinov, B. N. Eds. MIT Press, pp 199-253.

GILLAN, D. J. AND BIAS, R. G., 1994. Use and Abuse of Metaphor in Human-Computer Interaction. *In: Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics, Eds*. San Antonio, pp 1434-1439

GLASER, B. G. AND STRAUSS, A. L., 1967. *The discovery of grounded theory; strategies for qualitative research,* New York, Aldine de Gruyter

GOLOVCHINSKY, G. AND CHIGNELL, M. H., 1997. The newspaper as an information exploration metaphor. *Information Processing & Management,* 33 (5), pp 663-683

GRAHAM, T. C. N., MORTON, C. A. AND URNES, T., 1996. Clockworks: Visual programming of component-based software architectures. *Journal of Visual Languages and Computing,* 7 (2), pp 175-196

GRAHAM, T. C. N. AND URNES, T., 1996. Linguistic support for the evolutionary design of software architectures. *In: 18th International Conference on Software Engineering (ICSE '96), Eds.* Berlin, Germany, IEEE Computer Society Press, pp 418-427

GRAM, C. AND COCKTON, G. 1996 *Design Principles for Interactive Software,* Chapman and Hall

GRAY, P. D., ENGLAND, D. AND MCGOWAN, S., 1994. XUAN: Enhancing the UAN to capture temporal relationships among Actions. *In: Proceedings of the BCS HCI '94,* Cockton, G., Draper, S. W. AND Weir, G. R. S., *Eds.* Glasgow, UK, Cambridge University Press, pp 26-49

GREEN, M., 1983 Report on Dialogue Specification Tools. *In:* User Interface Management Systems, Pfaff, G. Eds. Springer-Verlag, pp 9-20.

GRIFFITHS, T., BARCLAY, P. J., PATON, N. W., MCKIRDY, J., KENNEDY, J., GRAY, P. D., COOPER, R., GOBLE, C. A. AND DA SILVA, P. P., 2001. Teallach: a model-based user interface development environment for object databases. *Interacting with Computers,* 14 (1), pp 31-68

HALASZ, F. AND MORAN, T., 1982. Analogy Considered Harmful. *In: Human Factors in Computer Systems Conference, 1982, Eds.* NBS, pp 383-386

HALL, A., 1990. Seven Myths of Formal Methods.IEEE Software, 7 (5). September

HAREL, D., 1987. Statecharts - a Visual Formalism for Complex Systems. *Science of Computer Programming,* 8 (3), pp 231-274

HARTSON, H. R., SIOCHI, A. C. AND HIX, D., 1990. The Uan - a User-Oriented Representation for Direct Manipulation Interface Designs. *Acm Transactions on Information Systems,* 8 (3), pp 181-203

HIX, D. AND HARTSON, H. R., 1993. *Developing User Interfaces : Ensuring Usability Through Product & Process,* John Wiley and Sons

HOARE, C. A. R., 1985. *Communicating Sequential Processes,* Hoare, C. A. R., Prentice Hall

HOLE, L., CROWLE, S. AND MILLARD, N., 1998. The Motivational User Interface. *In: Human-Computer Interaction'98,* May, J., Siddiqi, J. AND Wilkinson, J., *Eds.* Sheffield Hallam University, UK, pp 68-69

HORROCKS, I., 1999. *Constructing the User Interface with Statecharts,* Addison-Wesley

HUDSON, S. E., 1994. User Interface Specification Using an Enhanced Spreadsheet Model. *ACM Transactions on Graphics,* 13 (3), pp 209-239

HUSSEY, A., 2000. Formal Object-Oriented User-Interface Design. *In: 2000 Australian Software Engineering Conference,* Douglas, D. D., *Eds.* Gold Coast, Queensland, Australia, IEEE, pp 129-137

HUSSEY, A. AND CARRINGTON, D., 1997. Comparing the MVC and PAC Architectures: a Formal Perspective. *IEE Proceedings of Software Engineering,* 144 (4), pp 224-236

HUSSEY, A. AND CARRINGTON, D., 1998. Which widgets? Deriving implementations from formal user-interface specifications. *In: DSV-IS '98: 5th International Eurographics Workshop on Design, Specification and Verification of Interactive Systems,* Markopoulos, P. AND Johnson, P., *Eds*. Abingdon, UK, Springer, pp 206-224

HUSSEY, A. AND CARRINGTON, D., 1999. Platform Independent Graphical User Interface Design. Software Verification Research Centre, University of Queensland

INDURKHYA, B., 1986. Constrained Semantic Transference - a Formal Theory of Metaphors. *Synthese,* 68 (3), pp 515-551

ISO, 1986. 8879:1986 Information Processing -- Text and Office Systems -- Standard Generalized Markup Language (SGML). International Organisation for Standardization

JAAKSI, A., 1995. Object-Oriented Specification of User Interfaces. *Software-Practice & Experience,* 25 (11), pp 1203-1221

JACKSON, M., 2001. *Problem Frames: Analyzing and structuring software development problems,* Addison-Wesley

JACOB, R. J. K., 1985. A State Transition Diagram Language for Visual Programming. *Computer,* 18 (8), pp 51-59

JACOB, R. J. K., DELIGIANNIDIS, L. AND MORRISON, S., 1999. A Software Model and Specification Language for Non-WIMP User Interfaces. *ACM Transactions on Computer-Human Interaction,* 6 (1), pp 1-46

JACOBSON, I., CHRISTERSON, M., JONSSON, P. AND ÖVERGAARD, G., 1992. *Object-Oriented Software Engineering: A Use Case Driven Approach,* 1. Addison-Wesley

JAMBON, F., GIRARD, P. AND BOISDRON, Y., 1999. Dialogue Validation from Task Analysis. *In: Proceedings of the 6th International Eurographics Workshop of Design, Specification and Verification of Interactive Systems '99,* Duke, D. J. AND Puerta, A., *Eds*. Braga, Portugal, Springer, pp 205-224

JOHNSON, P., JOHNSON, H. AND WILSON, S., 1995 Rapid Prototyping of User Interfaces Driven by Task Models. *In:* Scenario-based Design, Carroll, J. Eds. John Wiley and Son, pp 209-246.

JONES, C. B., 1986. *Systematic software development using VDM,* Hertfordshire, UK, Prentice Hall International (UK) Ltd

KAHNEY, H. AND EISENSTADT, M., 1982. Programmers' mental models of their programming tasks. *In: Proceedings on the Fourth Annual Meeting of the Cognitive Science Society, Eds*. pp 143-145

KAY, J. 1999 *User Modeling: Proceedings of the Seventh International Conference, UM99,* Springer Verlag

KELLE, U., 1995. *Computer-Aided Qualitative Data Analysis : Theory, Methods and Practice,* Sage Publications

KHALIFA, M. AND KIRA, D., 1992. A Graphical Task Analysis Language (GTAL). *INFOR,* 31 (2), pp 65-79

KIERAS, D. E., 1988 Towards a Practical GOMS Model Methodology for User Interface Design. *In:* Handbook of Human-Computer Interaction, Helander, M. Eds. Elsevier Science Publishers, B.V., pp 135-157.

KIERAS, D. E. AND MEYER, D. E., 1997. An overview of the EPIC architecture for cognition and performance with application to human-computer interaction. *Human-Computer Interaction,* 12 (4), pp 391-438

KIM, W. C. AND FOLEY, J. D., 1990. DON: user interface presentation design assistant. *In: Symposium on User Interface Software and Technology, Eds*. Snowbird, Utah, United States, ACM Press, pp 10-20

KIRK, D., 2003. Graphics Architectures: The Dawn of Cinematic Computing. *In: Proceedings of the 1st international conference on Computer graphics and interactive techniques in Austalasia and South East Asia,* Adcock, M., Gwilt, I. AND Tsui, L. Y., *Eds*. Melbourne, Australia, ACM Press, pp 9

KRASNER, G. E. AND POPE, S. T., 1988. A Cookbook for Using the Model-View-Controller User Interface Paradigm in Smalltalk-80. *Journal of Object-Oriented Programming,* 1 (3), pp 26-49

KUHN, W. AND FRANK, A. U., 1991 A Formalization Of Metaphors And Image-Schemas In User Interfaces. *In:* Cognitive and Linguistic Aspects of Geographic Space, Mark, D. AND Frank, A. U. Eds. Kluwer, pp 419-434.

LAKOFF, G., 1992 The Contemporary Theory of Metaphor. *In:* Metaphor and Thought, Ortony, A. Eds. Cambridge University Press, pp 202-251.

LAKOFF, G. AND JOHNSON, M., 1980. *Metaphors We Live By,* University of Chicago Press, Chicago

LANDAY, J. A. AND MYERS, B. A., 2001. Sketching interfaces: Toward more human interface design. *Computer,* 34 (3), pp 56-64

LEVINE, J. R., MASON, T. AND BROWN, D., 1992. *Lex & Yacc,* 2. O'Reilly

LIMBOURG, Q., PRIBEANU, C. AND VANDERDONCKT, J., 2001. Towards Uniformed Task Models in a Model-Based Approach. *In: Interactive Systems. Design, Specification, and Verification, 8th International Workshop, DSV-IS 2001,* Johnson, C., *Eds*. Glasgow, Scotland, Springer, pp 164-182

LOHSE, J., 1991. A Cognitive Model for the Perception and Understanding of Graphs. *In: Human Factors in Computer Systems (CHI '91),* Robertson, S. P., Olson, G. M. AND Olson, J. S., *Eds*. New Orleans, The Association for Computing Machinery, Inc

Addison-Wesley Publishers Ltd, pp 137-144

LONG, J., 1997. Research and the Design of Human-Computer Interactions or 'What Happend to Validation? *In: People and Computers XII, Proceedings of HCI '97,* Thimbleby, H. W., O'Conaill, B. AND Thomas, P., *Eds*. Springer, pp 223-243

LOVGREN, J., 1994. How to Choose Good Metaphors. *Ieee Software,* 11 (3), pp 86-88

LUO, P., SZEKELY, P. AND NECHES, R., 1993. Management of Interface Design in Humanoid. *In: Conference on Human Factors and Computing Systems, Eds*. Amsterdam, The Netherlands, ACM Press, pp 107-114

LUYTEN, K. AND CONINX, K., 2001. An XML-Based Runtime User Interface Description Language for Mobile Computing Devices. *In: Interactive Systems: Design, Specification and Verification, 8th International Workshop, DSV-IS 2001,* Johnson, C., *Eds*. Glasgow, Scotland, Springer, pp 1-15

MAGLIO, P. AND MATLOCK, T., 1998. Metaphors we surf the web by. *In: Workshop on Personalized and Social Navigation in Information Space, Eds*. Stockholm, Sweden, pp 138-149

MARCUS, A., 1994. Metaphor mayhem: mismanaging expectation and surprise.interactions, 1 (1). January 1994

MARKOPOULOS, P., 1995. On The Expression Of Interaction Properties Within An Interactor Model. *In: Design, Specification and Verification of Interactive Systems '95,* Palanque, P. AND Bastide, R., *Eds*. Toulouse, Springer-Wien, pp 294-311

MARKOPOULOS, P., 1997. *A compositional model for the formal specification of user interface software*, Thesis Queen Mary and Westfield College, University of London.

MARKOPOULOS, P., 2001 Interactors: formal architectural models of user interface software. *In:* Encyclopedia of Microcomputers, Kent, A. AND Williams, J. G. Eds. Marcel Dekker, pp 203-235.

MARKOPOULOS, P. AND MARIJNISSEN, P., 2000. UML as a representation for interaction design. *In: Proceedings of OZCHI 2000`,* Paris, C., Ozkan, N., Howard, S. AND Lu, S., *Eds*. pp 240-249

MARKOPOULOS, P., PAPATZANIS, P., JOHNSON, P. AND ROWSON, J., 1998. Validating semi-formal specifications of interactors as design representations. *In: DSV-IS '98: 5th International Eurographics Workshop on Design, Specification and Verification of Interactive Systems,* Markopoulos, P. AND Johnson, J., *Eds*. Abingdon, UK, Springer, pp 102-116

MARKOPOULOS, P., SHRUBSOLE, P. AND DE VET, J., 1999. Refinement of the PAC model for the component-based design and specification of television based interfaces. *In: 6th International Eurographics Workshop on Design, Specification and Verification of Interactive Systems,* Duke, D. J. AND Puerta, A., *Eds*. Braga, Portugal, Springer, pp 117-132

MASSINK, M., DUKE, D. AND SMITH, S., 1999. Towards Hybrid Interface Specification for Virtual Environments. *In: Interactive Systems. Design, Specification, and Verification, 6th International Workshop, DSV-IS 1999, Eds*. Braga, Portugal, Springer-Verlag, pp 30-51

METROWERKS, 2003. CodeWarrior 8. *Metrowerks*

MICROSOFT, 2001. Visual C++ *Microsoft*

MILLARD, N., HOLE, L. AND CROWLE, S., 1998. The Motivational User Interface. *In: Human-Computer Interaction '98,* May, J., Siddiqi, J. AND Wilkinson, J., *Eds*. Sheffield, UK, pp 68-69

MORAN, T. P., 1980 A Framework for Studying Human-Computer Interaction. *In:* Methodology of Interaction, Guedj, E. R. A., Hagen, P. t., Hopgood, F. R., Tucker, H. AND Duce, D. A. Eds. North Holland Publishing Company, pp 293 - 301.

MORAN, T. P., 1983. Getting into a system: External-Internal Task mapping analysis. *In: Proceedings of CHI'83 Human Factors in Computing Systems, Eds*. New York: Association for Computing Machinery, pp 45-49

MUELLER, A., FORBRIG, P. AND CAP, C., 2001. Model-Based User Interface Design Using Markup Concepts. *In: Interactive Systems: Design, Specification, and Verification. The 8th International Workshop, DSV-IS 2001.*, Johnson, C., *Eds*. Glasgow, Scotland, Springer, pp 16-27

MUHR, T., 2002. Atlas.ti 4.2. *Scientific Software*

MYERS, B. A., 1995. User Interface Software Tools. *ACM Transactions on Computer-Human Interaction,* 2 (1), pp 64-103

NARDI, B. AND ZARMER, C., 1993. Beyond Models and Metaphors: Visual Formalisms in User Interface Design. *Journal of Visual Languages and Computing,* 4 pp 5-33

NAUR, P., 1984. Citation Classic - Revised Report on the Algorithmic Language Algol-60. *Current Contents/Physical Chemical & Earth Sciences,* (2), pp 16-16

NAVARRE, D., PALANQUE, P., PATERNÒ, F., SANTORO, C. AND BASTIDE, R., 2001. A Tool Suite for Integrating Task and System Models through Scenarios. *In: Interactive Systems. Design, Specification, and Verification, 8th International Workshop, DSV-IS 2001,* Johnson, C., *Eds*. Glasgow, Scotland, Springer-Verlag, pp 88-113

NORMAN, D. A. AND DRAPER, S. W., 1986 Cognitive Engineering. *In:* User Centred System Design, Norman, D. A. AND Draper, S. W. Eds. Lawrence Erlbaum Associates, pp 31 - 61.

PALANQUE, P., PATERNÒ, F., BASTIDE, R. AND MEZZANOTTE, M., 1996. Towards an integrated proposal for Interactive Systems design based on TLIM and ICO. *In: 3rd International Eurographics Workshop on Design, Specification, and Verification of Interactive Systems,* Bodart, F. AND Vanderdonckt, J., *Eds*. Namur, Belgium, Springer, pp 162-187

PATERN, F. AND MANCINI, C., 1999. Developing task models from informal scenarios. *In: Proceedings of ACM SIG CHI'99, Late Breaking Results, Eds*. ACM Press, pp 228-229

PATERNO', F., 2000. Model-based design of interactive applications.Intelligence, 11 (4).

PAYNE, S. J., 1984. Task-Action Grammars. *In: Human-Computer Interaction - INTERACT '84,* Shackel, B., *Eds*. Elsevier Science, pp 527-532

PENTLAND, A., 2000. Perceptual intelligence. *Communications of the Acm,* 43 (3), pp 35-44

PETERSON, J. L., 1981. *Petri Net THeory and the Modeling of Systems,* Prentice Hall PTR

PETZOLD, C., 1999. *Programming Windows, The Definitive Guide to the Win32 API,* 5th. Microsoft Press

PIDGEON, N., 1996 Grounded theory: theoretical background. *In:* Handbook of Qualitative Research Methods for Psychology and the Social Sciences, Richardson, J. Eds. Kogan Page Ltd, pp 75-85.

PREECE, J., ROGERS, Y., SHARP, H., BENYON, D., HOLLAND, S. AND CAREY, T., 1994. *Human-Computer Interaction,* Addison-Wesley

PREECE, J. AND ROMBACH, H. D., 1994. A Taxonomy for Combining Software Engineering and Human-Computer Interaction Measurement Approaches - Towards a Common Framework. *International Journal of Human-Computer Studies,* 41 (4), pp 553-583

PRIBEANU, C., LIMBOURG, Q. AND VANDERDONCKT, J., 2001. Task Modelling for Context-Sensitive User Interfaces. *In: Interactive Systems. Design, Specification, and Verification, 8th International Workshop, DSV-IS 2001,* Johnson, C., *Eds*. Glasgow, Scotland, Springer, pp 49-68

PUERTA, A., 1996 The Mecano Project: Comprehensive and Integrated Support for Model-Based Interface Development. *In:* Computer-Aided Design of User Interfaces, Vanderdonckt, J. Eds. Presse Universitaires de Namur, pp 19-25.

PUERTA, A., 1997. A Model-Based Interface Development Environment.IEEE Software, (July/August).

PUERTA, A., CHENG, E., OU, T. AND MIN, J., 1999. MOBILE: User-Centered Interface Building. *In: ACM Conference on Human Factors in Computing Systems, Eds*. Pittsburgh, ACM, pp 426-433

PUERTA, A. AND EISENSTEIN, J., 1999. Towards a general computational framework for model-based interface development systems. *Knowledge-Based Systems,* 12 (8), pp 433-442

PUERTA, A., ERIKSSON, H., GENNARI, J. AND MUSEN, M., 1994. Beyond Data Models for Automated User Interface Generation. *In: Proceedings of HCI '94: People and Computers IX,* Cockton, G., Draper, S. W. AND Weir, G. R. S., *Eds*. Glasgow, Cambridge University Press, pp 353-366

QSR, 2002. N6 6. *QSR*

RAJAGOPALA, M. G., HSIEH, S., SOTELINO, E. D. AND WHITE, D. W., 1997. MUIApp: an object-oriented graphical user interface application framework. *Engineering Computations,* 14 (3), pp 256-280

RICHARDS, I. A., 1936. *The Philosophy of Rhetoric,* Oxford University Press

RICHARDSON, J., ORMEROD, T. C. AND SHEPHERD, A., 1998. The role of task analysis in capturing requirements for interface design. *Interacting with Computers,* 9 (4), pp 367-384

RODRIGUEZ, F. G. AND SCAPIN, D. L., 1997. Editing MAD* task descriptions for specifying user interfaces, at both semantic and presentation levels. *In: 4th International Eurographics Workshop on Design, Specification, and Verification of Interactive Systems,* Harrison, M. D. AND Torres, J. C., *Eds*. Granada, Spain, Springer, pp 193-208

ROSSON, M. B., 1999. Integrating development of task and object models. *Communications of the ACM,* 42 (1), pp 49-56

SAGE, M. AND JOHNSON, C., 1997a. Interacting with Haggis: Implementing Agent Based Specifications in a Functional Style. *In: Interact 1997,* Howard, S., Hammond, J. AND Lindgaard, G., *Eds*. Sydney, Australia, Chapman and Hall, pp 126-133

SAGE, M. AND JOHNSON, C., 1997b. Interactors and Haggis: Executable specifications for interactive systems. *In: 4th International Eurographics Workshop on Design, Specification, and Verification of Interactive Systems,* Harrison, M. D. AND Torres, J. C., *Eds*. Granada, Spain, Springer, pp 93-108

SAGE, M. AND JOHNSON, C., 1998. Pragmatic Formal Design: A Case Study in Integrating Formal Methods into the HCI Development Cycle. *In: DSV-IS '98: 5th International Eurographics Workshop on Design, Specification and Verification of Interactive Systems,* Markopoulos, P. AND Johnson, P., *Eds*. Abingdon, UK, Springer-Verlag, pp 134-154

SAVIDIS, A., STEPHANIDIS, C. AND AKOUMIANAKIS, D., 1998. Unifying Toolkit Programming Layers: a Multi-purpose Toolkit Integration Module. *In: Proceedings of the 5th Eurographics Workshop on Design, Specification and Verification of Interactive Systems,* Markopoulos, P. AND Johnson, P., *Eds*. Abingdon, UK, Springer, pp 177-192

SCAPIN, D. L. AND PIERRET-GOLBREICH, C., 1989. Toward a method for task description: MAD. *In: Proceedings of Work with Display Units 89,* Berlinquet, L. AND Berthelette, D., *Eds*. Elsevier Science, pp 27-34

SCHNEIDER, K. A. AND CORDY, J. R., 2001. Abstract User Interfaces: A Model and Notation to Support Plasticity in Interactive Systems. *In: Interactive Systems: Design, Specification, and Verification, the 8th International Workshop, DSV-IS 2001,* Johnson, C., *Eds*. Glasgow, Scotland, Springer, pp 28-48

SCHREIBER, S., 1994. The BOSS System: Coupling Visual Programming with Model Based Interface Design. *In: Eurographics Workshop Design, Specification, and Verification of Interactive Systems,* Paternò, F., *Eds*. Carrara, Italy, Springer-Verlag, pp 161-179

SHNEIDERMAN, B., 1983. Direct manipulation: A step beyond programming languages. *IEEE Computer,* 16 (8), pp 57-69

SHNEIDERMAN, B. AND MAYER, R., 1979. Syntactic/Semantic Interactions in Programmer Behaviour: A Model and Experimental Results. *International Journal of Computer Information Sciences,* 8 (3), pp 219-239

SMALL, D., 1996. Navigating large bodies of text. *IBM Systems Journal,* 35 (3 & 4), pp 515-525

SMITH, D. C., IRBY, C., KIMBALL, R., VERPLANK, B. AND HARSLEM, E., 1982. Designing the Star User Interface. *Byte,* 7 (4), pp 242-282

SMITH, S. P., DUKE, D. J. AND WILLANS, J. S., 2000. Designing world objects for usable virtual environments. *In: DSV-IS 2000: 7th International Workshop on Design, Specification and Verification of Interactive Systems,* Palanque, P. AND Paternò, F., *Eds*. Limerick, Ireland, Springer, pp 309-319

SOMMERVILLE, I., 2001. *Software Engineering,* 6th. Addison-Wesley

SPENCE, R., 2001. *Information Visualization,* ACM Press

SPIVEY, J. M., 1989. *The Z notation: a reference manual,* Upper Saddle River, NJ, Prentice-Hall, Inc

STAMMERS, R. B., CAREY, M. S. AND ASTLEY, J. A., 1990 Task Analysis. *In:* Evaluation of Human Work: A practical ergonomics methodology, Wilson, J. R. AND Corlett, E. N. Eds. pp 134-160.

STARY, C., VIDAKIS, N., MOHACSI, S. AND NAGELHOLZ, M., 1997. Workflow-Oriented Prototyping for the Development of Interactive Software. *In: COMPSAC '97 - 21st International Computer Software and Applications Conference, Eds*. IEEE, pp 530-535

STORRS, G., 1995. The Notion of Task in Human-Computer Interaction. *In: People and Computers X (HCI'95),* Kirby, M. A. R., Dix, A. J. AND Finlay, J. E., *Eds*. Huddersfield, Cambridge University Press, pp 357-365

SUTCLIFFE, A., 2000. On the effective use and reuse of HCI knowledge. *ACM Transactions on Computer-Human Interaction,* 7 (2), pp 197-221

TOOK, R., 1992 The Active Medium: A Conceptual and Practical Architecture for Direct Manipulation. *In:* Building Interactive Systems: Architecture and Tools (Workshops in Computing), Gray, P. AND Took, R. Eds. Springer-Verlag, pp 6-21.

TORRES, J. C., GEA, F. L., GUITIERREZ, M., CABRERA, M. AND RODRIGUEZ, M., 1996. GRALPLA: An Algebraic Specification Language For Interactive Graphic Systems. *In: 3rd International Eurographics Workshop on Design, Specification, and Verification of Interactive Systems,* Bodart, F. AND Vanderdonckt, J., *Eds*. Namur, Belgium, Springer, pp 272-291

TRÆTTEBERG, H., 1998. Modelling direct manipulation with Referent Statecharts. *In: DSV-IS '98: 5th International Eurographics Workshop on Design, Specification and Verification of Interactive Systems,* Markopoulos, P. AND Johnson, P., *Eds*. Abingdon, UK, Spinger, pp 278-292

VAN DANTZICH, M., GOROKHOVSKY, V. AND ROBERTSON, G., 1999. Application redirection: hosting Windows applications in 3D. *In: Proceedings of the 1999 workshop on new paradigms in information visualization and manipulation in conjunction with the eighth ACM internation conference on Information and knowledge management, Eds*. Kansas City, Missouri, USA, ACM Press, pp 87-91

VAN DER VEER, G. C. AND VAN WELIE, M., 1999. Groupware Task Analysis. *In: Tutorial Notes for the CHI99 workshop "Task Analysis Meets Prototyping: Towards seamless UI Development", Eds.* Pittsburgh, Pennsylvania, USA, ACM, pp 1-15

VAN WELIE, M., VAN DER VEER, G. C. AND ELIËNS, A., 1998. An Ontology for Task World Models. *In: Proceedings of the Eurographics Workshop on Design, Specification, and Verification of Interactive Systems '98,* Markopoulos, P. AND Johnson, P., *Eds*. Abingdon, UK, Springer, pp 57-70

VANDERDONCKT, J. AND BERQUIN, P., 1999. Towards a Very Large Model-Based Approach for User Interface Development. *In: Proceedings of the 1st International Workshop of User Interface to Data Intensive Systems '99,* Paton, N. W. AND Griffiths, T., *Eds*. Edinburgh, UK, IEEE Computer Society Press, pp 76-85

VILLER, S. AND SOMMERVILLE, I., 1999. Coherence: An approach to representing ethnographic analyses in systems design. *Human-Computer Interaction,* 14 (1-2), pp 9-41

VISUALEDGE, 1997. UIM/X 3.0 3. *Visual Edge Software*

WALLACE, M. D. AND ANDERSON, T. J., 1993. Approaches to Interface Design. *Interacting with Computers,* 5 (3), pp 259-278

WASSERMAN, A. I., 1985. Extending State Transition Diagrams for the Specification of Human-Computer Interaction. *IEEE Transactions on Software Engineering,* SE-11 (8), pp 699-713

WHITEFIELD, A. AND HILL, B., 1994. Comparative-Analysis of Task-Analysis Products. *Interacting with Computers,* 6 (3), pp 289-309

WILLIAMS, P., 1992 Surface Interaction*: A paradigm for Object Communication. *In:* Building Interactive Systems: Architecture and Tools (Workshops in Computing), Gray, P. AND Took, R. Eds. Springer-Verlag, pp 23-33.

WINDSOR, P., 1990. An object-oriented framework for prototyping user interfaces. *In: Human-Computer Interaction - INTERACT '90,* Diaper, D., Gilmore, D., Cockton, G. AND Shackel, B., *Eds.* Elsevier Science, pp 309-314

WROBLEWSKI, D. A., 1991 The Construction of Human-Computer Interfaces Considered as a Craft. *In:* Taking Software Design Seriously: Practical Techniques for Human-Computer Interaction Design, Karat, J. Eds. Academic Press, pp 1-19.

YOURDON, E., 1994. *Object-Oriented Systems Design: An Integrated Approach,* Prentice-Hall International, Inc.

ZAJICEK, M. P. AND WINDSOR, R., 1995. Using Mixed Metaphors to Enhance the usability of an electronic multimedia document. *In: IEE Colloquium 'Human-Computer Interface Design for Multimedia Electronic Books, Eds.* Washington, pp 21-27