

# Supporting Several Levels of Restriction in the UML

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

Department of Computer Science, University of Aarhus,  
Aabogade 34, 8200 Aarhus N, Denmark

`{damm,marius,miksen,tyrsted}@daimi.au.dk`

**Abstract.** The emergence of the Unified Modeling Language (UML) has provided software developers with an effective and efficient shared language. However, UML is often too restrictive in initial, informal, and creative modelling, and it is in some cases not restrictive enough, e.g., for code generation. Based on user studies, we propose that tool and meta-level support for several levels of restriction in diagrams and models is needed. We furthermore present a tool, Knight, which supports several levels of restriction as well as ways of transferring models from one level of restriction to another. This approach potentially increases the usability of the UML, and thus ultimately leads to greater quality and adoption of UML models.

## 1. Introduction

Modelling is an important activity in object-oriented development [18]. For visual modelling, the Unified Modelling Language (UML; [21]) has been widely accepted. Much research has focussed on extending, formalising, or refining the UML [8][9][3]. Another important research question is, however, to what extent the UML supports visual modelling as it is being carried out in actual practice? This paper seeks to answer this question, although not in a general sense, as we will mostly discuss the existing practice of initial, creative, and collaborative problem domain modelling. The question will be addressed from two perspectives: One focussing on the UML itself as a modelling language, and one focussing on the support by current tools and technologies for UML modelling.

The UML is widely used by practitioners, but our previous studies suggest that certain parts of the UML are too restrictive with respect to what can be described [5]. For example, while modelling, practitioners often draw incomplete relationships or represent concepts (classes) by illustrative freehand drawings.

Likewise, tools and technologies for UML-based modelling are in wide use. It is a striking fact, however, that Computer Aided Software Engineering (CASE) tools are not widely used and adopted [1][13][15][17]. Some of the explanations are that current CASE tools focus too much on hard aspects of software engineering, while

support for soft aspects such as collaboration, creativity, and idea generation is lacking [14].

This paper takes a closer look at the practice of initial, creative, and collaborative problem domain modelling through a number of concrete projects. Based on these, we argue for the usefulness of several levels of restriction of the UML metamodel for the usefulness of transferring a model element from one level of restriction to another. Furthermore, we present and discuss a tool, *Knight*, which illustrates the main ideas. The main characteristics of the Knight tool are: Support for a direct and fluid use, support for collaboration, and support for different levels of restriction in both model and presentation, ranging from incomplete, freehand UML diagrams to UML models close to code.

The rest of this paper is structured as follows. First, we analyse current modelling practice and its application of the UML and tools. Second, we present the Knight tool, which tries to overcome some of the problems uncovered in the analysis. Then, related and future work is discussed, followed by conclusions.

## 2. Current Modelling Practice

We will focus our examination of the UML's support for modelling practice on studies of two large development projects and on our own personal experience as system developers. The first project, Dragon [4], involved a research group and a large, globally distributed shipping company. The goal of the project was to implement a prototype of a global customer service system for the company. This was realised over a one and a half year period by the development of a series of successful prototypes. In this project, three of the authors participated actively and observed ongoing work.

The second project, Danfoss (<http://www.cit.dk/COT/case2-eng.html>), was concerned with implementing an embedded control system for flow meters. The project lasted a year and involved experienced developers from a research group and engineers from a private company. One of the authors participated in this project. This involved formal observations of work and active participation. Thus our approach was a mix of (ethnographic) observations and active involvement in the project studied [11].

Both projects used an iterative object-oriented approach to system development. Throughout development, UML was used on whiteboards and in CASE tools to visualise an emerging understanding of the problem and solution domains.

In order to focus the discussion, the next section presents three representative scenarios distilled from the studies. These scenarios will then subsequently be used as a basis for analysis of current modelling practice.

## 2.1 Scenarios

Christina and Tom are two developers working on an administrative system for a university. Lisa is a problem domain expert knowledgeable in the domain of university administration.

**Scenario One: Using Incomplete and Freehand Elements.** Christina and Tom have invited Lisa to a session in which they will start to model the student administration part of the administrative system. Since Christina and Tom have little knowledge of this part of the problem domain, Lisa explains her understanding of student administration. While doing this, she makes heavy use of a whiteboard, and she draws freehand diagrams of the study structure at the university. Christina and Tom, in collaboration with Lisa, then try to transform the verbal and graphical account to a class model, which contains the relevant aspects of the problem domain. Sometimes, Christina or Tom draws an incomplete association or generalisation, since it is not yet clear which classes are involved.

**Scenario Two: Shifting Between Levels of Completion.** Christina and Tom continue working on the diagram after the initial session with Lisa. Because they work in an iterative manner, they already have a running version of the overall administrative system, and they now want to incorporate the newly discovered student details into the system. Since the previous session was done in a flexible and creative manner, the diagram contains a lot of non-UML elements such as incomplete relationships and freehand drawings. They implement the model in code. In the process they transform the problematic parts in simple ways, e.g., by converting freehand drawings to verbal descriptions in the form of comments, and by either deleting the incomplete relationships or creating stub classes for the dangling ends. During implementation, they discover conceptual errors, which they fix in the code. For later sessions with Lisa, these changes will have to be consistent with the original diagram.

**Scenario Three: Using a UML Tool.** The development team uses a traditional CASE tool for various software engineering tasks such as code generation, configuration management, and documentation. Christina and Tom thus need to add the student administration model and diagram that they modelled with Lisa to their UML repository. In order to do that, they have taken photos of the whiteboard with the diagram in various stages of completion. Next, they create image files for the important freehand drawings. They then redraw the diagram in their CASE tool by hand, making sure that all parts of the model conform to the UML. References to the photos are then connected to relevant parts of the diagram.

## 2.2 Analysis

The three scenarios give examples of some of the obvious characteristics of practical UML modelling. This includes the use of different types of information as well as the shifts between these.

Scenario one shows that freehand drawings are used in the early phases of modelling and that sometimes open-ended modelling using, e.g., freehand drawings and incomplete relations are appropriate. At other times, a close connection to code is essential, and this puts some restrictions on the structure of the models.

During the lifetime of a model, it inevitably becomes gradually more stable and complete. Initially, the developers' knowledge of a domain is limited, and the focus is therefore on understanding the overall structures of the domain. Many issues are unclear at this point, even some of the overall structures, and it is of less importance to understand the details of the constituent parts, e.g., attributes and methods and their types and parameters. It is thus necessary to do more expressive, but less restrictive, modelling.

At some point, the model is sufficiently mature to be turned into code. It is, however, not satisfactory to have to wait until the model is complete, before turning it into code, and hence the question arises: how to turn a model with incomplete elements into code? The solution adopted in the scenarios was to fill the holes manually with default values, e.g., setting potential attributes' types to "void". By doing this, we lose information about the model, and at a following analysis/design session, the model looks different from the previous session. In the later phase, it is thus necessary to have a more restricted model, and therefore also less expressive modelling abilities.

As a project progresses, the models get more elaborated in some areas, but new areas are also investigated, like the student administration in the scenarios. This means that the same model can be stable and detailed in some areas, while being initial and containing incomplete elements in other areas.

As is evident in the scenarios, modelling is often done in collaboration, with several developers working together and often also involving domain experts. When people with different competencies work together, these people use different means to express their ideas. In scenario one, Lisa was not a computer expert, so she used the whiteboard to draw sketches. Christina and Tom, on the other hand, focused on creating a class diagram and thus used the UML notation on the whiteboard.

Also, scenario three highlighted the large gap between whiteboards and traditional UML tools. The whiteboard contained many elements that had to be modified to fit in the UML tool, and the transfer itself was manual and cumbersome. Moreover, since development was iterative, such shifts will often occur.

### **3. UML Tool Support for Modelling Practice**

The fact that the UML metamodel disallows practices used in modelling suggests that the UML metamodel should be modified. As the analysis in section 2.2 shows, the way of expressing knowledge in a model varies. Sometimes, the flexibility to create freehand drawings and incomplete elements is needed. At other times, the model should be sufficiently restricted to allow for direct code generation.

One solution would be to make the UML metamodel more expressive so as to allow models to contain all kinds of elements needed as well as all kinds of relationships between those elements. However, in order for a model to be useful for,

e.g., code generation, it is necessary that the model is restricted only to contain elements that have a straightforward translation to programming language constructs. As an example, Bunse and Atkinson propose the *Normal Object Form* (NOF) [3], which is a version of the UML metamodel that is closer to object-oriented programming languages and therefore better supports code generation from models. Thus, there *should* be support at the metamodel level for different levels of restriction.

Instead of having several more or less incidental metamodel, the UML metamodel itself could contain different *levels*. The levels of the UML metamodel should cover the range from very expressive models to models close to code. There should, at least, be the following levels:

- A flexible level that supports freehand drawings, incomplete elements, etc.
- A level that allows the models to be analysed (this could be the current UML metamodel)
- A level, in which models are close to object-oriented programming languages, so that code generation is possible (this could be the NOF)

Ideally, the UML should support a continuum of metamodels at different levels of restriction, but, realistically, a fixed number of standardised levels should be incorporated in the UML

For each different metamodel, there will be constructs that relate to models (“*meta-model*”) and constructs that relate to diagram or presentation (“*meta-presentation*”). An unrestricted meta-model will, e.g., allow attributes with no type information, and an unrestricted meta-presentation will, e.g., allow expressive freehand drawings to be used. It is interesting to consider how these varying meta-models and meta-presentations may be coupled. At times, a very restrictive meta-model and a very restrictive meta-presentation will be useful, such as when exchanging models and diagrams between organisations. At other times, there will be a need for a very expressive meta-model coupled to a very expressive meta-presentation, such as when brainstorming. Also, a coupling between a restrictive meta-model and an expressive meta-presentation is beneficial: If domain experts are involved in modelling, a presentation much like that on a whiteboard is useful.

Such a modification of the UML that supports different levels of restriction will help modelling practice. However, there is also a need for tools to support those levels. The next section describes a prototype of a tool, the *Knight tool*, which is designed to support flexible and creative modelling. We have used the Knight tool to experiment with different levels and the transition between those levels.

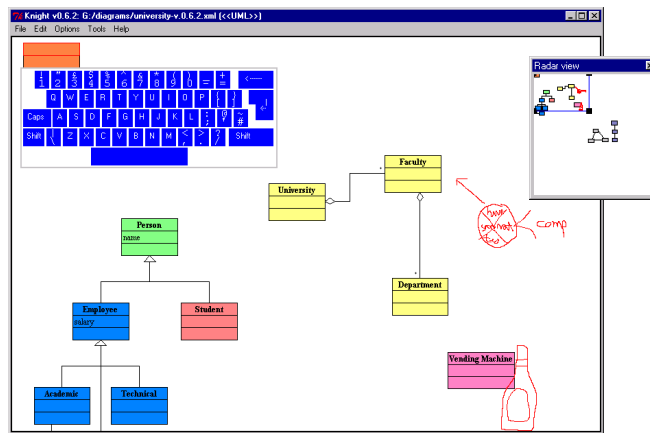
### 3.1 The Knight Tool

The Knight tool is a tool for the initial analysis and design in object-oriented software development. With a lightweight interface, it combines freehand drawings with UML diagrams. We have found that this is very important for the modelling in early phases of object-oriented software development [5]. Problems of today’s Windows, Icons, Menus, and Pointers (WIMP) interfaces are many and well known [2]. The Knight

tool tries to overcome such problems by basing the interaction on gestures, marking menus, and a whiteboard metaphor.

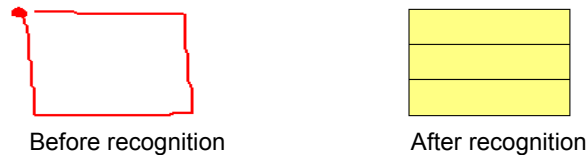
The Knight Tool is implemented in Itcl [19], an object-oriented extension of Tcl/Tk [21]. For further information on the Knight tool, refer to <http://www.daimi.au.dk/~knight> or [5][6][7].

**Interaction with Few Breakdowns.** The whiteboard metaphor is the cornerstone of the Knight tool (Figure 1): The user interface is initially a blank, white surface on which the user draws strokes. These strokes are then recognised as gestures using Rubine's algorithm [24] and interpreted as commands.



**Figure 1. User interface of the Knight tool**

The gestures resemble what is typically drawn on an ordinary whiteboard. For example, the gesture for drawing a class is a box, as shown in Figure 2.

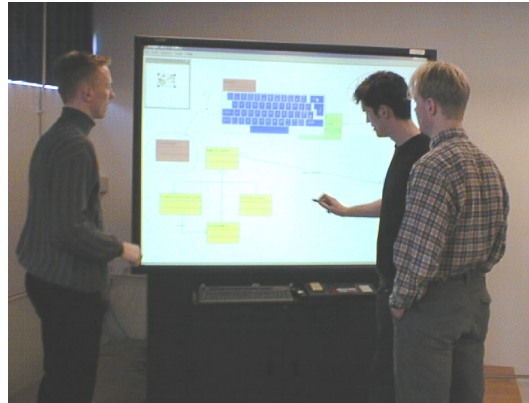


**Figure 2. Gesture recognition in the Knight tool.**

In addition to the gestures for creating UML elements, there are also gestures for moving, editing, and deleting elements. Less common commands may be accessed through a context-sensitive pie menu, which is activated by pressing on the drawing surface for a moment.

The rationale for using gestures is to make the interaction fast and fluid: few breakdowns and focus shifts should be caused by the interaction with the tool. So far, evaluations with users have supported this claim [5].

**Collaboration Support.** The Knight tool may be used on an ordinary workstation with a mouse or a tablet. If used on an electronic whiteboard, collaboration between a number of developers and domain experts is supported (Figure 3).



**Figure 3. Collaborating using the Knight tool on an electronic whiteboard**

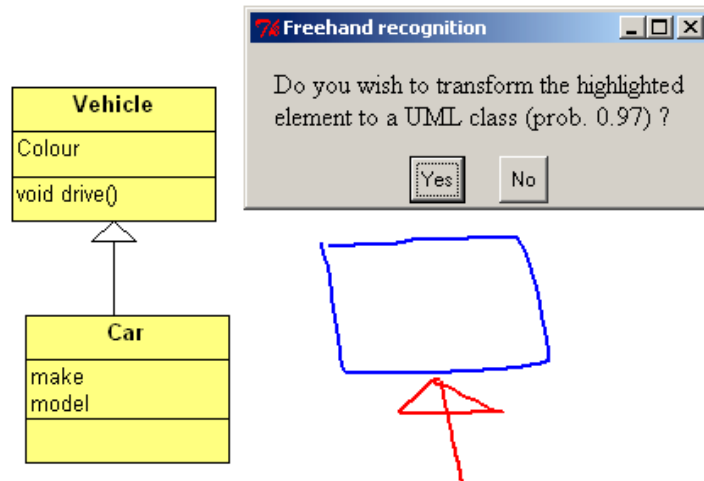
We have used the Knight tool on several types of electronic whiteboards, including the SMART Board (<http://www.smarttech.com>) and the Mimio (<http://www.mimio.com>). The SMART Board (Figure 3) is a large, touch-sensitive surface, whereas the Mimio is a small device, which can be attached to ordinary whiteboards. Given the Knight tool and a projector, both technologies give a total interaction much like that on a whiteboard.

### 3.2 Knight's Support for Varying Levels of Restriction

From the analysis in the previous section, we derive that a UML tool should support metamodels with differing levels of restriction, since modelling is not done on one level of abstraction at all times. Also shifts between levels should be supported. This section discusses the current support for varying levels of restriction in the Knight tool.

**Shifting Between Levels of Restriction.** As identified above, there is a need for shifting between the levels of expressiveness and restriction. A user may use a tool in three ways to transform a model, so that it conforms to a certain restriction level.

- *Automated.* The tool may automatically transform the model.
- *Guided.* The tool may identify the elements that do not conform to the new level of restriction, and it may provide the user with advice on how each element may be transformed. The user can then decide which transformation is appropriate.
- *Manual.* The user has to transform the model in the usual way, i.e., without any special assistance from the tool.

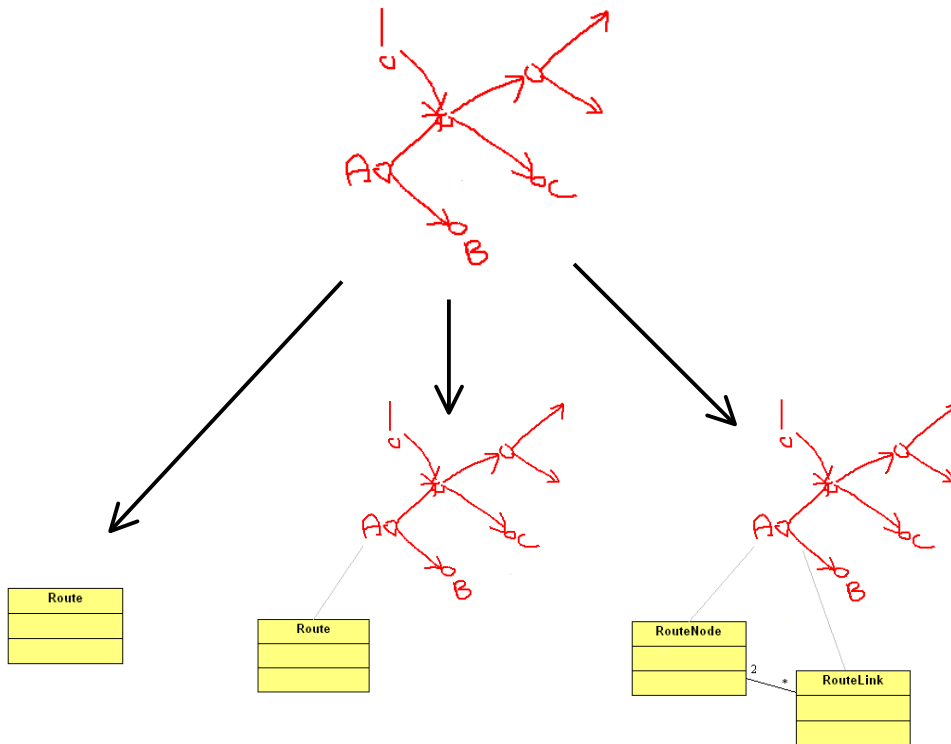


**Figure 4. Guided transformation of freehand elements**

The Knight tool implements all three kinds of support to varying degrees. Initially, strokes are conceptually elements in an unrestricted meta-model and -presentation. The default behaviour of Knight is to automatically transform these into UML elements. In this way, a restriction is made in the presentation as well as in the model. During integration with other CASE tools, an automated restriction is also necessary. As an example, freehand drawings are converted into comments, and incomplete elements are omitted from the restricted model. These are all examples of automated transformations.

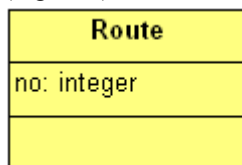
Guided transformations are also supported in several ways. The user can ask Knight to identify the elements that are illegal in the more restricted meta-model. These will be marked graphically, and the user may then request guidance on how to transform the elements. For example, an incomplete association turns red, and in order to make it complete, the user can either delete the association or move the dangling end to a class. When the illegal elements are all removed, the model has been restricted. Furthermore, the user can ask the tool to guide a transformation where freehand elements are restricted into UML elements. The tool will then iterate through the individual marks in the drawing and present the user with a possible restriction (see Figure 4).

Figure 5 shows examples of manual transformations. In the example, a freehand drawing showing important details of the concept of a "Route" has been drawn. Depending on the context, the user may use different transformations. If the drawing is no longer significant, the user may choose to transform the drawing into a class and give it the name "Route". If it is still important to be able to refer to the drawing, the user may transform the drawing to a class and a relation to the drawing, thus using the drawing as an icon for the class. Likewise, the user may use the drawing to document a more elaborate model of a route.



**Figure 5. Restrictive transformations of a freehand drawing**

**Preserving the Look of Diagrams.** Knight supports the creative phases of object-oriented modelling, because it allows users to create expressive models. For example, a UML class may not have a name yet, and only some of the attributes may be known and specified. However, in Knight and other UML tools, the elements often *look* finished, even when they are not (Figure 6).



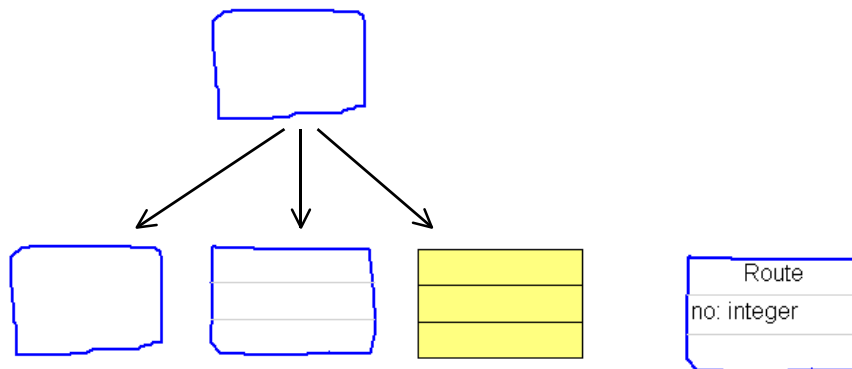
**Figure 6. Unfinished elements may look finished**

There is evidence that the appearance of, e.g., a diagram or a user interface influences how people perceive it [16]. A sketchy look of a diagram makes people subconsciously believe that the diagram is not finished, and hence they are more willing to suggest changes to it.

Knight supports preservation of the look of elements. An element can be displayed in three different ways:

- it may look exactly the way it was drawn, i.e., it is not transformed,
- it may have a semi-transformed look, which is the same for all elements of the same category, or
- it may have a transformed look, e.g., the usual UML notation.

These three looks are exemplified for a UML class in Figure 7a. In Figure 7b, the Route class from Figure 6 is displayed with an untransformed look, which suggests that it is unfinished.



**Figure 7. a) A class may be displayed in three ways in Knight,  
b) the Route class is not finished**

It is possible for different elements in the same model to be displayed differently. In fact, as discussed in section 2.2, it is often the case that certain parts of a diagram are detailed and finished, while other parts have not yet been elaborated on. Displaying the elements corresponding to whether they are finished or not can show the situation to the users.

### 3.3 The Knight Tool at Work

The Knight tool was designed to support the modelling practice found in actual software development projects. In order to demonstrate *how* the use of Knight on an electronic whiteboard accomplishes this, this section will describe what the three scenarios from section 2.1 could look like, if the Knight tool was used.

**Scenario One.** Instead of using a traditional whiteboard, Lisa, Christina, and Tom use Knight. Because Knight supports freehand drawing, Lisa is able to illustrate the study structure directly in Knight. Christina and Tom gradually transform Lisa's freehand drawings to corresponding classes, while retaining the original drawings in the diagram.

**Scenario Two.** Christina and Tom have to transfer the model created together with Lisa to the overall administrative system. The model contains non-UML elements, but Christina and Tom use Knight's guidance to remedy some of the problematic

elements. The rest of the non-UML elements are handled automatically by Knight, which deletes some incomplete associations and adds default types to some attributes.

**Scenario Three.** Because the model was created directly in Knight, there is no need to capture the contents using a camera. Freehand elements may, moreover, be connected to UML elements directly. Also, the shifts to and from ordinary whiteboards are eliminated through the use of Knight on an electronic whiteboard.

Even though the scenarios highlight the positive features of Knight, there is room for a lot of improvement. Problems and plans are discussed in the next session.

#### **4. Related & Future Work**

As mentioned in the introduction, Jarzabek and Huang [14], among others, explain the low adoption of CASE tools with that they are too concerned with software engineering aspects of development. Instead, aspects such as creativity, flexibility, and idea generation should be focussed on and more widely supported. We concur that focussing on softer aspects of system development in CASE tool development may be a way for CASE tools to gain wider acceptance. This is the very basis for our work on Knight and is what we will continue working on also in the context of other formal notations than the UML. A very important means of ensuring that Knight supports this kind of work will be to set up longitudinal studies of Knight in use. We are currently trying to do this at several Danish companies. Although many of the basic mechanisms for creative modelling are present in Knight, we suspect that further refinements are needed. For example, it should be possible to incorporate other types of media than drawing such as photographs, videos, and sound in a seamless manner. Also, the use of and transition between different levels of restrictions will have to be refined.

In [12], Haake et al. present a taxonomy for categorising information-structuring systems based on the perspectives of the user and the system. The taxonomy has two dimensions. The first dimension describes how explicitly the user specifies the types of objects. Does the user, e.g., state that a specified object is of a certain type or not? The second dimension describes how the system represents the data. Does the system, e.g., retain or discard the type information? In our default transformation when users draw UML elements by gesturing, the user does not specify that it is, e.g., a class that is drawn. However, the system infers and represents the class via type information. In this sense there is no difference between selecting a class icon from a palette and drawing a gesture for a class. In the case in which the user draws a freehand drawing and then later transforms it manually, the situation changes from one in which the user has a typed representation of an element, whereas the system does not, to a situation in which both user and system has a typed representation. We concur with Haake et al. that this provides the users with a flexibility to support the creation of objects without the overhead of deciding types immediately. The focus of this paper has been on how presentation and model in UML has been represented. This is in

many ways system-centric, and one may envision a conceptualisation of levels of restriction in which cognitive aspects of users are taken into account.

In [3], Bunse and Atkinson propose the *Normal Object Form* (NOF), which is closer than UML to object-oriented programming languages. The NOF is basically a subset of UML with certain predefined extensions, and with some additional constraints on the structure of models. The authors make a distinction between *refinement* and *translation* of a UML model. The former involves adding details to the model, while remaining within the UML, and the latter is a transformation of the UML model to an object-oriented programming language. Thus, a model should be gradually *refined* until it satisfies the NOF criteria, and then it should be *translated* to code. The NOF is an example of a useful version of the UML with a specific purpose, and thus it supports our claim that there should be multiple versions of the UML. Also, models should be gradually transformed from one form (current UML, expressive) to another (NOF, restrictive), and this process could benefit from guidance or automation of a tool. The *refinement patterns* mentioned in the paper could play a role in the process, be it guidance or automation. Whereas NOF is a very restrictive version of the UML metamodel, what we propose is that the default, or lowest, level of restriction of the UML metamodel should be such that the flexible modelling described in this paper can be directly supported.

Another strand of future work is to experiment with the transportable Mimio technology, which is more lightweight than the SMART Board. A starting configuration for Knight using Mimio may be a mode, in which Mimio is used with marking pens but without projector. Strokes drawn may then either be interpreted during a session, yielding a possibility to give feedback, or be transformed to UML elements after the session using the techniques described in this paper. In-between this scenario and the one in which Mimio acts as an ordinary electronic whiteboard, a number of possibilities can be imagined. For example, UML elements may be drawn with a non-marking pen while freehand elements may be drawn with marking pens. One of our current goals is to offer a smooth transition between such scenarios.

Finally, the tool is being commercialised by Ideogramic (<http://www.ideogramic.com>). Part of this is, naturally, to support all diagram types in the UML.

## 5. Conclusion

The UML enables the construction of object-oriented models in a usable, visual way. However, in an object-oriented software development project – and in the process of creating a UML model – the UML is *not* sufficiently flexible, and hence it does not support all phases of object-oriented modelling. Likewise, existing tools do not support initial modelling phases in a satisfactory way.

Bunse and Atkinson [3] argue that, when coding starts, a restricted version of the UML metamodel is needed. We, on the other hand, argue that UML should support the initial phases of modelling, where incomplete diagram elements and freehand drawing are heavily used. The answer to both requests is to introduce different levels of restriction in the UML metamodel and to split it into meta-model and meta-

presentation. In this way, it is possible to combine expressive or restrictive meta-presentations with expressive or restrictive meta-models.

A change in the UML metamodel should be accompanied by appropriate tool support. Modelling involves going from more or less expressive diagrams to more restricted models, and tools should support this process. This implies that tools should be aware of the different levels of restriction and be able to assist the user in going from one level to another, either automatically or semi-automatically.

In this paper, we have presented the Knight tool, which was designed to support modelling practice. The Knight tool supports collaboration in modelling, fluid modelling, and shifts between levels of restriction in modelling. Using Knight, we have demonstrated that it is in fact possible to make tool support for actual modelling practice.

**Acknowledgements.** 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. Thanks to Ole Lehrmann Madsen for comments that improved this paper.

## 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., Lyytinen, 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. Beaudouin-Lafon, M. (2000). Instrumental Interaction: An Interaction Model for Designing Post-WIMP User Interfaces. In *Proceedings of Computer Human Interaction (CHI'2000)*. The Hague, The Netherlands.
3. Bunse, C., Atkinson, C. (1999). The Normal Object Form: Bridging the Gap from Models to Code. In *Proceedings of <<UML>>'99 – The Unified Modeling Language*, pp. 675-690, Fort Collins, CO, USA, October.
4. 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.
5. 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)*. The Hague, The Netherlands.
6. Damm, C.H., Hansen, K.M., Thomsen, M., & Tyrsted, M. (2000). Creative Object-Oriented Modelling: Support for Intuition, Flexibility, and Collaboration in CASE Tools. To appear in *Proceedings of ECOOP'2000*. Cannes, France.
7. Damm, C.H., Hansen, K.M., Thomsen, M., & Tyrsted, M. (2000). Tool Integration: Experiences and Issues in Using XMI and Component Technology. To appear in *Proceedings of TOOLS Europe'2000*. Brittany, France.
8. Eged, A. & Medvidovic, N. (1999). Extending Architectural Representation in UML with View Integration. In France, R. & Rumpe, B. (Eds.) <<UML>>'99 – *The Unified*

*Modelling Language. Beyond the Standard*. Second International Conference. LNCS 1723, Springer Verlag.

9. Evans, A. & Kent, S. (1999). Core Meta-Modelling Semantics of UML: The pUML Approach. In France, R. & Rumpe, B. (Eds.) <<UML>>'99 – *The Unified Modelling Language. Beyond the Standard*. Second International Conference. LNCS 1723, Springer Verlag.
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. Haake, J.M., Neuwirth, C.M., Streitz, N.A. (1994). Coexistence and Transformation of Informal and Formal Structures: Requirements for More Flexible Hypermedia Systems. In *Proceedings of the 1994 ACM European conference on Hypermedia technology*, pp. 1-12. Edinburgh, Scotland.
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. Landay, J.A. & Myers, B.A. (1995) Interactive Sketching for the Early Stages of User Interface Design. In *Proceedings of Computer Human Interaction (CHI'95)*.
17. Lending, D. & Chervany, N.L. (1998). The Use of CASE Tools. In Agarwal, R. (Eds.), *Proceedings of the 1998 ACM SIGCPR Conference*, ACM.
18. Madsen, O.L., Møller-Pedersen, B., & Nygaard, K. (1993). *Object-Oriented Programming in the BETA Programming Language*, ACM Press, Addison Wesley.
19. McLennan, M.J. (1993). [incr Tcl]: Object-Oriented Programming. In *Proceedings of the Tcl/Tk Workshop*, University of California at Berkeley, June 10-11.
20. Object Management Group (1998). *XML Metadata Interchange (XMI)*, document ad/98-07-01, July.
21. Object Management Group (2000). *OMG Unified Modeling Language Specification*, document formal/00-03-01, March.
22. Ousterhout, J. (1994). *Tcl and the Tk Toolkit*. Addison-Wesley.
23. Rogerson, D. (1997). *Inside COM. Microsoft's Component Object Model*. Microsoft Press.
24. Rubine, D. (1991). Specifying Gestures by Example. In *Proceedings of SIGGRAPH'91*, 329-337.