ISML: An Interface Specification Meta-Language

Simon Crowle[1] and Linda Hole[2]

[1] Bournemouth University, Royal London House,
Christchurch Road, Bournemouth Dorset,UK
e-mail:scrowle@bournemouth.ac.uk,
WWW home page: http://dec.bournemouth.ac.uk/staff/scrowle/
[2] Bournemouth University, Poole House, Talbot Campus,
Poole, Dorset, UK
e-mail:lhole@bournemouth.ac.uk,
WWW home page: http://dec.bournemouth.ac.uk/staff/lhole/index.htm

Abstract. In this paper we present an abstract metaphor model situated within a model-based user interface framework. The inclusion of metaphors in graphical user interfaces is a well established, but mostly craft-based strategy to design. A substantial body of notations and tools can be found within the model-based user interface design literature, however an explicit treatment of metaphor and its mappings to other design views has yet to be addressed. We introduce the Interface Specification Meta-Language (ISML) framework and demonstrate its use in comparing the semantic and syntactic features of an interactive system. Challenges facing this research are outlined and further work proposed.

1 Introduction

Xerox's Star system [33] is the most famous, early example of the application of metaphors in the design of a graphical user interface (GUI). Later generations of this 'desktop metaphor' are found in many of today's commercial personal computer systems with little apparent modification other than superficial changes in presentation. Metaphors are said to enhance a user's understanding and manipulation of the system state through the provision of interactive entities that mimic the appearance and behaviour of real-world objects [31]. In this way users are able to achieve their goals without having to re-cast their problem to fit the domain native to the computer system.

To date, human-computer interaction (HCI) practitioners enjoy relatively limited support for the development of novel user interface metaphors. Those who wish to develop interfaces in this way may turn to psychological accounts of metaphor [22][21][12] or a few highly abstract, mathematical models [18][20]. Whilst providing useful insights into the nature of metaphor generally, this work has relatively little to offer in terms of advice for interface development. For this, designers may refer to a number of case-studies found in the literature providing quantitative and qualitative evidence for the application of specific metaphorical 'devices' to design problems (see [35][25][14][5]). More general guidelines for

metaphor design can also be found in [13][23]. Arguably, the most formalized approach to design can be found in Alty's framework [3][4]. Here, a six stage process is outlined in which the intended system functionality and work environment is examined for potential metaphor 'vehicles' - these are subsequently evaluated against a set of guidelines. Although approaches such as these outline useful methods for developing user interface metaphors, they remain strongly craft-based and not integrated with contemporary HCI design methods.

An emerging field in HCI, model-based user interface design (MB-UID) methods offer the designer a variety of frameworks, notations and tools with which to integrate a number of design perspectives. In a review of contemporary MB-UID research, da Silva [10] posits three major advantages of the approach:

1. Higher levels of abstraction not provided by traditional UI development tools
2. Systematic development of UI in which models may be refined and reused
3. Provision of an infrastructure that provides automatic support for design and implementation

Many notations and tools support abstractions for various design views including devices [26][19], graphical components and direct manipulation [16][9][27], task models [34][30] and domain modelling [15][11]. However, advances in user interface technology present the MB-UID community with new challenges, most particularly with respect to designing for multiple platforms and task contexts [7][32]. Recent advances in interface technologies have resulted in multiple hardware and software platforms each of which implement varying degrees of technical capability. Currently, model-based methods do not adequately address this complexity or exploit the potential found in the diversity of these technologies [28].

To address this problem, it is necessary to expand the design views currently addressed in MB-UID beyond that of the ubiquitous desktop user interface. Breaking this mould is an opportunity to reconsider the use of metaphors in design and their role in the generation of new model-based methods. We argue that the abstraction of a metaphor model can provide a useful mechanism for carrying user interface design solutions between platforms since:

1. Metaphors are frequently implied in many user interface designs
2. A metaphor can be 'carried' from platform to platform
3. High fidelity interface technologies invite metaphor development

It would be difficult to find popular commercial software that did not make use of metaphorical concepts such as *the desktop*, *files*, *folders*, *cutting and pasting*, *dragging and dropping* and so on. However, features such as these are frequently an implicit part of the implementation solution, and are not explicitly specified in model-based design. Consequently, the underlying concepts and mechanics of the metaphor are hidden behind a higher level of component-based abstraction. The numerous variants of the desktop metaphor that can be found on different hardware and software platforms illustrate this phenomenon: the 'look and feel' of each system might vary considerably, but the basic concepts remain more or less constant.

The pace of change in user interface technologies is rapid, however. As the increasing availability of high performance multimedia, 3D graphic and mobile computing hardware grows, so too does the potential for entirely new forms of interaction. The application of these technologies has already been demonstrated in the field of information visualization [8]. Delivery of graphically rich and interactively complex environments is a common feature in the computer games industry where such features are highly attractive to users. Such systems present the user with objects, behaviours and interactions that have high congruities with the physical world.

The increasing diversity of the means with which to interact with computing devices, coupled with the expansion of graphically and semantically rich environments is therefore an enormous challenge to the HCI community. It is unlikely that existing abstractions of user interface components will be sufficient to express the concepts for the next generation GUI designs. One possible way forward is to examine how an explicit metaphor model might be used to extend design concepts in the MB-UID community.

In the proceeding sections, we introduce a high-level framework that supports metaphor models (section 2), followed by a more detailed look at its five principal layers (section 3). Finally, we examine the utility of this approach with respect to mapping metaphorical design concepts to multi-target platforms (section 4).

2 An overview of ISML

In this section, we provide a high-level overview of the Interface specification meta-language (ISML). From the outset, ISML was developed with the intention that metaphors (shared concepts between the user and the computer) be made explicit in design. Further, this mechanism in itself has no absolute manifestation with respect to its implemented appearance and operation at the user interface. ISML de-couples the metaphor model from any particular implementation, and expresses mappings between the concepts shared between the user and the system. For such a model to become useful, ISML provides a framework that supports mappings between both user-oriented models (such as task descriptions) and software architecture concerns (interactor definitions). The ISML framework composites these concepts within five layers, using a variety of mappings to link them together (see figure 1). Each of these layers is supported by computational formalisms similar to those already found within the literature (including communicating objects [6]; state modelling [1]; abstract-to-concrete mappings [24]; event modelling [6]; and task models [30]). The layers in the model inherit or implement abstractions from one another using this shared computational basis.

Devices are simple abstractions of user interface input/output hardware used to model entities such as the mouse, keyboard and graphics adapter. Logical abstractions of user input and output objects are specified as *components*, these map to devices for implementing their function. *Meta-objects* define the underlying metaphor model, expressed using rules governing the semantics and syntax

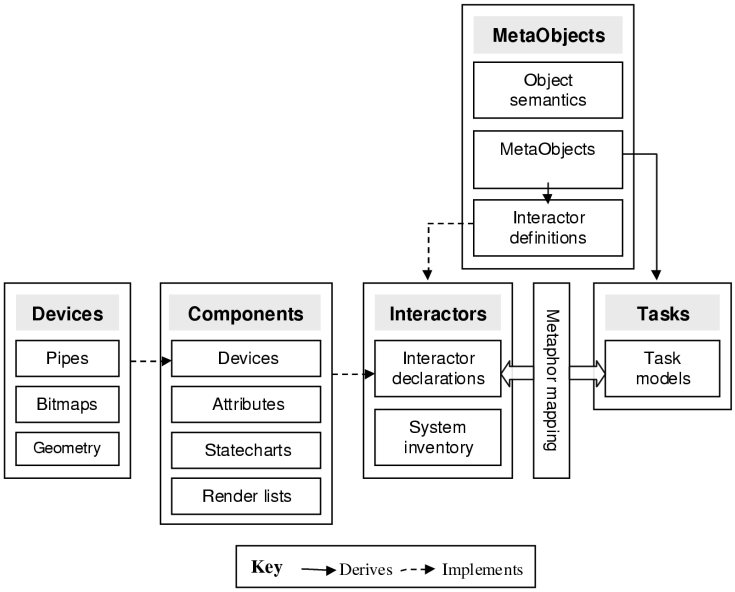

Fig. 1. ISML framework

of inter-communicating objects. This abstraction is also used for part of the task model, and forms the basis for the 'concrete' implementation of the metaphor. There are two advantages to this approach. First, the translation between a user's task model and its execution through a metaphorical medium is expressed using the same language. Secondly, since the implementation of the concrete user interface is based on the meta-object layer, it is possible to show how components express metaphor.

Interactor definitions use meta-objects as a basis for a specific design solution using just such a mapping - this is accomplished through the mapping of components to interactor 'display parts' (similar to those found in MVC or PAC, see [17]). The *task* layer combines meta-object definitions of objects and actions with a simple, hierarchical decomposition of tasks, similar to the approaches found in [30].

Finally, the intersection of meta-objects in use in both interactor and task models is described in the *metaphor mapping* definition. This definition specifies potential analogies between the execution of actions on objects in the task model and their equivalencies in the metaphor model, actualized through interactors. In the proceeding sections, we will examine each layer in more detail and then examine how an underlying meta-object model might be implemented using two different intraction styles.

3 The ISML layers

The framework ISML uses a Backus-Naur Form based grammar to specify the user interface, presented here in the XML language. Since space does not permit a detailed examination of the ISML, only brief outline is provided - for a more detailed coverage, readers are directed to the main author's web site. For the sake of brevity, most of the XML structures used to express the ISML framework will be presented graphically, using Altova's *XML Spy* graphical notation [2].

3.1 Attributes, logic and state models

Throughout the ISML framework, attributes, states and logic are used extensively. Attributes have a required name, type and access. Basic types of ISML attribute include common programming data types of *boolean*, *integer*, *float* and *string*; attributes may also be of type *set*. 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 (at present, expressed using the 'C' grammar) for the expression of mathematical formula, conditional logic tests.

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

3.2 Devices

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. A definition of a simple mouse device, for example, might include a boolean and two integers representing the hardware button state and a motion vector. Presently, these definitions only provide a rudimentary lexicon for input/output devices (supporting only pointing devices, keyboards and 2D display devices) but in principle could be extended in the future.

3.3 Components

A component definition specifies the *presentation* features of a 'concrete' interface object - it may contain attributes such as 'height' or 'width' or 'font name'. State models are also allowable in component definitions and may be used to poll devices for new input information or render graphics (such as displaying images of a button in an armed or 'unarmed' state).

Rendering is managed through the specification of 'render lists' of which only one may be active at any one time (this is called the 'render focus'). Each render list is a collection of functional calls to the devices associated with the component.

3.4 Meta-objects

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. This abstraction can be logically sub-divided into two parts: i)*syntax, semantics and meta-objects* and ii) *meta-interactor* definitions.

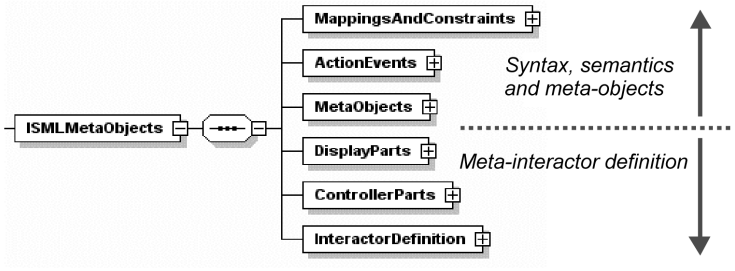

Fig. 2. ISML Meta-object abstraction

Syntax and semantics At the beginning of any meta-object abstraction (see figure 2), syntactical and semantic rules must be described such that relationships between meta-objects and communications between them can be specified. 'Action-events' are syntactic descriptions of all possible communications between objects. For example, one object may request 'ownership' (see below) of another object by invoking an action-event:

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

Here, the parameter event sender identifies the calling object and the x,y parameters specify a method of object selection, based on the principle that objects occupy space in at least two dimensions.

A concept such as *dragging* is expressed using a *mapping-constraint* expression that may contain morphisms and constraints based on mathematical and logical arguments. Both specify source and target attributes upon which to operate; in the example below the *yPosition* of the target object is evaluated within a range specified by *min* and *max* attributes in the *source*:

```
 <Constrain>
   <Target>yPosition</Target>
   <Source>
      <AttrRef>min</AttrRef>
      <AttrRef>max</AttrRef>
   </Source>
   <Predicate>
   <Statement>
      ( ( target > min) && (target < max) )
   </Statement>
   </Predicate>
</Constrain>
```

Note: XML characters are not 'escaped' for clarity

Whilst in operation within a meta-object, mappings are continuously enforced whilst constraints may be tested for satisfaction by an internal state model.

Meta-object definition Metaphorical objects specified in the ISML framework are defined as meta-object types, the abstract parts of which are comprised of attributes and state models. The semantics section determines the object's use of previously defined *action-events* and *mapping-constraints* which may be classed as either *affective*, *effective*, *both affective and effective* or *exclusively affects*. In this way, each meta-object is determined as being capable of enforcing, or being subject to, the syntax and semantics of the metaphor abstraction. Any object enforcing a mapping-constraint maintains a set that holds references to the objects it affects. For every *action-event* an object is subject to, a handler must be defined within which a response to an action takes place. Responses may include set operations, tests or procedural logic. Operations on local mapping-constraint sets include emptying, adding and subtracting, or the ability to invoke actions on the members of that set. Tests include checking an object's existence, state, class type, affective and effective capabilities or satisfaction of constraint.

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 parts and binding them with a meta-object. Display parts are subsequently mapped to component abstractions in later interactor declarations (see section 4). This allows derived interactor classes the ability to receive input from or render to multiple components. It may be desirable for interactors to temporarily suspend their behaviour according to the state of the underlying system. For this reason, controller definitions list a collection of mapping-constraints or action-events that can be turned on or off as appropriate.

3.5 Interactors

The interactor abstraction realizes the underlying metaphor through refinement and mapping to previously defined components. Derived from a meta-interactor (and consequently, a meta-object abstraction) the interactor may also include attributes, state models, handle over-loaded or additional action-events of its own and make calls to underlying system functionality.

Display bindings map the display parts associated with the interactor to components. Subsequent attribute bindings link the attributes found within the interactor (or parent meta-object) to attributes implemented by the component. Rendering of the mapped components is achieved through an explicit render directive. Interactors that shared the same display type may also re-target the destination of the rendering with one another.

3.6 Tasks

The ISML 'task world' re-uses the basic meta-object abstractions (mapping-constraints, action-events and meta-objects) to describe extant task related entities and their role within a hierarchical description of tasks. Presently, an ISML task model is a very simple hierarchy of linked nodes, each of which may be abstract or action-based. Abstract nodes serve to label higher order task plans or goals. Nodes that contain actions refer to a source object performing some afferent action upon a target (based on the meta-object definitions). Sequences of actions are specified in node lists (including a parent node, excepting the 'root' sequence), either serializing nodes or specifying a choice using the *enable* and *or* connectives respectively. A node may also specify an *iterate* condition for any action. In this case, a task node is said to continuously repeat until either a mapping-constraint test (see section 3.4) or logical test of an object's attributes evaluates to true.

3.7 Metaphor mapping

The final part of the ISML specification builds mappings from the objects and actions defined in the task model to the interactors and interactions at the user interface. Object maps may be simple name-space mappings indicating interactor equivalents of some task object, or they be may refined further by including mappings of specific attributes or states. Action analogies are drawn from the linking of a task object and action to one or many interactor and action couplings. Since interactors are derived from the underlying meta-object model, it is therefore possible to show how a particular user interface design does (or does not) represent and enact an underlying user interface metaphor.

4 Discussion

The ISML framework extends the application of existing formalisms used in model-based user interface design through the introduction of a metaphor abstraction layer. Metaphorical mappings are expressed in two important ways:

1. Interactor designs are built on top of an underlying metaphor
2. User tasks can be translated to physical interactions that enact the metaphor design.

In the following sections, we outline how the ISML framework can map different physical implementations to a common design, and also show to what extent a specific implementation allows direct engagement with underlying metaphorical concepts.

4.1 Abstract metaphor as a basis for concrete design

Unlike many other MB-UID technologies, ISML does not resolve to any particular component-based implementation (such as Microsoft's Foundation Class or Sun's Java *Swing* classes). The definition of components based on abstract devices affords an ISML specifier some useful freedoms. For example, within a metaphorical environment, it may be desirable to describe a pointing entity (such as a hand). The implementation of this object may be realized in many different ways, depending on both the devices and components available for use. The display mappings of the interactor expression of the hand (the screen cursor)

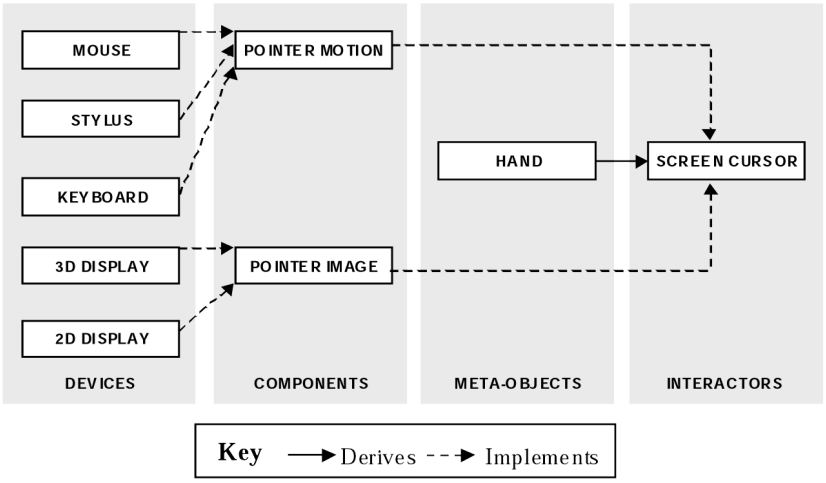

Fig. 3. Implementing metaphors

may have two parts; one for input, another for output. Input to the cursor may be received from a component that describes relative motion as a vector. The actual device that describes that motion may be a mouse, or a graphics stylus or a keyboard, depending on the particular component's mapping. Similarly, the technology used to display the cursor image can be flexibly determined by the choice of pointer image component (and its particular mappings to output technologies). In both cases, the technologies used to implement the component are

decoupled from other design views, since it is the mapping of attributes exposed by the component that are used by the interactor (see figure 3). The de-coupling

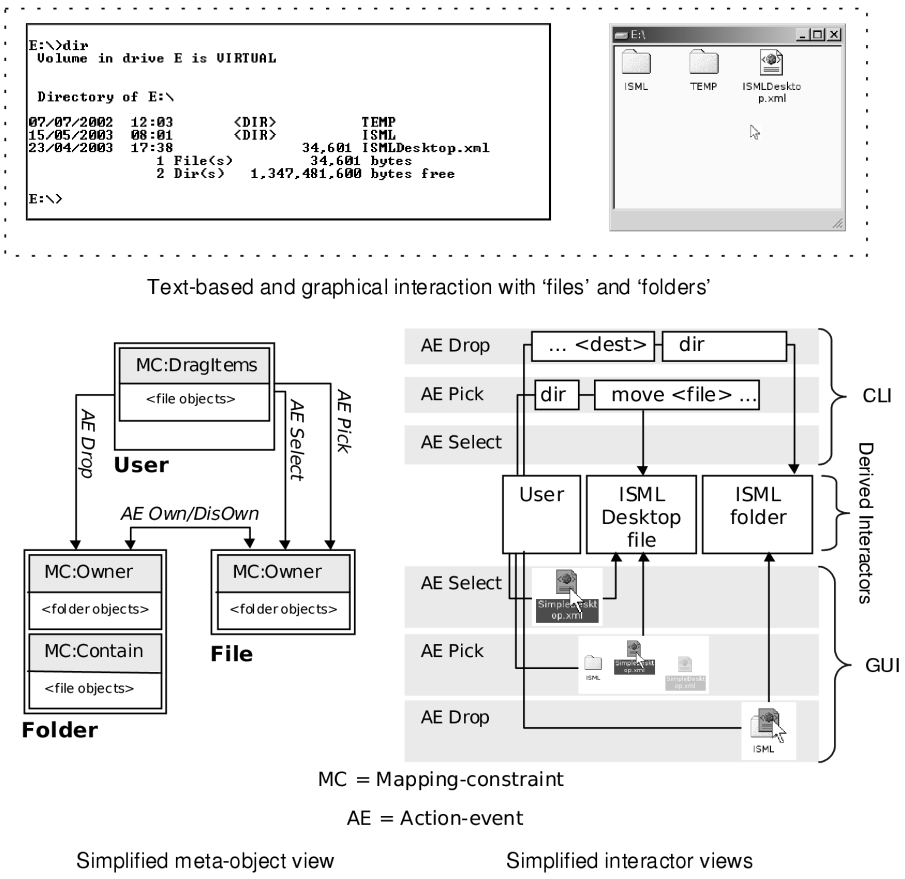

Fig. 4. Simplified ISML views of moving a file

of a common metaphor 'fragment' is illustrated in figure 4, in which a high-level view of the moving of a file named *ISMLDesktop.xml* into a folder called *ISML* is presented. Irrespective of either interactive solution, a fundamental analogical model of the nature of files and folders (or 'directories') must be understood by the user for the task to be completed successfully. The very least that can be said of this model is that files are entities that are contained by folders. In addition, a file may also be moved from one place to another. This action can be further split up into three stages: a) the selection of a specific file (action-event 'Select'), b) the removal of its presence from an existing folder (action-event 'Pick') and c) its subsequent appearance in another (action-event 'Drop'). The movement

of the file object begins with a *pick* action and its motion is expressed as a mapping-constraint (MC: DragItems) in which the spatial properties of the file are translated by the pointing entity, controlled by the user. Movement ends on the execution of a *drop* action, at which point the file object is removed from the mapping-constraint set. Two additional action-events ('Disown' and 'own') are used to a) execute the removal of the reference to the file object maintained by the source folder (stored in the mapping-constraint set 'Contain') and b) to add a new reference to the target folder.

Implementing the meta-model shown in figure 4 requires the derivation of each meta-object to an interactor that is coupled to actual user interface components. In the case of the command line interface (CLI), some of the properties of the file and folder objects (such as name and type) are rendered as lines of text whilst input from the user is polled from the keyboard and sent to the display in the same way. For the GUI example both file and folder objects have images and co-ordinates in space that can be modified. The user communicates via a mouse and a cursor display is modified by vectors polled by the mapped mouse device.

In addition to clear qualitative differences in appearance, the implementation of the underlying actions can also be compared. The 'user' interactor for the CLI solution must interpret commands (sequences of keystrokes, ending in the return key). Here, the user does not use the input device to enact actions on objects directly, but instead must allow the system to act as a proxy. The interactor object dealing with user interaction in this case must internally translate the parameters passed by the user into equivalent action-events. By contrast, the GUI implementation maps direct, spatial actions executed by the user through the mouse to their equivalencies in the meta-object model.

4.2 Enacting tasks at the physical interface

The de-coupling of task, interactor and metaphor views that the ISML framework provides allows the designer to more clearly specify the translation of user goals into interactions with the system. In Norman's approximate theory of action [29], users are said to continually execute a cycle of action specification and output evaluation. To accomplish a goal, users must formulate intentions for changing the system state that are expressed as actions, and executed at the user interface. Any changes to the system are reflected at the interface, interpreted by the user and evaluated with respect to their progress towards their goal (see figure 5). The 'articulatory' and 'semantic' distances that a user must traverse, argues Norman, are reduced if the user interface reflects the problem domain with which the user works.

Meta-object abstractions form the basis for both the eventual user interface design and the task model; this allows the designer to consider the semantic and articulatory distances a user must travel in order to complete their goals. If the mappings expressed between task and interactors cannot be demonstrated to map back to the underlying metaphor then, arguably, the user has a greater

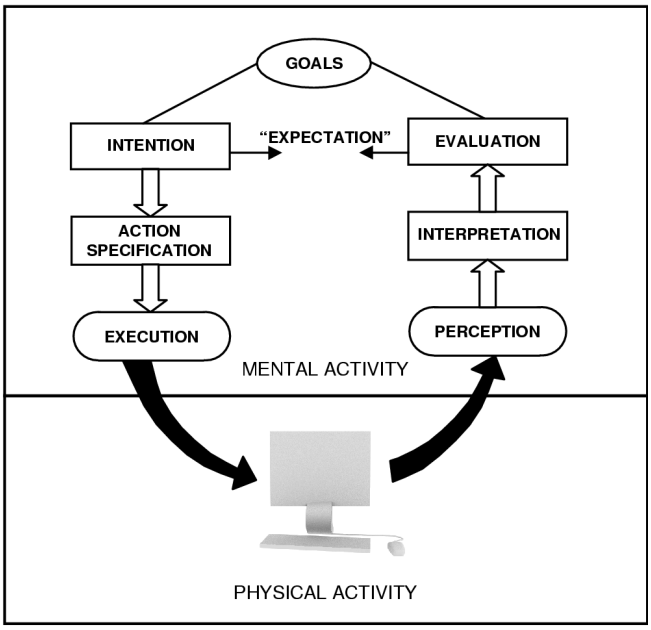

Fig. 5. Norman's model [29]

semantic distance to travel. In the example described above, the CLI implementation increases this semantic distance since operations executed by the user do not have the spatial properties or direct engagement of action that the natural movement of an object from one place to another entails.

A comparison of the interactions required for both the CLI and GUI implementations also show differences in 'syntactic' distance. In the former case, commands must be recalled by the user (these are related to, but not an explicit part of, the underlying meta-object model) and subsequently translated by the interactor abstraction to affect action-events. Furthermore, the use of the 'dir' command is required to provide feedback on the state of the objects being manipulated. Here, a user must recall the functionality of the system to check the success of their task rather than utilize the immediate, spatial features of the GUI. The relationship between system functionality (S) and metaphorical interface design features (M) is described as intersection by Alty et al. [4]. Each of the four conditions can be shown as continuous or discontinuous mappings within the ISML framework:

1. S+M+: those features of the system that map directly to the metaphor. In ISML, this means the underlying metaphor is directly represented and enacted by derived interactors (that also make calls to system functionality), which in turn can be mapped to task actions and objects.
2. S+M-: system functionality that does not exist in the metaphor features of the interface. Within an ISML specification, this means that the interac-

tor layer must either translate the implemented interaction to meta-object abstractions (if they exist) or make calls directly to system functionality.
3. S-M+: here, features of the metaphor model do not map to system functionality. In this case, mappings from the meta-object to interactor layers may not exist at all, or if they, do not actually result in calls to the underlying system functionality.
4. S-M-: neither system nor metaphor features exist. In this case, no mappings can be made from the task model to the interactor layer to support the completion of a task.

Through explicitly modelling a metaphor abstraction and providing mappings to interactors and tasks, it is therefore possible to identify aspects of a user interface design that support the user's work domain as well as those that do not.

5 Conclusion

In this paper we have introduced the ISML framework and examined its use in explicitly specifying a metaphorical concept and its mappings to more than one possible implementation. ISML was developed to make metaphors an explicit part of model-based design, de-coupling the metaphor model from specific implementation details. The framework uses a Backus-Naur Form based grammar to specify the user interface, interactor-based implementations of designs are built on top of an underlying metaphor. User tasks can be examined as translations to physical interactions that enact the metaphor design. Currently, the ISML XML schema and tool-based support is still in the very early stages of development. An XSLT-based transformation that will allow semi-automatic translation to executable code (supported by an ISML run-time kernel) is in progress. We hope to report on further work, including a case-study using ISML, in the near future.

References

1. Accot, J., S. Chatty, et al.: A Formal Description of Low Level Interaction and its Application to Multimodal Interactive Systems. 3rd International Eurographics Workshop on Design, Specification, and Verification of Interactive Systems, Namur, Belgium, Springer-Verlag (1996)
2. XML Spy, Altova. http://www.altova.com/ (2003)
3. Alty, J. L. and R. P. Knott.: Metaphor and human computer interaction: a model based approach. Proceedings of Computation for Metaphors, Analogy and Agents: An International Workshop. (1998)
4. Alty, J. L., R. P. Knott, et al.: A framework for engineering metaphor at the user interface. Interacting with Computers 13(2) (2000) 301-322
5. Ark, W., D. C. Dryer, et al.: Representation Matters: the Effect of 3D Objects and a Spatial Metaphor in a Graphical User Interface. Proceedings of HCI 98, the Conference on Human-Computer Interaction, Springer. (1998)

6. Bastide, R. and P. Palanque.: A visual and formal glue between application and interaction. Journal of Visual Languages and Computing 10(5) Academic Press. (1999) 481-507
7. Braubach, L., A. Pokahr, et al.: Using a Model-based Interface Construction Mechanism for Adaptable Agent User Interfaces. Proceedings of AAMAS Workshop 16 - Ubiquitous Agents on Embedded, Wearable, and Mobile Devices. (2002)
8. Card, S. K., J. D. Mackinlay, et al.: Readings in Information Visualization: Using Vision to Think. San Francisco, CA, Morgan Kaufmann Publishers. (1999)
9. Carr, D.: Interaction Object Graphs: An Executable Graphical Notation for Specifying User Interfaces. Formal Methods for Computer-Human Interaction. P. Palanque and F. Paterno', Springer-Verlag (1997) 141-156.
10. da Silva, P. P.: User interface declarative models and development environments: A survey. Interactive Systems. Design, Specification, and Verification, 8th International Workshop, DSV-IS 2001, Glasgow, Scotland, Springer-Verlag Berlin. (2001)
11. Eisenstein, J., J. Vanderdonckt, et al.: Applying model-based techniques to the development of UIs for mobile computers. International Conference on Intelligent User Interfaces archive Proceedings of the 6th international conference on Intelligent user interfaces, Santa Fe, New Mexico, United States, ACM Press. (2001)
12. Gentner, D., B. Bowdle, et al.: Metaphor is like analogy. The analogical mind: Perspectives from cognitive science. D. Gentner, K. J. Holyoak and B. N. Kokinov. Cambridge, MA, MIT Press. (2001) 199-253.
13. Gillan, D. J. and R. G. Bias.: Use and Abuse of Metaphor in Human-Computer Interaction. Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics, San Antonio. (1994)
14. Golovchinsky, G. and M. H. Chignell.: The newspaper as an information exploration metaphor. Information Processing and Management 33(5) Elsevier Science (1997) 663-683.
15. Griffiths, T., P. J. Barclay, et al.: Teallach: a model-based user interface development environment for object databases. Interacting with Computers 14(1) Elsevier Science (2001) 31-68.
16. Hartson, H. R., A. C. Siochi, et al.: The Uan - a User-Oriented Representation for Direct Manipulation Interface Designs. Acm Transactions on Information Systems 8(3) (1990) 181-203.
17. Hussey, A. and D. Carrington.: Comparing the MVC and PAC Architectures: a Formal Perspective. IEE Proceedings of Software Engineering 144(4): (1997) 224-236.
18. Indurkhya, B.: Constrained Semantic Transference - a Formal Theory of Metaphors. Synthese 68(3) (1986) 515-551.
19. Jacob, R. J. K., L. Deligiannidis, et al.: A Software Model and Specification Language for Non-WIMP User Interfaces. ACM Transactions on Computer-Human Interaction 6(1): (1999) 1-46.
20. Kuhn, W. and A. U. Frank.: A Formalization Of Metaphors And Image-Schemas In User Interfaces. Cognitive and Linguistic Aspects of Geographic Space. D. Mark and A. U. Frank. Technical University Vienna, Austria, Kluwer. (1991) 419-434.
21. Lakoff, G.: The Contemporary Theory of Metaphor. Metaphor and Thought. A. Ortony, Cambridge University Press (1992)
22. Lakoff, G. and M. Johnson.: Metaphors We Live By, University of Chicago Press, Chicago. (1980)
23. Lovgren, J.: How to Choose Good Metaphors. Ieee Software 11(3) (1994) 86-88.
24. Luyten, K. and Coninx,K. : An XML-Based Runtime User Interface Description Language for Mobile Computing Devices In: C. Johnson (Ed.): Interactive Systems:

Design, Specification, and Verification 8th International Workshop, DSV-IS 2001. Glasgow, Scotland, (2001) 1-15
25. Maglio, P. and T. Matlock.: Metaphors we surf the web by. Workshop on Personalized and Social Navigation in Information Space, Stockholm, Sweden. (1998)
26. Massink, M., D. Duke, et al.: Towards Hybrid Interface Specification for Virtual Environments. Interactive Systems. Design, Specification, and Verification, 6th International Workshop, DSV-IS 1999, Braga, Portugal, Springer-Verlag. (1999)
27. Navarre, D., P. Palanque, et al.: A Tool Suite for Integrating Task and System Models through Scenarios. Interactive Systems. Design, Specification, and Verification, 8th International Workshop, DSV-IS 2001, Glasgow, Scotland, Springer-Verlag. (2001)
28. Nilsson, E. G.: Combining Compound Conceptual User Interface Components with Modelling Patterns - a Promising Direction for Model-based Cross-platform Interface Development. Interactive Systems. Design, Specification, and Verification, 9th International Workshop, DSV-IS 2002, Rostock, Germany, Springer. (2002)
29. Norman, D. A. and S. W. Draper.: Cognitive Engineering. User Centred System Design. D. A. Norman and S. W. Draper, Lawrence Erlbaum Associates. (1986) 31 - 61.
30. Paterno', F. and C. Mancini.: Developing Task Models from Informal Scenarios. Proceedings ACM CHI'99, Pittsburgh, ACM Press. (1999)
31. Preece, J., Y. Rogers, et al.: Human-Computer Interaction, Addison-Wesley. (1994)
32. Pribeanu, C., Q. Limbourg, et al.: Task Modelling for Context-Sensitive User Interface. Interactive Systems. Design, Specification, and Verification, 8th International Workshop, DSV-IS 2001, Glasgow, Scotland, Springer. (2001)
33. Smith, D. C., C. Irby, et al.: Designing the Star User Interface. Byte 7(4) (1982) 242-282.
34. van der Veer, G. C. and M. van Welie.: Groupware Task Analysis. Tutorial Notes for the CHI99 workshop Task Analysis Meets Prototyping: Towards seamless UI Development. (1999)
35. Zajicek, M. P. and R. Windsor.: Using Mixed Metaphors to Enhance the usability of an electronic multimedia document. IEE Colloquium Human-Computer Interface Design for Multimedia Electronic Books, Washington. (1995)