

# Creative Object-Oriented Modelling: Support for Intuition, Flexibility, and Collaboration in CASE Tools

Christian Heide Damm, Klaus Marius Hansen, Michael Thomsen, and  
Michael Tyrsted

Department of Computer Science, University of Aarhus,  
Aabogade 34, 8200 Aarhus N, Denmark  
{damm,marius,miksen,tyrsted}@daimi.au.dk

**Abstract.** A major strength in object-oriented development is the direct support for domain modelling offered by the conceptual framework underlying object-orientation. In this framework, domains and systems can be analysed and understood using models at a high level of abstraction. To support the construction of such models, a large number of Computer-Aided Software Engineering tools are available. These tools excel in supporting design and implementation, but have little support for elements such as creativity, flexibility, and collaboration. We believe that this lack of support partly explains the low adoption of CASE tools. Based on this, we have developed a tool, Knight, which supports intuition, flexibility, and collaboration by implementing gesture based UML modelling on a large electronic whiteboard. Such support improves CASE tools, and can thus potentially lead to increased adoption of CASE tools and thus ultimately help improving the overall quality of development projects.

## 1 Introduction

Object-oriented programming languages provide many technical qualities but more importantly object-oriented languages and object-oriented development in general also provide a conceptual framework for understanding and modelling [19] [21]. This conceptual framework provides abstraction mechanisms for modelling such as concepts (classes), phenomena (objects), and relations between these (inheritance, association, composition) that allow developers to describe *what their system is all about* and to formulate solutions on a higher level of abstraction than program code.

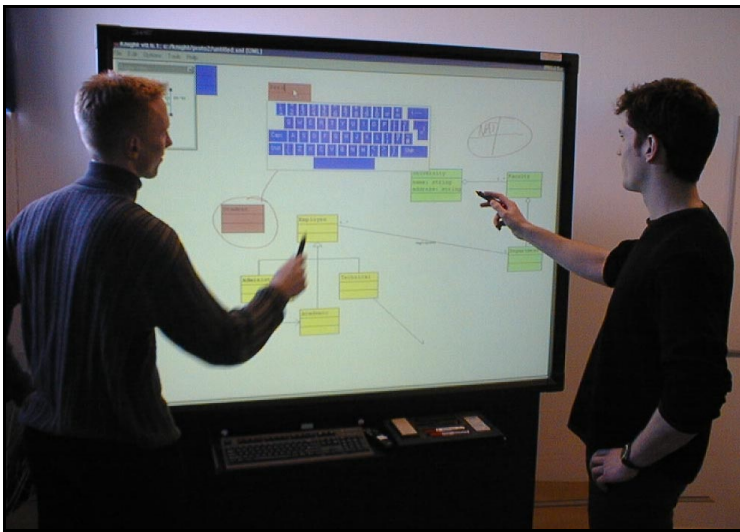
In practice, developers build models at several abstraction levels. The program code can be seen as an executable model, but for purposes such as analysis, specification, documentation, and communication, models visualised graphically in the form of diagrams are often used. The Unified Modeling Language (UML [31]) is a prominent example of a graphical notation that is used for such purposes.

The so-called Computer-Aided Software Engineering tools (CASE tools) provide tool support for modelling. These can among other things help users to:

- create, edit, and layout diagrams,
- perform syntactic and semantic checks of diagrams and simulate and test models,
- share diagrams between and combine diagrams from several users,
- generate code or code skeletons from diagrams (forward engineering) and generate diagram or diagram sketches from code (reverse and round-trip engineering), and
- produce documentation based on diagrams and models.

But even though CASE tools offer these many attractive features, they are in practise not widely and frequently used [1][15][18]. CASE tools clearly support design and implementation phases, but have less support for the initial phases, when the focus is on understanding the problem domain and on modelling the system supporting the problem domain. To rectify this problem, we propose inclusion of support for intuition, flexibility, and collaboration, and that CASE tools should have more direct, less complex user-interfaces, while they should still preserve the current support for more technical aspects such as implementation, testing, and general software engineering issues.

We have implemented a tool, Knight, which acts as an addition to existing CASE tools. It uses a large electronic whiteboard (Fig. 1) to facilitate co-operation and a mixture of formal and informal elements to enable creative problem solving. This paper discusses the support of the Knight tool for modelling through gesture based creation of UML diagrams.



**Fig. 1.** Use of Knight on an electronic whiteboard

Section 2 gives a theoretical overview of modelling, and section 3 discusses related work on current tool support for object-oriented modelling and the adoption of CASE tools. Section 4 presents studies of modelling in practice and section 5 discusses the Knight tool. Finally, section 6 discusses future work and section 7 concludes.

## 2 Modelling and Interpretation

Simplistically, object-oriented software development can be viewed as mapping a set of real-world phenomena and concepts to corresponding objects and classes [19][5]. The set of real-world phenomena and concepts is called the *referent system* and the corresponding computerised system is called the *model system* [19] (Fig. 2).

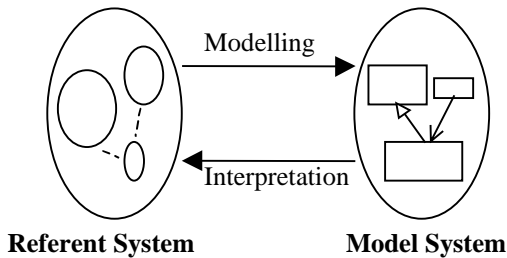


Fig. 2. Modelling and interpretation

Mapping a referent system to a model system is called *modelling*. Modelling is often iterative and it is thus important to be able to discuss a model system in terms of the referent system. We call this reverse process *interpretation* [5]. Modelling is concerned with expressing an understanding of a referent system in a fluent, formal, and complete way, for which usable and formal notations (such as diagramming techniques and programming languages) are crucial. Interpretation is concerned with understanding a model system in terms of the referent system. For doing this, it is important that central concepts in a model system are understandable in the referent system.

Neither understanding the referent system nor building a model system is unproblematic. Understanding the referent system is within the domain of user-oriented disciplines such as ethnography [12] and participatory design [2][11]. The ethnographic perspective on system development is concerned with basing system design on actual work performed within the referent system. The rationale is to avoid a major cause of system failure, namely that the developed system does not fit the work practice within which it is to be used [12]. The participatory design perspective on system design is concerned with designing systems in active collaboration with potential users. This is done for both a moral and a practical reason: Users have to work with the future system, and users are competent practitioners who are knowledgeable of the referent system. Building the model system is within the domain of object-oriented software engineering. Software engineering is concerned with building robust, reliable, and correct systems. In iterative development, then, coordination, communication, and collaboration between different perspectives and individual developers is crucial [5]. Thus, when considering tool support for a system

development process, tool support for the modelling and the interpretation processes is important.

In iterative development, modelling and interpretation are interleaved. When different competencies work together, it is crucial that both processes are, to a certain extent, understood by all involved parties. For example, when doing user involvement it is both important that the developers understand the work of the users (i.e., the referent system) and that the users are able to react on and help design the application (i.e., the model system.)

### 3 Related Work on Tool Support

A large variety of tools that support software development, and modelling in particular, exist. In [28], Reiss presents an overview of 12 categories of software tools. CASE tools, or *Design Editors* as they are also referred to, are tools that let a user design a system using a graphical design notation, and that usually generate at least code skeletons or a code framework.

A large body of literature discusses the lack of adoption of CASE tool. In their study of CASE tool adoption [1], Aaen et al. showed that less than one fourth of the analysts in the companies studied used CASE tools and that about half of the respondents had used the tools for two projects or less. In [15], Kemerer reported that one year after the introduction of a CASE tool, 70 % of them were never used again.

In [13], Iivari confirms, and explains, the low adoption and use of CASE tools. Two of the main conclusions of the study are that high CASE tool usage does indeed increase productivity, but that a high degree of voluntariness in using the CASE tool tends to decrease the use of the tool. Important for our concern, the study also disclosed that many developers found it hard to appreciate CASE tools as they often were perceived as having a high complexity. From this, Iivari concludes that any intelligent means of reducing the perceived complexity might be a very profitable investment. Thus, the study to some extent supports our claim that the user-interface of CASE tools needs to be less complex and more direct.

A recent study by Lending and Chervany [18] examines a number of aspects of use of CASE tools. It examines whether CASE tools are perceived as being useful, and if developers using CASE tools use the same methodologies and perform the same activities as developers who do not. Finally, the study points to the features of CASE tools that are being used. Surprisingly, the participants who did use CASE tools, were quite neutral when it came to the perceived usability of CASE tools. Also, the study showed that users of CASE tools on average spent more than twice as much of their time doing analysis, and considerable less time on programming, test, and maintenance, than non-users did. Unfortunately, the study did not show whether this increased time spent on analysis actually resulted in a better analysis model, or whether it was due to overhead caused by the CASE tool.

The respondents indicated that only a small amount of the functionality offered by the CASE tool was actually used. They all used the facilities for creating and editing models and diagrams but used little else. The only other functionality that was used at least “sometimes” was functionality to detect inconsistencies in diagrams, and to create documentation. The study in this way highlights that a focus on increasing the

support for creating and editing diagrams in CASE tools is appropriate, and perhaps even the most important support.

In [14], Jarzabek and Huang present the view that current CASE tools are far too focused on hard aspects of software development such as software engineering. In their opinion, softer aspects such as support for creativity and idea generation are needed if CASE tools are to gain a wide acceptance. Concretely, they believe that CASE tools should allow the developer more freedom in expressing ideas and that their environment should be more tolerant, and allow the developers to think and work “at the level of application end users”. Also the authors argue that current CASE tools are too method-oriented, and that they should provide a more natural process-oriented framework. We concur with Jarzabek and Huang, and try to support creativity, through enhanced and more flexible support for modelling.

Usage of whiteboards as a support in creative, problem-solving meetings has been studied in several contexts [3][23]. Electronic whiteboards have consequently been used as a computational extension of traditional whiteboards. A goal in systems running on electronic whiteboards has often been to preserve desirable characteristics of whiteboards such as light-weight interaction and informality of drawings [26]. To our knowledge no one has investigated this use in the setting of Computer-Aided Software Engineering. We try to extend the box of tools for electronic whiteboards with a tool supporting object-oriented modelling.

## **4 Modelling and Interpretation in Practice**

We have empirically studied the practice of modelling – and thus interpretation – in three distinct situations with three different user groups. The main results of each of the studies are given below. In each situation, the groups contained a mixture of competencies such as ethnographers, participatory designers, experienced and inexperienced developers, and end users. For a more detailed account, refer to [6].

### **4.1 The Dragon Project: Designing a New System**

This study was carried out informally during the Dragon Project [5]. The project involved participants from a university research group and a major shipping company. Development in the Dragon project took place in active collaboration with users in the sense that at least one user was co-located with the development group at any given time. Whenever modelling of major conceptual areas of the shipping domain took place the users participated actively in this. Actual modelling in these sessions almost always took place on a whiteboard. Although most information from the users was in the form of domain knowledge as verbal or written accounts, drawings were often made by the end user on, or in connection to, the diagrams on the whiteboard. User drawings were informal, in contrast to formal UML models, which described key concepts or relationships from the shipping domain. Eventually, the users nevertheless picked up parts of the UML notation and commented on, e.g., multiplicities on an association.

## 4.2 COT: Reengineering an Existing Application

We have studied a technology transfer project (COT, <http://www.cit.dk/COT>) involving a university research group and an industrial partner. The project reengineered an existing industrial application for control of a flow meter, while the inexperienced developers from the industrial partner learned object-oriented analysis and design. For the modelling, this project used a mixture of CASE tools, projectors, and whiteboards.

Simplistically, the inexperienced developers explained the existing implementation, whereas the experienced developers modelled a reengineered version. Eventually, this balance shifted as the inexperienced developers picked up larger parts of the UML and participated in the modelling. This introduced a certain number of syntactical errors in the use of the UML. These errors were either repaired, if they disturbed the shared understanding of what was modelled, or ignored, if the meaning was clear from the context. After each modelling session, photographs of the diagrams were taken for future record or for manual entry into a CASE tool. Programming was then done in an ordinary editor and the CASE tool was used to reengineer code into diagrams.

## 4.3 Mjølner: Restructuring an Existing Application

This study investigated the redesign of the Mjølner integrated development environment (<http://www.mjolner.com>). The group performing the redesign consisted of six developers with different experience in the domain of the integrated development environment. All developers had experience in object-orientation and a fair understanding of UML. Two of the developers had an in-depth knowledge of the development environment, other two developers had a knowledge of the part of the tool that they developed, and the last two developers were introduced to the software architecture of the environment while participating in the redesign.

During the redesign session, the most used artefacts were a whiteboard and a laptop. The whiteboard was used to draw the software architecture of the existing environment and to edit these drawings. The laptop was used whenever a developer needed to look at code in order to remember the actual architecture. This use took place whenever another person was at the whiteboard.

Although almost everything they drew was in actual UML notation, the notation was tweaked in three ways. First, UML did not suffice to explain certain aspects of the architecture leading to informal drawings of this. Second, the language used to implement the environment is BETA [19], which has a number of language and modelling constructs not supported by the UML. Two of these constructs, inner and virtual classes, were used heavily in the implementation, and thus the developers invented new notational elements on the fly. Third, the information drawn was filtered in the sense that often only important attributes, operations, and classes were shown.

## 4.4 Key Insights

From our analysis of the user studies a number of lessons on the co-ordinative, communicative, and collaborative aspects of object-oriented modelling can be learned. We group the insights into the three categories 'tool usage', 'use of drawings', and 'collaboration'.

**Tool Usage.** Typically, a diagram existed persistently in a CASE tool as well as transiently on a whiteboard. CASE tools were primarily used for code generation, reverse engineering, and documentation whereas whiteboards were used for collaborative modelling and idea generation. This mix caused a number of problems: Whereas whiteboards are ideal for quickly expressing ideas collaboratively and individually, they are far from ideal for editing diagrams etc. This means that in all user studies, drawings have been transferred from whiteboards to CASE tools and from CASE tools back to whiteboards.

**Use of Drawings.** Most of the drawing elements were in the form of elements from UML diagrams such as class, sequence, and use case diagrams. However, these elements were combined with non-UML elements in two forms. Either as rich "freehand" elements that explained part of the problem domain or as formal additions to the UML such as notations for inner classes or grouping. Timings from one of the user studies show that approximately 25% of the meeting was spent on actual drawing on the whiteboard. The drawing time was divided into 80% for formal UML diagrams and 20% for incomplete or informal drawings.

Another key observation is the use of filtering. Filtering was used for several reasons. First, even whiteboard real estate is limited. Second, not all parts of a diagram are interesting at all times. Third, users may employ a specific semantic filtering to decide the important elements of a diagram. Such a filtering could, e.g., be that only the name of a class is shown, or that modelling is restricted to the part of an application related to the user interface.

**Collaboration.** It has been a striking fact in our user studies that all collaborative construction of models has been co-ordinated as turn-taking. This is somewhat in contrast to other observations on shared drawing [3], but we believe it to be general for the kind of work that object-oriented modelling is about.

What was, however, not co-ordinated via turn-taking, was verbal communication and the use of other artefacts. The people engaged in the meetings, e.g., discussed among themselves while another person was drawing at the whiteboard, or they used other artefacts concurrently.

## 4.5 Implications for Tool Support for Modelling

The user studies show that computerised support for collaboration and communication in modelling is beneficial. This includes support for turn-taking and not hindering communication. Moreover, context switches in turn-taking need to be fast and transparent to users. Also, possible tool support needs to integrate with a computational environment, i.e., provide functionality such as editing of diagrams, code generation, and reverse engineering. Integration of this functionality should not hinder collaboration.

Aspects of the UML notation were too restraining for initial modelling. A tool may try to help in several ways, including supporting semi-formal, incomplete drawings and integrated informal “freehand” annotations. Moreover, a formal notation is often not completely adequate for the problem at hand. Thus, it should be possible to tweak the notation on the fly, in such a way that the notation becomes more appropriate for the problem, while still preserving the original properties of the notation.

Filtering is needed in a wide sense. A CASE tool provides a potentially unlimited workspace, whiteboards do not. CASE tools often provide filtering mechanisms such as zooming, panning, and showing/hiding attributes on diagrams. On a whiteboard, on the other hand, it is possible to, e.g., contract several relationships into a single relationship. Both kinds of filtering, visual and semantic, are useful and should be integrated.

## 5 The KNIGHT Tool

The user studies and the study of other CASE tools have been used as a basis for implementing a tool supporting collaborative modelling and implementation. This tool, the *Knight tool*, uses a large touch-sensitive electronic whiteboard (currently a SMART Board, <http://www.smarttech.com>, see Fig. 1) as input and output device. This naturally enables collaboration via turn-taking among developers and users. The interaction with and functionality of the tool is discussed in the next sections.

### 5.1 Functionality and User-Interface

A major design goal of the Knight tool was to make the interaction with the tool similar to that on an ordinary whiteboard. Therefore, the user interface (Fig. 3) is very simple: it is a plain white surface, where users draw UML diagrams using non-marking pens.

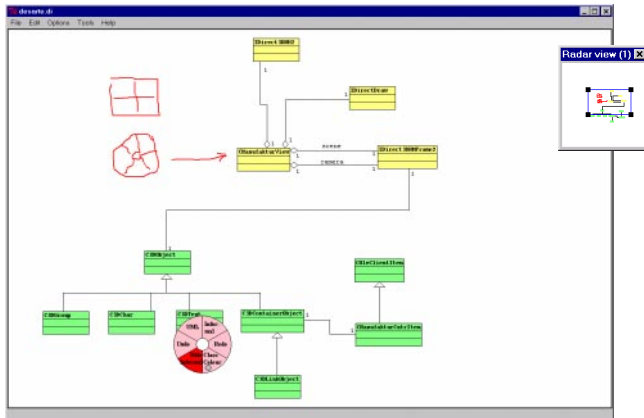
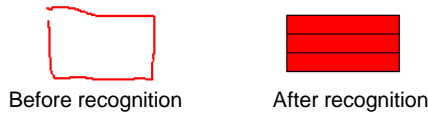


Fig. 3. Knight user interface

The interface is based on gesture input. For example, in order to create a new class, the user simply sketches a rectangle, which the tool then interprets as a class (Fig. 4). Gestures have several advantages over existing toolbars: They allow input directly on the workspace, are fast to draw, and have a low cognitive overhead.



**Fig. 4.** Recognition of the gesture for a class

Also using gestures, a user may associate two classes by drawing a straight line between them. In general, the gestures for creating UML elements have been chosen so as to resemble what developers draw on ordinary whiteboards. This directness makes the gestures easier to learn and use.

Another way of achieving an intuitive interaction is by using compound gestures [17] and eager recognition [30]. Compound gestures combine gestures that are either close in time or space to one drawing element. So for example, when users draw an inheritance relationship on an ordinary whiteboard, they normally draw a line and then an overlaying triangle. Analogously, in the Knight tool, a user first draws a line between two classes and then a triangle at the appropriate line end.

With eager recognition, the tool continuously tries to classify gestures while they are being drawn. This is used for moving: To move an element, the user draws a *squiggle* gesture on the item to be moved. When the squiggle gesture can be classified with a high confidence, feedback is given in order to show that the gesture was recognised: the item follows the pen.

The technical aspects of the gesture recognition are discussed in the “Design & Implementation” section.

**Informality vs. formality.** A continuum from informality to formality is supported in two ways. First, users may draw incomplete diagrams, such as relationships only belonging to one class (Fig. 5). The incomplete elements can later be “completed”, e.g., by attaching another class to the relationship.



**Fig. 5.** A relationship with only one class specified

Second, a separate *freehand* mode is provided. In freehand mode, the pen strokes are not interpreted. Instead, they are simply transferred directly to the drawing surface. This allows users to make arbitrary sketches and annotations as on an ordinary whiteboard (see Fig. 3 upper-left). Unlike on whiteboards, these can easily be moved around, hidden, or deleted. Each freehand session creates a connected drawing element that can be manipulated as a single whole.

**Navigation.** The tool provides a potentially infinite workspace. This allows users to draw very large models, but is potentially problematic in terms of navigating.

Generally there is a need for an easy way of navigating from one point in the diagram to another point in the diagram, and it is desirable to be able to focus on a smaller part of the diagram while preserving the awareness of the whole context. To achieve this in the Knight tool, any number of floating radar windows may be opened (Fig. 6). These radar windows, which may be placed anywhere, show the whole drawing workspace, with a small rectangle indicating the part currently visible. Clicking and dragging the rectangle pans while dragging the handles of the rectangle zooms.

Ordinary pull-down menus are not appropriate for activating the functionality of the tool given the large size of the whiteboard screen. Instead we use gestures as explained above and pop-up menus that can be opened anywhere on the workspace. The pop-up menus are implemented as *pie menus* that are opened when the pen is pressed down for a short while (see Fig. 7). For faster operation, the commands can also be invoked by drawing a short line in the direction of the pie-slice holding the desired command [16]. Furthermore, the menus are context-dependent. The left side of Fig. 7 shows the default menu, whereas the right side shows a more specialised menu that is opened when close to the end of a relationship.

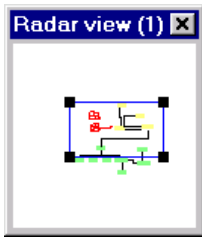


Fig. 6. Radar windows provide context awareness

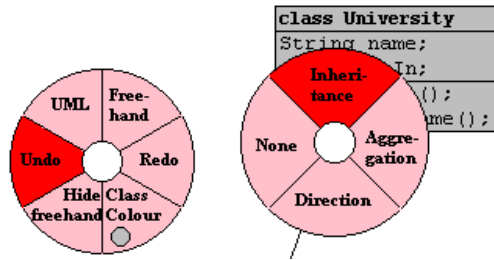


Fig. 7. Context-dependent pie menus

**Inputting text.** The tool offers five different ways of inputting text (Fig. 8). Apart from normal keyboard input these are: *Virtual keyboard* (a), *Stylus-based Gestures* (as on PDA's) (b), *Cirrin* (c) [20], and *Quikwrite* (d) [27]. They are shown in increasing order with respect to speed of text-entry and difficulty to learn. There is thus support for both casual users without any knowledge of the more specialised ways of inputting text, and advanced users who wish to enter text quickly.

### 5.2 Design & Implementation

The Knight tool is implemented in the Itcl [22] object-oriented extension of Tcl/Tk [25]. Since the implementation uses Microsoft COM [29] for tool integration, it currently only runs on the Microsoft Windows platform.

**Software Architecture.** The software architecture of the Knight tool is shown in Fig. 9 using a UML package diagram. The Knight tool and CASE tools that Knight is integrated with are separate processes. The connectors between these processes are, as discussed further below, currently implemented using Microsoft COM.

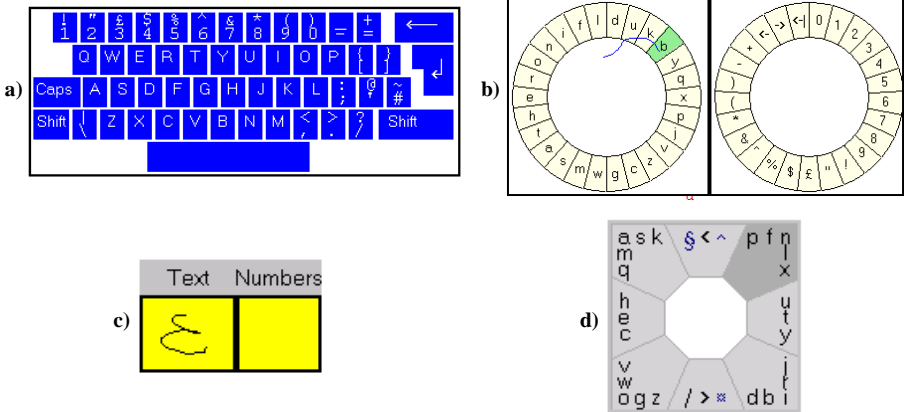


Fig. 8. Text input possibilities in Knight

Internally in Knight, connectors are either event broadcasts or direct method calls. The structure of Knight follows the *Repository* architectural pattern [32]. The packages surrounding the Repository are *readers* and *readers/writers*. They work independently on the repository.

The *Repository* is a repository of UML diagrams. Basically, the class structure is a subset of the UML metamodel. When changes occur in the diagram, Observers [10] on the diagram are notified.

The *Radar* is observing the Repository and it gives an overview by reading and representing the state found in the diagrams.

The *Workspace* both reads and writes from and to the diagrams to display and edit these. Writing in the Repository is implemented using a combination of the Composite and the Command patterns [10], providing undo/redo functionality.

The *CASE Tool Integrator* is a Mediator [10] between the Knight tool and multiple CASE tools (see below).

**Gesture Recognition.** Rubine's algorithm [30] is used for gesture recognition. The main advantage of this algorithm is that it is relatively easy to train: For each gesture to be recognised, it must simply be provided with a number of prototypical examples of the gesture. Based on these examples, the algorithm then computes a representative vector of features. Examples of features are the total length of a gesture and the total angle traversed by a gesture. Subsequently, these representative vectors can be compared to a vector computed from a user's input, and the most resembling gesture can be chosen based on a statistical analysis.

A few circumstances complicate the gesture recognition a little. First, different types of input devices have different physical characteristics. Thus, the recogniser should ideally be trained once per type of input device. Second, Rubine's algorithm requires that different types of gestures are distinguishable with respect to the feature vector. We handle this by using compound gestures, e.g., when drawing different types of relationships, thus reducing the number of gestures to be recognised. Third, all users do not draw, e.g., a rectangle starting in the same corner and in the same direction. In order to preserve the intuitiveness of and the resemblance to ordinary whiteboards, the gesture recogniser must thus be able to recognise a rectangle starting in all four corners going both clockwise and counter-clockwise. To lessen the burden

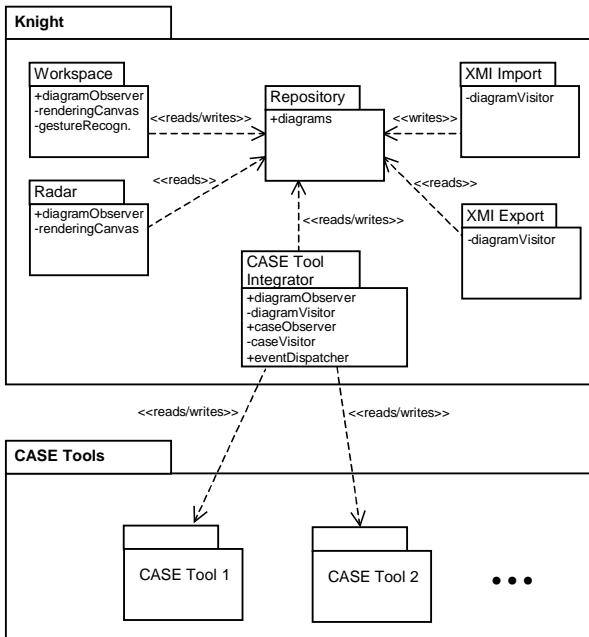


Fig. 9. Software architecture of the Knight tool

of training the recogniser with all these variations, we have implemented a script that takes, e.g., a set of rectangle examples all drawn starting in one corner and going in one direction and then permutes these by rotating and mirroring to obtain eight combinations. In this way the full gesture set of 49 different gestures can be achieved from only 9 sets of examples.

**Integration with other CASE tools.** There is a need to provide common CASE tool functionality such as code generation and reverse engineering. Different tools support different tasks well and users choose to use different kinds of tools based on the work at hand. Thus, we have chosen an integration strategy instead of building a full-featured CASE tool.

We are implementing both batch and incremental integration with CASE tools. Batch integration is currently achieved in two ways: using *XML Metadata Interchange* format (XMI [24]) and using component technology. To implement the XMI exchange, we used a DTD based on the UML metamodel. Given this DTD, we implemented support for importing models from and exporting models to XML files following its grammar. Since the DTD is generated as specified in the XMI standard, it enables us to share models with other CASE tools that support UML and XMI.

Incremental integration is done for two reasons. First, batch integration may lose information and rely on other tools' interpretation of information. Second, we wish to integrate with CASE tools that support incremental round-trip engineering [4]. For this to be useful, incremental updates are needed so that code is always available and changes in code are immediately reflected in diagrams. The incremental integration is currently done via Microsoft COM [29].

The details of the integration are described in a companion paper [7].

## 6 Evaluation and Future Work

### 6.1 Evaluation

We have performed qualitative evaluations of the Knight tool in use. The primary objective of the evaluations has been to evaluate Knight in real work settings. Typically, a facilitator introduced the tool briefly. Following this, each subject was given a chance to try the interaction of Knight and learn the gestures for manipulating diagrams. The facilitator also helped if the subjects had problems using the tool in their work.

After the introduction, the subjects used Knight to work on their current project. While the subjects worked on their project, the use was videotaped and notes were taken. After the sessions, the subjects were interviewed.

The evaluations were all positive and showed that the tool was useful in the design situation. Also, the subjects considered the tool to be a better enabler for both collaboration and creativity than traditional CASE tools. They especially liked the large electronic whiteboard's collaboration support and the tool's interaction style, with its combination of informal and formal elements.

A number of minor problems surfaced. A few of the gestures were hard to learn by some of the participants, although they seemed to learn them as they used the tool further. Also the integration of freehand, formal, and informal elements was generally construed as useful (see Fig. 10), but further functionality was desired, e.g., to associate a freehand element and a formal element to each other.

In summary, the evaluations validate our general approach to supporting collaborative modelling. The issues raised as results of the evaluations were mostly smaller interaction problems, and these were outweighed by the tool's advantages. The users especially found that the tool's lighter, more intuitive interaction allowed for more creativity and that the tool less frequently than ordinary CASE tools caused breakdowns in the modelling.

### 6.2 Future Work

**Longitudinal Studies of Actual Use.** The tool should be evaluated in real settings over a longer period of time, preferably for the duration of a whole project. Further evaluation could also include a comparative, quantitative study of the Knight tool's impact on the quality of the resulting design and the amount of time spent using the tool.

**Distributed Use.** Considering distributed use of the Knight tool suggests some interesting possibilities. The obvious possibility is to connect two electronic whiteboards in different locations and to have designers work on the same model synchronously. But other combinations may also be possible. For example, one might choose to have a different view on a model via an ordinary PC located in the same meeting room as the electronic whiteboard.

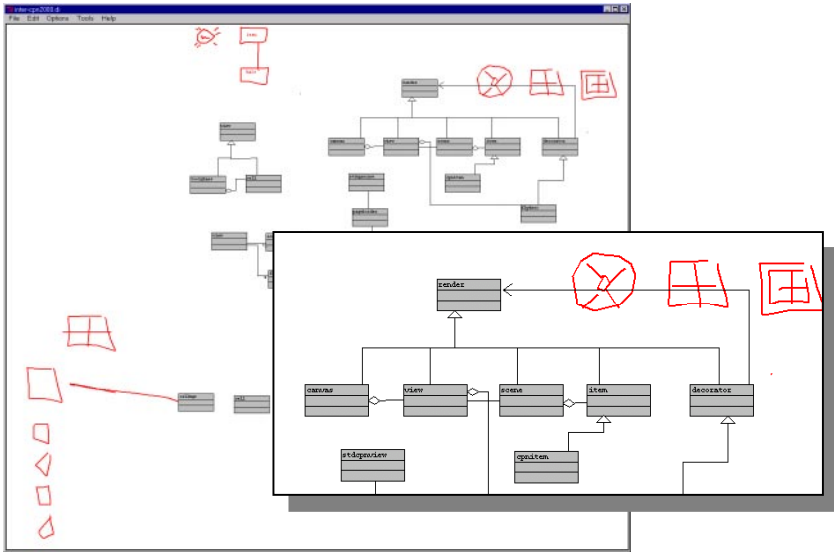


Fig. 10. Diagram produced during an evaluation session (with part of the diagram emphasised)

**End User Customisation.** An area of research that we so far have left more or less untouched is the ability for the user to customise the tool in a number of ways. Currently, we are working with the idea of a *personal pen* that holds several settings for the user. In this way, a user who chooses a specific pen will use the settings of the pen, such as gesture set, freehand/UML mode, or current colour, when interacting with the electronic whiteboard. A bit further along the way is the possibility for the user to customise the notation, ultimately “on the fly”. Simple customisations of this type include changing the appearance of drawings or adding user-defined stereotypes to diagrams. A more elaborate tailoring scheme in which the user actively models on the metamodel of the tool may be possible. Such a tool could be considered a light-weight meta-CASE tool [8][9].

**Generalisation.** Obviously, full support for all the diagram types of the UML is desirable. Moreover, many of the observations that we have made of object-oriented modelling seem to be true of other kinds of formal modelling as well. This suggests that the Knight tool could support other kinds of diagramming with the same basic interaction.

Other generalisations are also possible: Even though the gesture based interaction has been designed for use on an electronic whiteboard, the interaction seems direct and natural and it would be worthwhile investigating its use on an ordinary PC with other input devices, e.g., tablets, mice, or trackballs. We would also like to explore other more exotic input/output devices such as tablets with built in LCD displays, connected PDAs communicating with a large screen, or electronic extensions of traditional whiteboards such as Mimio [33].

## 7 Conclusion

Support for modelling is one of the major advantages of object-oriented development. Through the conceptual framework underlying object-orientation, both the problem domain of the system and the system to be built can be understood and formulated.

It is often advantageous to have tool support for modelling. Tool support can, e.g., aid in creating and editing models, in checking the syntax and semantics of models and in generating code from the models. Studies have shown, however, that existing *CASE tools* are rarely used. We believe that part of the explanation for this lies in the poor support for intuition, flexibility, and collaboration in *CASE tools*. To experiment with and to prove the advantages of such support, we have designed and implemented a tool, Knight, which complements existing *CASE tools*. The Knight tool provides a direct and fluid interaction that in many ways resemble the interaction with regular whiteboards, it provides support for collaboration in its large shared workspace, and it allows for more flexibility by supporting models with both informal and incomplete elements.

The tool has so far been successfully evaluated in qualitative experiments that validated the basic design. Further evaluation and studies of use are to be performed, but we believe that an extension of current *CASE tools* with support for creativity, flexibility and collaboration as found in Knight can ultimately help in improving the overall quality of development projects.

**Acknowledgements.** We thank Ole Lehrmann Madsen for comments that improved this paper. This project has been partly sponsored by the Centre for Object Technology (COT, <http://www.cit.dk/COT>), which is a joint research project between Danish industry and universities. COT is sponsored by The Danish National Centre for IT Research (CIT, <http://www.cit.dk>), the Danish Ministry of Industry and University of Aarhus.

## References

1. Aaen, I., Siltanen, A., Sørensen, C., & Tahvanainen, V.-P. (1992). A Tale of two Countries: CASE Experiences and Expectations. In Kendall, K.E., Lytinen, K., & DeGross, J. (Eds.), *The impact of Computer Supported Technologies on Information Systems Development* (pp 61-93). IFIP Transactions A (Computer Science and Technology), A-8.
2. Blomberg, J., Suchman, L., & Trigg, R. (1994). Reflections on a Work-Oriented Design Project. In *Proceedings of PDC '94*, pp. 99-109, Chapel Hill, North Carolina: ACM Press.
3. Bly, S.A. & Minneman, S.L. (1990). Commune: A Shared Drawing Surface. In *Proceedings of the Conference on Office Information Systems* (pp. 184-192). ACM Press.
4. Christensen, M. & Sandvad, E. (1996). Integrated Tool support for design and implementation. In *Proceedings of the Nordic Workshop on Programming Environment Research*, Aalborg, May 29-31.
5. Christensen, M., Crabtree, A., Damm, C.H., Hansen, K.M., Madsen, O.L., Marquardsen, P., Mogensen, P., Sandvad, E., Sloth, L., & Thomsen, M. (1998). The M.A.D. Experience: Multiperspective Application Development in Evolutionary Prototyping. In *Proceedings of ECOOP'98*, Bruxelles, Belgium, July, Springer-Verlag, LNCS series, volume 1445.

6. Damm, C.H., Hansen, K.M., & Thomsen, M. (2000). Tool Support for Cooperative Object-Oriented Design: Gesture Based Modeling on an Electronic Whiteboard. In *Proceedings of Computer Human Interaction (CHI'2000)*. Haag, The Netherlands, 2000.
7. Damm, C.H., Hansen, K.M., Thomsen, M., & Tyrsted, M. (2000). Tool Integration: Experiences and Issues in Using XMI and Component Technology. In *Proceedings of TOOLS Europe'2000*. Brittany, France.
8. Ebert, J., Süttenbach, S. & Uhe, I. (1997). Meta-CASE in Practice: a Case for KOGGE. In Olive, A. & Pastor, J. A. (Eds.): *Advanced Information Systems Engineering, Proceedings of the 9th International Conference, CAiSE'97*, pp. 203-216, Barcelona, Catalonia, Spain, June 16-20, LNCS 1250, Berlin: Springer.
9. Englebert, V. & Hainaut, J.-L. (1999). DB-MAIN. A Next Generation Meta-CASE. In *Information Systems*, pp. 99-112, 24 (2).
10. Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1995). *Design Patterns. Elements of Reusable Object/Oriented Software*. Addison-Wesley.
11. Greenbaum, J. & Kyng, M. (1991). *Design at Work: Cooperative Design of Computer Systems*. Hillsdale New Jersey: Lawrence Erlbaum Associates.
12. Hughes, J., King, V., Rodden, T., & Andersen, H. (1994). Moving Out of the Control Room: Ethnography in System Design. In *Proceedings of CSCW'94* (pp. 429-439). Chapel Hill: ACM Press.
13. Iivari, J. (1996). Why Are CASE Tools Not Used? In *Communications of the ACM*, 39(10).
14. Jarzabek, S. & Huang, R. (1998) The Case for User-Centered CASE Tools. In *Communications of the ACM*, 41(8).
15. Kemerer, C.F. (1992). How the Learning Curve Affects CASE Tool Adoption. In *IEEE Software*, 9(3).
16. Kurtenbach, G. (1993). *The Design and Evaluation of Marking Menus*. Unpublished Ph.D. Thesis, University of Toronto.
17. Landay, J.A. & Myers, B.A. (1995). Interactive Sketching for the Early Stages of User Interface Design. In *Proceedings of CHI'95*, 45-50.
18. Lending, D. & Chervany, N.L. (1998). The Use of CASE Tools. In Agarwal, R. (Eds.), *Proceedings of the 1998 ACM SIGCPR Conference*, ACM.
19. Madsen, O.L., Møller-Pedersen, B., & Nygaard, K. (1993). *Object-Oriented Programming in the BETA Programming Language*, ACM Press, Addison Wesley.
20. Mankoff, J. & Abowd, G.D. (1998). Cirrin: A Word-Level Unistroke Keyboard for Pen Input. In *Proceedings of the ACM Symposium on User Interface Software and Technology (UIST)*.
21. Martin, J. & Odell, J.J. (1998). *Object-Oriented Methods. A Foundation*. Second Edition. Prentice Hall.
22. McLennan, M.J. (1993). [incr Tcl]: Object-Oriented Programming. In *Proceedings of the Tcl/Tk Workshop*, University of California at Berkeley, June 10-11.
23. Moran, T.P., Chiu, P., Harrison, S., Kurtenbach, G., Minneman, S., & van Melle, W. (1996). Evolutionary Engagement in an Ongoing Collaborative Work Process: A Case Study. In *Proceedings of CSCW'96*, 150-159.
24. Object Management Group (1998). *XML Metadata Interchange (XMI)*, document ad/98-07-01, July.
25. Ousterhout, J. (1994). *Tcl and the Tk Toolkit*. Addison-Wesley.
26. Pedersen, E.R., McCall, K., Moran, T.P., & Halasz, F.G. (1993). Tivoli: An Electronic Whiteboard for Informal Workgroup Meetings. In *Proceedings of INTERCHI'93*, 391-398.
27. Perlin, K. (1998). Quikwriting: Continuous Stylus-Based Text Entry. In *Proceedings of the ACM Symposium on User Interface Software and Technology (UIST)*.

28. Reiss, S.P. (1996). Software Tools and Environments. In *ACM Computing Surveys*, 28(1), CRC Press, March 1996.
29. Rogerson, D. (1997). *Inside COM. Microsoft's Component Object Model*. Microsoft Press.
30. Rubine, D. (1991). Specifying Gestures by Example. In *Proceedings of SIGGRAPH'91*, 329-337.
31. Rumbaugh, J., Jacobson, I., & Booch, G. (1999). *The Unified Modeling Language Reference Manual*. Addison Wesley.
32. Shaw, M. (1996). Some Patterns for Software Architectures. In Vlissides, Coplien, Kerth (Eds.), *Patterns Languages of Program Design 2*. Addison Wesley, 1996.
33. Yates, C. (1999). Mimio: A Whiteboard without the Board. In *PC Computing*, June 28.