UML as a representation for Interaction Design

Panos Markopoulos and Peter Marijnissen
IPO - Center for User-System Interaction
Eindhoven University of Technology
Den Dolech 2, PO Box 513, 5600MB Eindhoven, The Netherlands
P.Markopoulos@tue.nl, P.J.Marijnissen@tue.nl

Abstract

This paper examines the use of the Unified Modeling Language (UML) as a representation for interaction design. We discuss the trade-offs related to applying UML outside its intended application domain and the suitability of UML components for modeling various aspects of interaction: representing user requirements, early envisionment of interaction, task modeling, navigation and detailed interaction specification. Where appropriate we propose and illustrate the combination of UML with purpose-specific notations.

Keywords: Unified Modelling Language, User Interface Specification, Task Modelling, Navigation, Interaction Design.

1. Introduction

UML [5, 20] is a family of notations which is emerging as the de facto industry standard for the specification of object oriented software. The spread of UML among software practitioners suggests that we consider whether and how UML can serve as a modeling technique for human computer interaction (HCI) design. Interaction design is outside the canonical scope of UML, which is the specification of a software system. Interaction design practice can benefit from adopting the industry standard in object modeling, particularly through potential improved communication among interaction designers and software developers. This possibility is drawing increasing attention, notably by a series of workshops on object modeling for interaction design, e.g., [2].

This paper examines how UML can help specify various representations employed in interaction design, the limitations of UML notations for this domain and where it is beneficial to combine UML with HCI purpose-specific notations. We constrain this 'pick and mix' approach to modeling by the following stipulations:

- Where modeling is served sufficiently by a UML notation then that notation should be used.
- We expose difficulties and caveats in applying UML notations outside their intended scope, weighing them against the use of purpose-specific representations.

The paper does not review HCI-notations extensively. Rather, it employs a few tried and tested notations which complement UML. Related work has proposed methodologies for combining interaction design and object modeling [9,19]. Here we take a less prescriptive approach discussing design representations, independently of the design process. Without proposing or advocating a methodology for interactive system design, we distinguish the following broad categories of representations used for interaction design:

- Scenarios for envisioning systems at early stages of the design process.
- Task and domain models for a more thorough and complete representation of user requirements.
- Abstract models of interaction, describing the coarse organisation of interactive elements into groups and navigation between such groups.
- Detailed interaction models, describing the 'look and feel' and the dynamic aspects of interaction down to the level of individual interactive objects.

Section 2 discusses scenarios and use cases, illustrating differences in form and content between their numerous variants. Section 3 discusses task models. Domain models are a common modeling concern for software

developers and interaction designers alike; they can be modeled as UML class diagrams and we do not discuss them further in this paper. Navigation and grouping of interaction objects is discussed in section 4. Section 5 discusses more detailed interaction specification. Finally, section 6 discusses the conclusions from this analytical investigation and future research steps.

2. Scenarios and use cases

There is some confusion concerning the relationship between scenarios as understood in the field of HCI (interaction scenarios for short) and use cases, as defined within UML. Very diverse representations serving different modeling purposes are described by their proponents as scenarios or use cases. While the dominance of UML makes use cases somewhat a less ambiguous concept, scenarios are not part of a singular family of notations, and scenario based design is not a single, coherent design method. Rather, scenarios are a theme common to very varied design approaches and activities. Two classic collections of articles in [22] and [6] illustrate how varied meanings the term scenario may have in the field of HCI and how varied design approaches are called scenario-based.

UML [5] defines a use case as a classifier, which describes a set of sequences of actions that a system performs to yield an observable result of value to an actor. In UML parlance a scenario is a single such sequence of actions, which instantiates a use case. UML use cases are used throughout software development and individual use cases are instrumental to support traceability between these models. We note the following major differences between use cases and interaction scenarios:

- Use cases describe the behaviour of the system under design not of the user using it. For some authors, an interaction scenario, captures primarily user rather than system behaviour. E.g., Carroll suggests that a scenario identifies the person, their motivations, describes the actions and the reasons that these actions were taken and characterises their results in terms of the users' motivations and expectations [7]. Even if system behaviour only is described, an interaction scenario describes functionality from the viewpoint of a single user – who may be one of the many actors related to a use case.
- Use cases are performed by actors who, apart from human users, can also be systems.
- Use cases specify system functionality. At a semantic level they can serve as a specification of interactions, but they normally abstract away from user interface specifics (see for example [8]). An interaction scenario aims precisely to envision and specify user interface design ideas or decisions [7, 11].

The points above are independent of the representation style; they concern the essence of use cases and interaction scenarios. Both use cases and interaction scenarios can be represented in many forms: narrative, structured text or they can be associated with storyboards, notations such as state machines, pre-and post-conditions associated with interactions, etc. We can think of use cases and interaction scenarios as different classes of design representations distinguished by content, viewpoint and intended use in design.

The overlap between use cases, scenarios as specifications of an interface and task models (the latter are the topic of section 3) is discussed in [2] propose a separate model, which summarises common content and supports the traceability between these three types of representations. Although a useful conceptualisation, that model does not adequately portray the shift in viewpoint between use cases and scenarios/task models. One way to distinguish the interactions with each actor, employed within the WISDOM method [19], is to describe use cases with activity diagrams, distinguishing the interactions of each actor on a separate 'swimlane'. The resulting activity diagram is still substantially different to an interaction scenario.

The example below illustrates the variation in content and form between use cases described as a structured narrative, adapting the template of Cockburn [8], as a user story [3] and as an interaction scenario also written as a story [11]. User stories are a form of use cases, described in a very informal text, used within streamlined software development [3]. A user story contains the most essential elements of the use case: the actor, the main event path and the context of execution. Like a use case it describes the system operation required by an actor to achieve its goal. Correspondingly, Erickson [11] describes a form of interaction scenarios, which he calls 'stories', which help capture design rationale, involve users and transfer design knowledge.

2.1. Example

This example is drawn from a collaborative project with BOVA, a Dutch luxury coach manufacturing company. The project aims to support their business process for product improvement, by an intranet-based system called BOVANET. BOVANET will automate field reporting and handling warrantee claims, to speed up detecting and resolving potential problems with operating coaches. It will be employed in different countries of Europe and it will interact with existing software systems, e.g., databases, company intranet, etc.

We will discuss the use case "Dealer Submits Field Report" which is introduced as a sub-case of "Handle Field Report". In the user and the interaction stories we use the pseudonym 'Anelka' for the main actor.

1. Use Case: Handle Field Report
Context: Supports the business level goal of speeding up the product improvement cycle.
Scope: BOVA organisation.
Primary actor: Dealer: An office worker in a company that sells BOVA coaches in Europe.
Other Actors: BOVA-NL.
Pre-condition: Dealer is notified of a technical problem of a coach in operation.
Main Event Path:
1.1 Dealer submits field report.
1.2 BOVA-NL manages problem report.
1.3 BOVA-NL publishes service bulletin.
Post Conditions: BOVANET confirms reception of Field Report and publishes it in a collection of all field reports.

1.1 Use Case: Dealer submits field report
Context: This use case supports the goal of automating report handling.
Scope: BOVANET only.
Primary actor Dealer
Other Actors: Client (owner of a BOVA coach).
Pre-condition: Dealer receives complaint or warrantee claim.
Main Event Path:
1.1.1 Dealer writes a Field Report.
1.1.2 Dealer enters Field Report to BOVANET.
1.1.3 BOVANET confirms that BOVA has received the Field Report.
Post Conditions: BOVANET confirms reception of field report.

User Story: Dealer submits Field Report

By talking to clients Anelka, a dealer for BOVA, notices a recurring mechanical problem with BOVA buses. Anelka describes the problem in a "Field Report" and enters the Field Report into the BOVANET system. After a while, BOVANET sends a confirmation message for reception of the field report.

Interaction Story: Dealer submits Field Report

A client from Paris calls Anelka to complain about a mechanical problem with a coach in operation. Anelka enters the information to BOVANET while speaking to this client on the phone. From the entry page of BOVANET, he selects a link Field Report and then opens a new Field Report. This is an electronic form that looks like the old paper version. The date, the establishment and his name are filled in automatically. BOVANET offers a choice of possible categories of problems: Anelka selects 'Engine'. He types in a brief description of the problem and enters the relevant part numbers with the help of the BOVANET part-browser. Anelka has a last look, and pushes the button labeled 'submit to BOVA-NL'. After a few seconds, an automatically generated e-mail arrives in his message box.

We make the following observations on this example:

- The interaction scenario is over-elaborate as a functionality specification and has an implementation bias: e.g., using links, forms, pushing buttons.
- The use case involves several actors, two of which are users of the BOVA system without clarifying the functionality offered to each. The interaction scenario, gives a user-specific view of the same functionality describing the interface for this user.
- The user story is the briefest and simplest description. It contains broadly the same information as the structured use case description, although the relevant parts are not explicitly characterised as 'actors', 'pre-condition', etc.
- The user story personifies the main actor, but this has no direct significance for the functional specification. On the contrary, identifying the user in the interaction story pertains to a categorisation of users. In our example, a user study has prompted a categorisation by the establishment they work for (rather than a finer distinction by job-function) and by nationality. Both are reflected in the interaction scenario.
- There are many more variants of scenarios and use cases, each with their own proponents and advantages. The similarity of representation forms involves a danger of confusing interaction scenarios for functionality specifications, or of committing use cases to interaction design decisions. While there is an overlap in content use cases and interaction scenarios have different content, purpose and viewpoint. Ensuring that these design representations have the right content, and are used appropriately is not a notational issue but, rather, a process issue, which should be addressed by the methodology for designing and developing interactive systems.

3. Task Modeling

HCI research has spawned a number of design approaches, which share a commitment to base design on task analysis, e.g., [14, 16, 21, 23]. The theoretical foundations of task analysis techniques vary, but the representations used converge in the following ways [4]:

- The decomposition of user tasks to sub-tasks is represented as a tree structure.
- Task trees are decorated with temporal ordering of task activity.

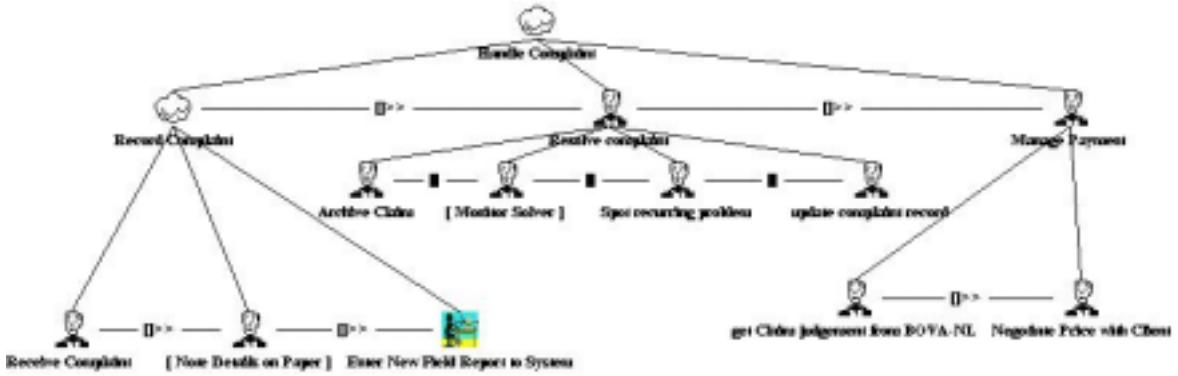

Figure 1. A ConcurTaskTree [21] representation of the Dealer's task 'Handle Complaint'. The diagram was created using the editor provided by CNUCE [10].

- Elementary task activity is associated with task-domain representations in terms of objects.

Clearly, a task model can be rendered in the same form as use cases. We note the following caveats:

- As with scenarios, a task model describes primarily user activity and can refer to user interface elements (depending on level of abstraction) while use cases should abstract away from these issues.
- Use cases confound functionality used by multiple actors of the system, while task models describe separately the goal oriented activity of individual users.
- The apparent gain of using the same notation for specifying functionality and tasks must be counterbalanced by the possible confusion between use cases modeling system-wide function and use cases describing single user interaction. On the other hand task hierarchies are simple representations easily graspable, even to share with users.

The example below compares the specification of tasks using a purpose-specific task modeling notation, use cases and UML's activity diagrams.

3.1. Example

This example specifies the task of handling a customer complaint. First the task is specified using ConcurTaskTrees [21]. Out of the many options for task modeling we choose this notation because:

- It provides powerful operators that capture recurring behaviours for user-system interaction. These behaviours would otherwise result in complex specifications in a simpler notation.
- Tool support is provided by CNUCE as a Java application downloadable from [10].

We explain briefly the task model of Figure 1. The top level task 'Handle Complaint' is *abstract*, shown as a cloud. This means that it involves a mix of interactive tasks, user tasks and application tasks. It decomposes to three sub-tasks. These are performed in sequence with an information flow between them, denoted by the symbol []>>. Sub-tasks represented by a face-icon, are performed by the user. The icon showing a character operating a computer denotes a task performed conjointly by human and machine. The symbol ||| relates tasks which are performed concurrently in any order.

The task model of Figure 1 does not describe the operation of the system under design, i.e. it specifies the task as it is performed prior to the introduction of the system under design. However, we could use the same notation to represent how the designer envisions the task performance with the system under design [18]. The tree describes the task of a single user and a separate 'cooperation tree' specifies the cooperation between users [21]. In UML, the involvement of many users is illustrated by a use case diagram, as in Figure 2. To specify user tasks we must constrain the use cases to describe the functionality used and observed by a single user. Different use cases would need to be written to specify the tasks of the remaining users.

A task model may describe user activities, outside the scope of the specified system functionality. If interaction-tasks are specified the task model may refer to user interface-specific elements. This is useful for thinking about the task and the user interface design, but is a verbose and unclear specification of system functionality when compared to the standard use case model.

We illustrate this point by specifying the task 'Handle Claim' using use cases. We obtain a succinct textual representation of the task, but one which has a distinctly different scope than the use cases of section 2. The change in name and context reflect the different viewpoint: we do not describe the business process of handling field reports but a single user task, embedded in this process.

Use Case: Dealer Handles Customer Complaint

Context: Complaint handling in current situation.
Scope: BOVA organisation
Primary actor Dealer
Other Actors: Clients, BOVA-NL, Solver (contracted garages)
Main Event Path:
1. Record Complaint
2. Resolve Complaint
3. Manage Payment
Post Conditions: Complaint has been resolved and paid. Information about the complaint is archived for later processing.

1. Use Case: Record Complaint
Context: A client telephones or faxes a complaint.
Scope: Dealer Establishment
Primary Actor: Dealer
Other Actors: Client
Main Event Path:
1.1. Receive Complaint from Client
1.2. Note Complaint Details on Paper
1.3. Type-in Standard Request Form

2. Use Case: Resolve Complaint
Context: Complaint Details have been recorded
Scope: Dealer and Solver Establishment
Primary Actor: Dealer
Other Actors: Client, Solver (contracted repair garage)

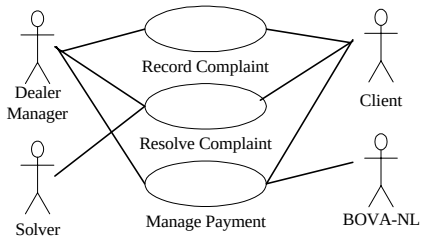

Figure 2. Use Case Model for the task 'Handle Complaint'

Main Event Path:
2.1. Archive Claim
2.2. Monitor Solver
2.3. Spot Recurring Problem

3. Use Case: Manage Payment
Context: Complaint has been resolved, client submits a warrantee claim
Scope: BOVA organisation
Primary Actor: Dealer
Other Actors: BOVA-NL, Client
Main Event Path:
3.1. BOVA-NL sends judgement concerning warrantee cover
3.2 Negotiate Price with Client

Extensions
1.1.a Submitted Data is incomplete.
1.1.a.1. Call customer
1.1.a.2. Ask info from company central database
2.2.a Solver deals directly with client
3.2.a Client uses invoices to negotiate discount for a new coach purchase.

The same task is specified in Figure 3 using UML's activity diagrams. This notation is most useful during requirements capture to specify the activity of different actors and during software specification to specify the order of execution of object-methods. Activity diagrams captures sequencing information about the task quite clearly and succinctly, allowing for the specification of unordered or concurrent activities. The allocation of activities to actors can also be shown, by partitioning the diagram into swimlanes (a construct not used here). Activity diagrams do not show as clearly (as concurTaskTrees) the hierarchical decomposition of tasks, and tend to bring in excessive detail.

In conclusion, we make the following observations:

- Use cases and activity diagrams are quite intuitive representations of temporal ordering of tasks.
- Different use cases should specify the task of each actor using the same system functionality.
- Both use case and activity diagrams do not show clearly task decomposition structure. The use of an external to UML notation like ConcurTaskTrees is a useful but not always necessary addition to UML.

4. Abstract Interaction Model

An abstract interaction model is a high level description of the user interface. Such models have been used within model based design approaches, e.g., [14,18], or are thought of as 'low-fidelity' prototypes in prototyping approaches, e.g., [15]. The abstract interaction model describes:

- The groupings of interactive objects into screens, windows or dialogues.
- The navigation between theses groups.

The drawing of figure 4 illustrates how interactive objects are grouped together into the screen Field Report. The abstract interface model is a rough indication only of the layout and the presentation of interactive objects.

Figure 5 uses statecharts (the state diagram component of UML) to specify the navigation between screens. In this diagram every state corresponds to a different page shown on the browser. Events of this diagram correspond to user input that effect transitions between pages, e.g., clicking on a link. This model describes high level navigation between screens, rather than finer issues such as backtracking, history, concurrent or multi-threaded behaviours of objects. Note, the composite state at the left of figure 5. It denotes that the user can view only one of the enclosed screens at a time, and that each screen is reachable from any other screen in this node. The black bullet indicates the starting state within the Field Reports node.

The WISDOM method [19] uses activity diagrams to outline storyboards associated with use cases. As a notation, state diagrams are a superset of activity diagrams, and are better suited to describe the screen transitions that can occur because of 'asynchronous' user input rather than the completion of an activity. The higher level constructs of statecharts also add to the economy of the representation (as for example the composite state of figure 5).

Modeling screens with the concept of a 'state' can pose some problems. In general, the same screen can correspond to multiple states of a system. This mismatch grows as more detail is added to the navigation model, and it becomes more practical to associate a state diagram with each interactive object. Purpose specific notations, e.g., Handie [1], overcome this problem by abstracting away from the actions that effect transitions between screen. Figure 6, illustrates how this notation helps specify the navigation from the field reports screen. Handie uses a form of constraints to specify conditional transitions between screens. Conditional transitions are shown as thick arrows, which end at a class node. A class node represents a collection of nodes, e.g., the field reports in our example, one of which is accessed according to the condition associated with the transition.

5. Detailed Interaction Specification

UML does not provide any special notational support for specifying the dynamic aspects of user-system interaction. State diagrams can serve this role, providing a framework for the specification and implementation of user interface software [13]. A state diagram associated with an interactive object can specify its dynamic behaviour. A more user oriented description is afforded by the User Action Notation (UAN) [12] which provides an extensible framework for specifying low level interaction tasks as well as their hierarchical composition. Here we focus on a user-centred description; [17] discusses the duality and complementarity between a specification of

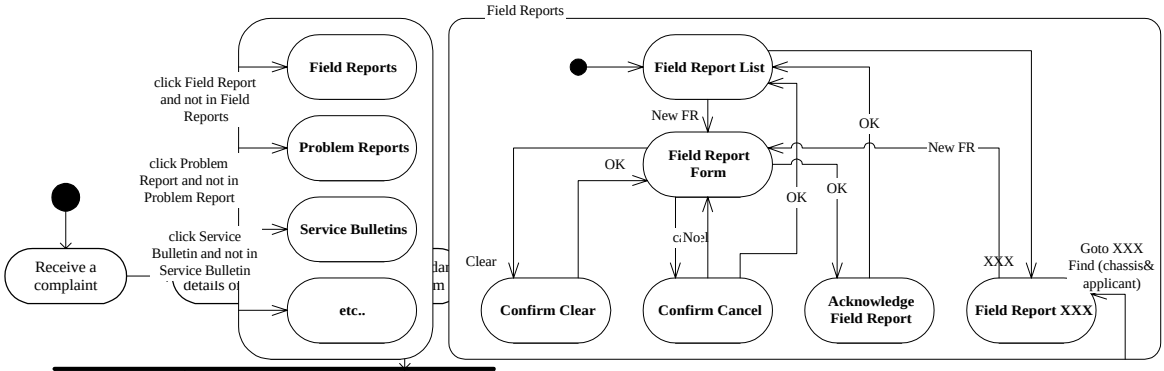

Figure 5. Specification of navigation with statecharts

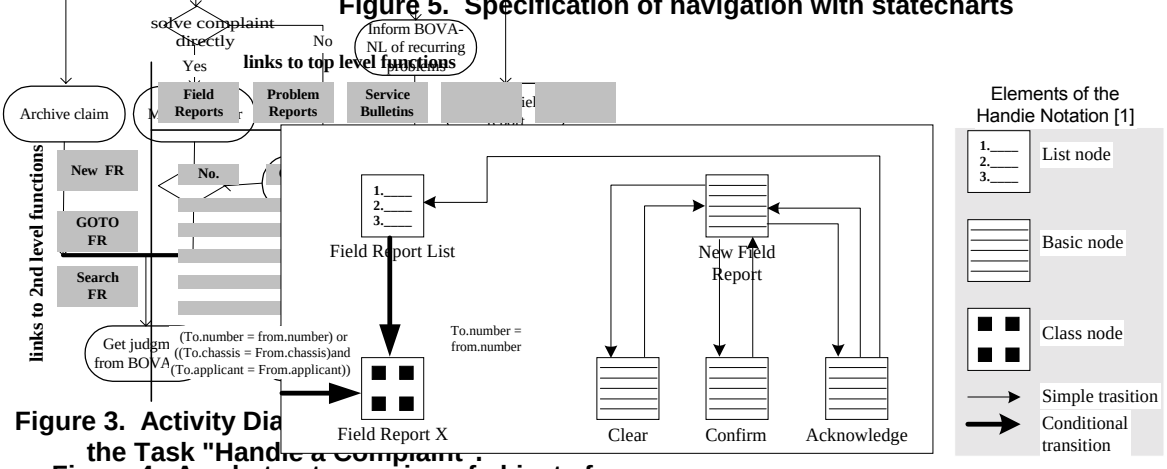

Figure 3. Activity Dia… the Task "Handle a Complaint".

Figure 4. An abstract grouping of objects for the screen "Field Report".

Figure 6. Navigation specified with the Handie notation [1].

interaction centred around interactive objects and the user centric description of UAN.

In UAN, a user interface is specified by a hierarchy of interaction tasks. The sequencing within each task is independent of that in the others. An interaction task is specified by a table with four columns. These columns specify: user actions, interface feedback, 'state' of the user interface and connection to the computation. These tables are read left to right, top to bottom (see [12] for a tutorial text).

5.1. Example

The two tables that follow illustrate the hierarchical composition of tasks using temporal operators. Here one subtask is enclosed in brackets, indicating that it is optional. In the first task (Handle a Complaint), subtasks are performed in sequence. The task 'Record Complaint' is decomposed further into subtasks, which are performed in any order.

Task: Handle Complaint
Record Complaint
Resolve Complaint
Manage Payment

Task: Record Complaint
Open existing Field Report |||
Create New Field Report

So far a UAN specification shows more or less the same information that is expressed using ConcurTaskTrees, although, compared to section 3, we focus now on the task as performed with the envisioned system. The choice between the diagrammatic form of ConcurTaskTrees and the textual-tabular form of UAN is largely a matter of taste and tool support. The lower level task specifications illustrate the detailed interaction specification using the four-column format of UAN. Note that the third column is empty as we do not add any extra behaviour to that of the browser that will be used to access the intranet application.

Hix and Hartson [12] describe short-hands for mouse and keyboard based interaction. E.g. clicking on "New" can be specified as ~[New]Mv^. This results in an even finer description, which is useful when the 'look and feel' of the interactive components used is also to be designed. In our case-study, this level of detail is not necessary because the built-in components of Java will be used. An example of specifying an alternative 'look and feel' using a variant of UAN can be found in [18].

We note the following:

- UAN can be used as an alternative to ConcurTaskTrees for the specification of tasks. It covers a wider range of abstraction levels. ConcurTaskTrees provides a more succinct overview of the task.
- The UAN specification refers to interaction objects. It makes little sense without some definition of the screen content as for example in figure 4.

6. Discussion

The field of HCI is in a similar state of affairs as object oriented analysis and design prior to the advent of UML. Researchers have produced a wealth of design representations and design approaches with varying strengths and weaknesses, but this type of knowledge is not always operationalised for the practitioners and there is a considerable inconsistency in the use of terminology. There is no commonly accepted 'best practice' either for specification or in terms of an interaction design method, although there is a gradually maturing understanding as to the strengths and weaknesses of different representations and design techniques.

Selecting the 'best-fit' notation and design approach on a 'problem by problem' basis is a flexible approach, that can make the best use of the range of notations that HCI and software engineering research have produced. However, this approach favours the researcher than the practitioner. In contrast, UML encapsulates a sufficiently rich set of representations with carefully chosen level of precision/formality, that is enough to attack most types of software design problems. UML offers an opportunity for HCI as a field to enhance its impact on software development practice, if it can be established as a common representation scheme for designers and software developers.

This paper has shown that even for the purposes of core interaction design activities, such as task modeling (discussed in section 3) and detailed interaction specification (discussed in section 4) UML provides a range of representational possibilities. However, we have argued that these are not always sufficient or as clear as purpose-specific representations. While the combination of UML with HCI-notations is feasible on an ad-hoc basis, it is clear that these limitations must be addressed within the framework of UML. Future work will examine the extension of UML to provide the required modelling capabilities.

A recurring issue throughout the paper, has been to ensure that a design representation has the right content and is used appropriately. This difficulty is essentially a methodological issue and this paper has tried to investigate a methodology-independent set of notations. In practice however, the way notations will be used is a major concern and methodological guidelines for the use of such notations are required. Further, the issue of the design process itself must be treated as part of our future work.

7. References

[1] Apperley, M.D. and Hunt, R.B. "Design Support for Hypermedia Documents", Sutcliffe, A., Ziegler, J. and Johnson, P. (Eds.) *Designing effective and usable multimedia systems*, Kluwer, 1998, pp. 41-56.

[2] Artim, J.; van Harmelen, M.; Butler, K.; Henderson, A.; kovacevic, S.; Lu, S.; Overmyer, S.; Roberts, D.; Tarby, J-C and Vander Linden, K. "Incorporating Work, Process and Task Analysis into Commercial and Industrial Object-Oriented Systems Development", *SIGCHI Bulletin*, Vol 30, No.4, 1998, pp.33-36.

[3] Beck, K. *Extreme Programming Explained: Embrace Change*, Addison Wesley, 1999.

[4] Bomsdorf, B. and Szwillus, G. "From Task to Dialogue: Task based interface design", *SIGCHI Bulletin*, 30(4), 1998, pp. 40-42.

[5] Booch, G.; Jacobson, I. and Rumbaugh, J. *The Unified Modeling Language User Guide*, 1999, Addison Wesley.

[6] Carroll, J. M. *Scenario-Based Design*, NY: Wiley, 1995.

[7] Carroll, J. M. "Scenario based design", Helander, M.; Landauer, T. and Prablu (Eds.) *Handbook of Human computer Interaction*, Elsevier, 1997, pp.383-406.

[8] Cockburn, A. "Structuring Use Cases with Goals", *Journal of Object Oriented Programming*, Sep/Oct 1997, pp.35-40 and Nov/Dec 1997, pp.56-62.

[9] Constantine,L.L., and Lockwood, L.A.D. *Software for Use: A Practical Guide to the Models and Methods of Usage-Centered Design*, Reading, MA: Addison-Wesley, 1999.

[10] CTTE software *giove. cnuce.cnr.it/ctte.html*

[11] Erickson, T. "Notes on Design Practice: Stories and Prototypes as Catalysts for Communication", in [6], 1995

[12] Hix, D. and Hartson, R.H *Developing user interfaces*, Wiley, 1993.

[13] Horrocks, I., *Constructing the User Interface with Statecharts*, Addison-Wesley, 1999.

[14] Johnson P.; Wilson S.; Markopoulos P. and Pycock J. "ADEPT - Advanced Design Environment for Prototyping with task models", *INTERCHI'93*, ACM Press, 1993, pp. 56.

[15] Landay, J.A. and Myers,. B.A. "Interactive Sketching for the early stages of user interface design", *CHI '95*, 1995, pp.45-50.

[16] Lim, K.Y. and Long, J. *The MUSE method for usability engineering*, Cambridge University Press, 1994.

[17] Markopoulos P; Papatzanis, G; Johnson P & Rowson J. "Validating semi-formal specifications of interactors as design representations", Markopoulos P & Johnson P (Eds) *DSV-IS '98*, Springer-Verlag, 1998, pp.102-116.

[18] Markopoulos, P.; Shrubsole, P., and de Vet, J., "Refinement of the PAC-model for the component-based design and specification of television based user interfaces", Duke, D.J. and Puerta, A. (Eds.) *DSV-IS '99*, Springer, 1999, pp. 117-132.

[19] Nunes, N.J and Cunha, J-F "A Bridge too Far: The Wisdom Approach", *ECOOP'99 Workshop on Interactive System Design and Object Models*, Lisbon, 1999.

[20] "OMG Unified Modeling Language Specification, version 1.3", *http://www.rational.com/uml/resources/ documentation, 1999.*

[21] Paternó, F. *Model-Based Design and Evaluation of Interactive Applications*, Springer, 1999.

[22] *SIGCHI Bulletin*, Vol. 24, No.2.

[23] Wilson, S. and Johnson, P. "Bridging the generation gap: From work tasks to user interface designs", Vanderdonckt, J., Ed., CADUI'96, Presses Universitaires de Namur, 1996, pp. 77-94.

Task: Create New Field Report			
User Action	**Feedback**	**User Interface State**	**Conn. To Computation**
Click "Field Reports"	Display list of field reports		
Click "New" button	Display empty field report		
Enter Applicant Information	Echo entered information		
Enter Coach Information	Echo entered information		
Enter Problem Description	Echo entered information		
Click "OK"	Display list of field reports Acknowledge new field report		Add to data base of field reports Get field report number
Click "Cancel"	Display list of field reports		
Click "Erase"	Display empty field report		

Figure 7. Specification of task "Create New Field Report" with UAN [12].