# USING USER INTERFACE MODELS IN DESIGN

Hallvard Trætteberg[1]

*Dept. of Computer and Information Sciences (IDI), Norwegian University of Science and Technology (NTNU). Sem Sælands v. 7-9. 7491 Trondheim, Norway.*

*hal@idi.ntnu.no*

**Abstract** We introduce a framework for classifying user interface design representations, and argue that multiple representations are must be used in the design process, and that modelling languages must support the transition between them. We present languages for modelling domain, task and dialog and show how they provide increased support for design, through flexibility and integration. Design patterns based on model fragments from these languages are suggested as a design and engineering tool.

**Keywords:** user interface model, task model, dialog model, design pattern

## 1. INTRODUCTION

With the standardization of UML and its emergence as a de-facto industrial standard, modelling of systems and software artefacts seems to be accepted in the engineering community. User interface modelling and model-based user interface design has however, not reached the mainstream software developer [1]. Red Whale's MOBILE [2] cover several models but lack integration with software engineering tools, while database-oriented tools like Genera's Genova [3] have a more narrow focus on concrete interaction object selection and layout. It seems that lack of tool support for going from comprehensive visual model diagrams to executable code may be a reason for this difference. But very few UML diagramming tools provide full code-generation support, yet developers are using these tools to draw UML diagrams, e.g. for *communication* purposes. The lack of penetration of model-based user interface design methods may instead be due to pragmatic aspects of UI modelling languages, e.g. they are difficult to understand, inflexible and impractical to use and lack the appropriate integration with each other. Although formality in itself is important, it

1. This work is part of a Dr.ing study funded by The Research Council of Norway.

may be that the focus should be language *usability* rather than theory, if the goal is reaching the mainstream. A *uniform user interface modeling language* must *integrate* sub-languages for domain, task and dialog modelling, be *unconstraining* and *flexible*, i.e. provide support for expressing design ideas, suggestions and decisions throughout the design process.

In this paper we will first introduce a framework for classifying design representation. The framework provides a means for understanding how they are utilized in the design process, which in turn has implications for language design and usage. We then briefly present modelling languages for tasks and dialogs and discuss how our understanding affected their design. We then turn to how design knowledge can be represented in the same languages in the form of model-based user interface design patterns. Finally, we suggest how these modelling languages may be integrated with UML, to provide better integration with contemporary methods.

## 2. DESIGN PROCESS AND REPRESENTATIONS

To better understand how design representations are used and the role they play, we use a model of *organizational knowledge creation* as a starting point [4]. According to this theory, there must during product development, be a constant conversion of knowledge between *tacit* and *explicit* forms using the four *transfer modes* shown in Figure 1. Hence, to better support product development, the process and *design representations* used, must support these transfer modes. By "design representation" we mean any design relevant knowledge that is externalized in a *human-readable* medium, whether textual, graphic or multimedia.

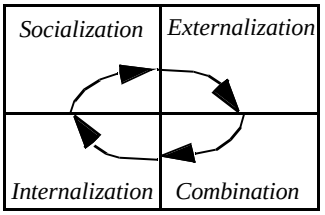


*Figure 1.* Nonaka's four modes of knowledge transfer.

To support *socialization*, the representation must support social interaction, for which flexible and informal representations are best suited. In the process of *externalization* and further *formalization*, the representation needs to shift from informal to formal, and the medium or language used will need to change. Throughout the development process there will be a drift towards increased formality, as the design is committed to and the need for transfer to "outsiders" increases [5]. Systems like SILK [6] and the Cocktail Napkin [7] target this transfer mode. During *combination* new knowledge is created by linking existing explicit knowledge, so the representations must supports analysis and reasoning. It is this transfer mode that model-based methods typically focus on. Finally, activities like usability testing and end-user validation requires a process of *internalization*.

The need for using and transferring between different design representations, implies that there exist different classes of representations. We have found it useful to classify representations along three dimensions, as shown right in Figure 1: 1) the perspective, ranging from problem- to solution-oriented, 2) the granularity of the objects covered and 3) the level of formality. A language or method will usually cover a large part of this space, while a particular diagram covers a smaller part. E.g. contemporary

task modelling languages are typically problem-oriented and formal, and suited for mid- to low-level actions. Workflow languages on the other hand, support a higher level view of actions. All the nine user interface models listed in [8] can be placed in the 2 dimensional space spanned by the perspective and granularity axes.

Nonaka's transfer modes correspond to movements along the formality dimension, i.e. the top double-arrowed line in Figure 1. There will be similar movements along the other two dimensions, according to the needs of the development process and the participants, and the intended role of the representation. Different roles are summarized in Table 1.

1. *Perspectives*: Movements from *solution to problem* is the "textbook" flow of a problem solving: first specify the problem, then solve it according to the constraints expressed in the specification. The representation should support looking "forward" towards constructing one or several solutions. Moving from *solution to problem* is used when analyzing a solution for the purpose of comparing it against a previously formulated specification.

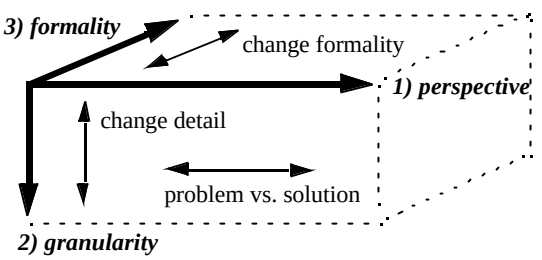


*Figure 2.* Dimensions of the representation space and movements within it.

2. *Granularity*: Moving *downwards*, details are constructed from a high-level description. This is similar to the problem-solution movement, in that it requires satisfying the constraints given by a more abstract specification. In the *upwards* movement, the aggregated characteristics are constructed from an analysis of one level, and relies on the *compositionality* of the language used.
3. *Formality*: When *formalizing* a design vague ideas are made less ambiguous and more precise. The common tacit understanding of design ideas must be made explicit through communication. Making a formal representation *less formal* may seem unnatural, but makes sense in a context where participants are unfamiliar with formal notations.

| **Role** | **Objective** |
|---|---|
| Semantic | Capture domain knowledge. |
| Communicative | Communicate domain as represented among participants |
| Constructive | Guide and constrain further design |
| Analytic | Interpret and evaluate current representations |

*Table 1.* Roles the design representation plays

The model-based design approaches usually focus on the movements from left to right, and top to bottom within the formal "plane". Movements are just as relevant, e.g. when using more exploratory and informal approaches. To develop a more industry-friendly model-based approach, we should design modelling languages with better support for all movements in the representation space.

## 3. INTEGRATED USER INTERFACE MODELLING

We have developed three languages for domain, task and dialog modelling, with the goal of making model-based user interface design more developer-friendly. The domain modelling language, mainly developed by Arne Sølvberg [9], is used in the context of both task and dialog modelling. The task modelling language is based on the APM workflow language [10], and is a hybrid of hierarchical task and workflow modelling languages. The dialog modelling language combines interconnected interactors for expressing information flow and the Statechart language for control and activation. The three languages will be briefly presented below. Although they all have a formal basis, the focus will be on expressiveness and flexibility, to illustrate how we believe they provide better support for the different movements discussed above.

### 3.1 Domain modelling with RML

Our domain modelling language, the Referent Modelling Language [9], provides constructs for naming concepts and their extensions (sets), partitioning them, relating them to each other and defining their attributes, all based on set theory. Figure 3 illustrates our notation in an RML model of task modelling concepts, similar to the ontology suggested in [8]. The rectangle labeled 'Task' defines the TASK concept, which is related to a set of RESOURCES, through the REQUIREMENT and ACHIEVEMENT relations (lines). Each TASK requires at least one RESOURCE, expressed by the small black circle. There are three kinds of resources that a task may require: ACTORS, CONCEPTS/SETS (domain data) and TOOLS. The encircled '+' sign indicates that these *specializations/ subsets* are disjoint. Each task is related to an ACTOR through the PERFORMANCE relation, and this relation is a *specialization/subset* of the REQUIREMENT relation. Tasks are related through an aggregation relation, indicated by the encircled 'x'. Each task's participation in the aggregation is the basis for classifying it as either SUPERTASK or ACTION and either TOPLEVEL or SUBTASK.

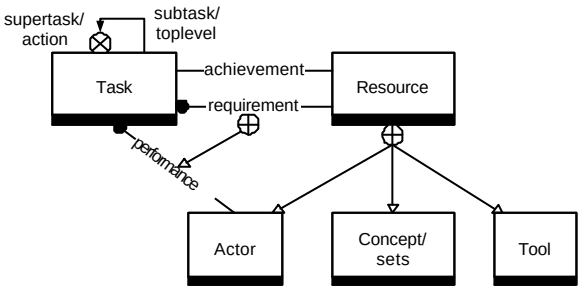


*Figure 3.* Generic model of tasks and their performance

In RML, instances may be (re)classified many times in their lifetime as their characteristics change. This differs from most object-oriented modelling languages like UML, where the instance-class relation is static. Compared to UML's class diagrams RML provides a richer and more flexible visual notation, e.g. attributes may be grouped in separate boxes and cardinality is visually indicated. RML provides better support for modelling differing world-views in a single diagram, e.g. by allowing several orthogonal specializations for the same concept. Although its philosophical roots

are distinct from UML's, most RML constructs have UML correspondences, so RML most models may be translated to UML for software engineering purposes.

### 3.2 Task modelling with TaskMODL and RML

Our task modelling language TaskMODL, utilizes and builds on RML in three ways. First, it is used for modelling the domain, as shown in Figure 4, upper right, where the model of MESSAGES and MAILBOXES provides a context for the READ EMAIL task. The second usage is for expressing contextual information, dataflow and pre- and post-conditions. In Figure 4, the IN mailbox instance is required for performing GET NEW EMAIL, a set of messages is required for MANAGE EMAIL and each MESSAGE in MESSAGES provides a context for a MANAGE MESSAGE task. The *cardinality* of the MANAGE EMAIL-MANAGE MESSAGE aggregation corresponds to the size of the set of messages. This exemplifies the tight coupling between the static domain and dynamic task model; the static structure constrains what is meaningful action, and is the reason these should be integrated both theoretically/formally and practically/visually.

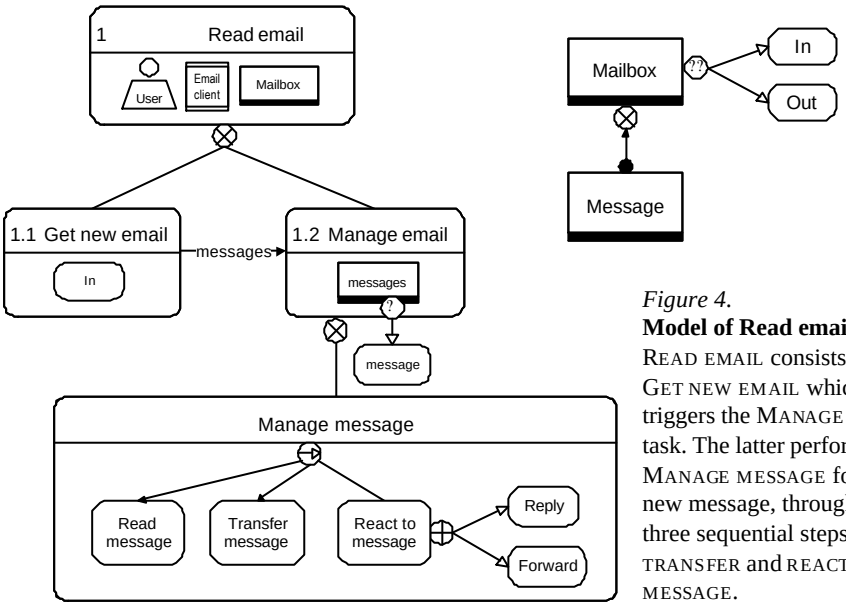


*Figure 4.*
**Model of Read email task**: READ EMAIL consists of GET NEW EMAIL which triggers the MANAGE EMAIL task. The latter performs MANAGE MESSAGE for each new message, through the three sequential steps READ, TRANSFER and REACT TO MESSAGE.

TaskMODL's third usage of RML is for its own definition and notation. In the RML model in Figure 3, it is expressed that SUPERTASKS contain SUBTASKS. This is a generic constraint, and in specific task models we express more specific constraints, e.g. that a GET NEW MESSAGES task will be performed as part of a READ EMAIL task. In a standard use of meta-models, the READ EMAIL task would be an *instance* of the TASK concept, but we have chosen to define it as a *specialization* of TASK, i.e. at the *same meta-level* as the generic task model in Figure 3. This means that TaskMODL models are really RML models in disguise, and therefore the integration is straight-forward. In fact, most of RML's notation is carried over, including cardinality and specialization.

RML lacks features for modelling how a domain evolves over time, so a set of standard sequence constraints have been added. Repetition is expressed through the

cardinality of the SUPERTASK-SUBTASK aggregation. Abstraction of human action is important [14], and this is supported by the dual use of the choice operator (+) inherited from RML. In the example model, the REPLY TO MESSAGE and FORWARD MESSAGE are *specializations* of REACT TO MESSAGE. The meaning of specialization is inherited from RML: the static constraints of the general task must be met by the specialized tasks. Several specializations of a task may defined for different conditions, e.g. classes of input data, user stereotypes and use of different interface designs.

Compared to ConcurTaskTrees [13], TaskMODL provides several advantages.

1. TaskMODL is semantically and visually integrated with a domain modelling language. A corresponding integration should be possible with UML, providing a smoother integration than that suggested between CTT and UML in [15].
2. Specialisation is given a natural interpretation, providing means for managing task knowledge.
3. The notation is more flexible: The *hybrid* tree and containment-based hierarchical notation, provides better support for a combination of traditional hierarchical and dataflow-oriented style. *Anonymous* tasks can be used when a supertask is needed for grouping, but no name is needed for readability, as shown left in Figure 5, where a complex constraint is expressed using two anonymous tasks.
4. The sequence constraints are more flexible than CTT's binary operators. They can be *decoupling* from supertasks, which in the left fragment in Figure 5 can be used to removed the anonymous tasks altogether. TaskMODL also supports *non-strict* constraint *trees*, as illustrated in the right fragment, which makes expressing complex sequence constraints even more practical.

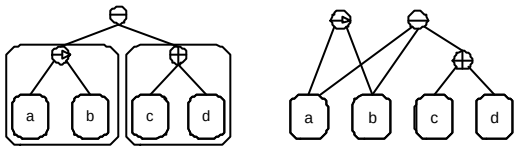


*Figure 5.* Left: order(sequence(a, b), choice(c, d))
Right: sequence(a, b) & order(a, b, choice(c, d))

### 3.3 Dialog modelling with DiaMODL and RML

Our dialog modelling language DiaMODL is based on the interactor abstraction from [18] for expressing information flow. We avoid basing control and activation on LOTOS and use Statecharts instead, a trade-off between utilizing previous theoretical results and making the language more practical. The choice is due to Statecharts' simplicity and the fact that it is already part of UML.

As previously mentioned, we are interested in supporting movements within the design representation space introduced in Section 2. The compositionality of interactors directly supports movements along the granularity dimension. Moving between abstract and concrete interaction is supported by providing an interactor definition for every standard widgets, as exemplified in Figure 6. The left RML fragment identifies the information that the interactor at the right can input and output. The interior represents a suitable widget supporting this interactor signature. If a folder view appears in a design sketch, we can suggest this interactor as its abstraction. Alternatively, if an

abstract dialog model is consistent with this fragment, we can use the folder view in our concrete design, or for explaining our abstract model. We have modelled most standard widgets as interactors, and are working on ways to mix abstract interactors and concrete widgets in a GUI builder, to support the abstract-concrete movement in both directions. Traditionally, interactors have been applied to structured dialogs and forms, but our inclusion of Statecharts provides support for direct manipulation, in the way suggested in [20].

In [16], it is suggested that the structure of interconnected interactors can be represented by UML's objects diagram. With our use of Statecharts, the control part could be represented as well. Our way of mixing interactors/objects with Statecharts is not directly representable, though, since UML's State diagrams are local to an object and the composition of Statecharts in objects aggregations is unclear.

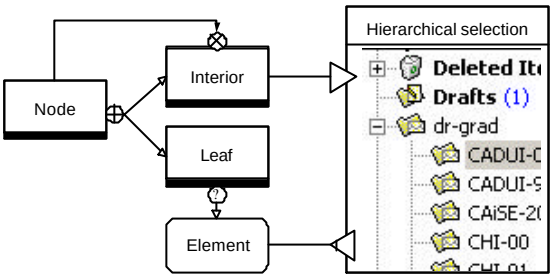


*Figure 6.* Interactor for selecting a leaf element from a hierarchy, and corresponding folder view

## 4. MODEL-BASED DESIGN PATTERNS

Design is about making *choices* concerning *which* sequences of action the user should be able to perform, *which* design elements are used and *how* they are composed to support this behavior. The movements presented in Section 2, correspond to such design choices, e.g. how tasks are mapped to dialog structure, how formal specifications are derived from design sketches and how dialog structures are decomposed. Making these choices or movements requires knowledge, and accordingly, our framework can be used for classifying design knowledge. E.g, rules for mapping from abstract dialog elements to concrete widgets would be placed as shown in Figure 7.

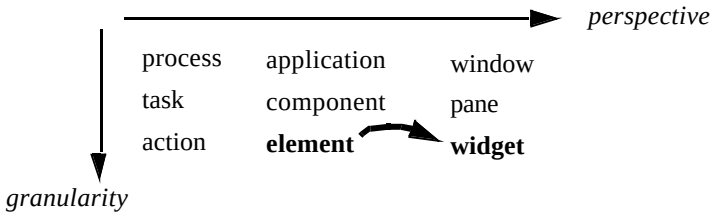


*Figure 7.* Classifying design knowledge:
Design knowledge for mapping dialog elementes to widgets

The UID community has a long tradition of formulating design knowledge in principles, rules and guidelines, and there exists some attempt to formalize it, e.g. [21]. A problem is that such knowledge is either very high-level and general or very specific [23]. For capturing "middle-level" design knowledge, the use of *UID patterns* is gain-

**Problem**: The user needs to browse the elements of a hierarchy and select one.

**Principle**: Provide separate connected panes for specialised viewers.

**Context**: Many application contains aggregated data, which the user must browse through. and the user often wants to invoke a function taking one of the parts as input parameter.

**Forces**: 1) Freedom for the application to visualise the set of containers, parts and individual items in specific ways. 2) Freedom for the user to resize each viewer.

**Solution**: Split a window into three panes, one for the viewing a set of containers, one for viewing a set of parts, and one for viewing individual parts.The former two must provide item selection. The selection of a container should determine the content of the parts pane, and the selected part should determine the content of the part pane.

**Rationale**: The desire to act on information often comes when seeing it. Hence, it makes sense to be able to use presented information as input.

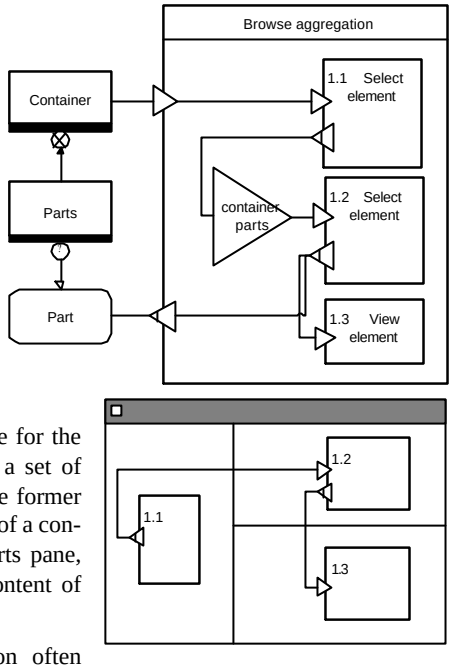


*Figure 8.* The ***Browsing a container*** design pattern for browsing aggregations

ing interest and momentum. The pattern concept originated in architecture ([22]), and simply stated, represents a generic, proven and accepted solution to a reoccurring design problem, in a way that facilitates (re)use in a new design context. We believe patterns can become a useful design and engineering tool, if we are pragmatic about philosophical and often almost religious issues concerning pattern discovery and formulation/formats. In our context, design patterns can simply be interpreted as recipes for how to perform sound movements within our representation space. As such, they can be used bottom-up as building blocks, top-down for design refinement and to move between perspectives. In a model-based approach to design, it is natural to use model fragments, and in our own experience, the abstraction and precision they provide is very helpful when formulating a pattern [24]. It is crucial that the modelling languages support combination of several perspectives, and this is part of the motivation for integrating them. We are currently experimenting with using model fragments utilizing our three modelling languages in pattern formulations.

Figure 8 shows a pattern for browsing aggregation hierarchies and selecting an element. The interactor signature can be seen as a specification, and its composition a suggested solution to a design problem. A layout is suggested for configuring them in window panes. This particular pattern concerns two movements, decomposition of dialog and mapping from abstract to concrete design. We have formulated patterns for selecting concrete dialog elements from abstract interactors, based on mapping like the one shown in Figure 6, and for mapping from tasks to dialogs, e.g. the Managing

Favorites pattern found at [25]. The latter kind is in our experience the most difficult to "find".

Use of formal model (fragments) in design patterns is controversial, partly because of the non-engineering pattern tradition, and partly because formal user interface models are rarely used. The theory of knowledge creation presented in Section 2 suggests that a patterns format should use both formal and informal representations, the former for precision and the latter for supporting recognition and application. In our experience, the most immediate effect of using formal model fragments in patterns is mental, e.g. enhancing the understanding of design and increasing the consciousness of (abstract) design knowledge. What tools and techniques are needed for more practical results, remains to be seen.

## 5. CONCLUSION AND FURTHER WORK

The use of formal methods and languages in user interface design, has always been a source of debate, mirroring the discussion on systems development approaches. UML is criticized for both being too informal and too formal, depending on the context. Most however, agree that an integrated language is important and UML's pragmatic line of compromise wrt. different modelling traditions may be fruitful also in the community's work towards a *uniform user interface modeling language*.

We have argued that there is a use for both formal models and informal design representation methods in user interface development. The role of informal representations is to support creativity, involvement and dialog, among both developers and end-users. Formal representations allows detaching the representation from the meaning-giving context, and supports reflection, analysis and transition to executable representations. More effort should be put into understanding:

- how (design representation) language is used throughout the design process
- the need for integration of modelling languages across perspectives
- how design knowledge can be integrated into a model-based approach to user interface design

## 6. REFERENCES


[1] Myers, B., Hudson, S.E., Pausch, R. Past, Present and Future of User Interface Software Tools. ACM Transactions on Computer-Human Interaction, 7, 2000, p. 3-28.

[2] Puerta, A.R., Cheng, E., Ou, T., Min, J. MOBILE: user-centered interface building. In Proceeding of the Conference on Human factors in computing systems, p. 426-433, 1999.

[3] www.genera.no

[4] Nonaka, I., Takeushi, H. A theory of the firm's knowledge-creation dynamics. In "The dynamic firm. The role of technology, strategy, organization and regions". Chandler jr, A.D, Hagstrøm, P., Søvell, Ø. (eds). Oxford University Press, 1998.

[5] Jones, S., Sapsford, J. The role of informal representations in early design. In Markopoulos, P., Johnson, P. (eds.): Proceedings of DSV-IS'98, Springer-Verlag/Wien. 1998.

[6] Landay, J.A., Myers, B.A. Interactive Sketching for the Early Stages of User Interface

Design. In Proceedings for CHI'95, v.1 p.43-50, 1995.

[7] Gross, M.D., Yi-Luen Do, E. Ambiguous Intentions: A Paper-Like Interface for Creative Design. In Proceedings of UIST'96, p.183-192, 1996.

[8] Vanderdonckt, J.M., Puerta, A.R., Introduction to Computer-Aided Design of User Interfaces. Preface of Vanderdonckt, J., Puerta, A.R. (eds.), Proceedings of CADUI'99, Kluwer Academic Publishers, Dordrecht, October 1999.

[9] Sølvberg, A. Data and what they refer to. P.P.Chen et al.(eds.): Conceptual Modeling, pp.211-226, Lecture Notes in Computer Science, Springer Verlag, 1999.

[10] Carlsen, S., Action Port Model: A Mixed Paradigm Conceptual Workflow Modeling Language. Proceedings of CoopIS - Cooperative Information Systems '98

[11] Trætteberg, H. Modelling work: Workflow and Task Modelling. In Proceedings of CADUI'99, Kluwer Academic Publishers, Dordrecht, October 1999.

[12] Van Welie, M., Van der Veer, G.C., Eliëns, A.:An Ontology for Task World Models. In: Markopoulos, P., Johnson, P. (eds.): Proceedings of DSV-IS'98, Springer-Verlag/Wien (1998) 57-70

[13] Paternò, F., Mancini, C., Meniconi, S.: ConcurTaskTrees: A Diagrammatic Notation for Specifying Task Models. Proceedings of Interact '97, Chapman & Hall (1997) 362-369.

[14] Malone, T.W., et. al. Tools for inventing organizations: Toward a handbook of organizational processes. Management Science 45(3), pp. 425-443, March 1999.

[15] Nunes, N.J. Object Modeling for User-Centered Development and User Interface Design: The Wisdom Approach. PhD thesis from Universidade da Madeira, April 2001.

[16] Markopoulos, P., Marijnissen, P. UML as a representation for Interaction Design. Presented at OZCHI 2000.

[17] Duke, D., Faconti, G., Harrison, M., Paternó, F. Unifying views of interactors. In Proceedings of the workshop on Advanced visual interfaces, June 1 - 4, 1994, pp. 143-152.

[18] Markopoulos, P. A compositional model for the formal specification of user interface software. PhD thesis at Department of Computer Science, Queen Mary and Westfield College, University of London. 1997.

[19] Harel, D. Statecharts: A Visual Formalism for Complex Systems. Science of Computer Programming 8, 1987.

[20] Trætteberg, H. Modelling Direct Manipulation with Referent and Statecharts. In Markopoulos, P., Johnson, P. (eds.): Proceedings of DSV-IS'98, Springer-Verlag/Wien. 1998.

[21] Vanderdonckt, J.M., Bodart, F. Encapsulating Knowledge for Intelligent Automatic Interaction Objects Selection. In Proceedings of INTERCHI'93, p.424-429, 1993.

[22] Alexander, C., Ishikawa, S., Silverstein, M., Jacobson, M., Fiksdahl-King, I.,& Angel, S. A Pattern Language. Oxford University Press, 1977.

[23] van Welie, M., van der Veer, G.C., Eliëns, A. Patterns as Tools for UI Design. International Workshop on Tools for Working with Guidelines, pp. 313-324. Biarritz, October 2000.

[24] van Welie, M., Trætteberg, H. Interaction patterns in user interfaces. PLoP'2000.

[25] http://bscw.gmd.de/pub/english.cgi/0/17771476