Assessing Graphical User Interfaces in Modelling Tools for MDA using the Cognitive Dimensions Framework

John Mathenge Kanyaru, Sheridan Jeary, Melanie Coles, Keith Phalp, Jonathan Vincent

Software Systems Research Centre,
Bournemouth University, Poole, Dorset. BH12 5BB.
jkanyaru;sjeary;mcoles,kphalp,jvincent@bournemouth.ac.uk

Abstract

Model driven development provides a method for moving through modelling phases, allowing for a separation of concerns, among Computationally Independent, Platform Independent and Platform Specific Models. In addition to the production of a variety of models, maintenance of the linkages among successive (e.g., derived) models is an important goal, and, hence, appropriate tool support is vital.

Although there are many tools providing support for software development based on a Model Driven Architecture (MDA) approach, little work has attempted to assess the extent to which these tools provide accessible modelling environments for the range of users who may often participate in model driven development processes. For example, whereas engineers may be familiar with the nuances of particular tools, non-technical users (e.g., business people), who often provide valuable requirements input may have difficulties in making revisions (or amendments).

This paper applies the Cognitive Dimensions framework to assess a number of modelling tools with respect to their support for some typical modelling tasks (e.g., for the creation of use cases, activity diagrams and class models).

Our assessment of the various tools indicated that no single tool satisfied all of the cognitive dimensions used with respect to the selected models. Further, that by recognising explicitly that there are trade-offs in cognitive dimensions in the production of such tools, we may be able to better understand the issues encountered by different groups of users.

1.0 Introduction

The Unified Modelling Language (UML) comprises notations for describing various aspects of a software system. For example, the use case notation is used primarily for expressing software requirements and specifications [1] and in a Model Driven development process is often used to augment Computationally Independent Models (CIM) [2,3,4].

Our work in the VIDE project [5] centres on the design and specification of an Integrated Development Environment (IDE) for MDA, though, clearly these ideas also apply more generally to model driven development. This IDE has to be accessible to business stakeholders, business analysts, systems developers and programmers. For a full discussion on this see [6]. We found that there has been little assessment of existing tools for MDA, save for Tariq and Akhter [7]. However, they do not assess the Graphical User Interfaces of the tools. In addition, there has been little work on assessing the suitability of MDA tools for building different types of model. Thus users have little information at the model level as to the usefulness of the interfaces. In addition, most MDA tools (e.g., [8,9,10]) are geared to programmers and software developers rather than non-technical stakeholders. Thus, the MDA process may not fully benefit from the participation of all stakeholders at the business level, due to the inaccessibility of the existing modelling environments. Hence, we attempt to assess the extent to which MDA tools provide accessible features for those common modelling tasks, such as the construction of use cases, class models and their associated activity models, which involve a breadth of stakeholders, and which, in the experience of the project partners, are those most vital in moving through the MDA process (notably from CIM to PIM).

For this study, we selected three tools, firstly due to their ubiquity in the modelling arena, and secondly due to their apparent use in a number of MDA developments. These are all tools on the Object Management Group list of MDA '*Committed Companies and their Products'* [11] and, in addition, they all support the construction of UML class models, an important notation for MDA-based development ([8,12,13]).

2.0 Cognitive Dimensions Framework

The evaluation of programming and modelling environments presents a number of challenges for which traditional usability techniques are not suitable for evaluating the environment and the notational design issues [14]. The Cognitive Dimensions Framework [15] provides a systematic way of assessing notations in terms of the cognitive impact on users. In addition, it allows for broad brush analysis thus avoiding 'death by detail' and is, thus, excellent for use early in design [14]. Furthermore, it allows for the articulation of many ideas and notions that designers have in mind but have not necessarily formulated fully [16]. The number of Cognitive Dimensions has been increasing although the standard set is thirteen [17]. The interested reader is referred to [16] for a full tutorial. However, this

research considers only those six core Cognitive Dimensions which were used successfully by Green and Petre in the evaluation of three different Programming environments [18]. The six dimensions, and how we will use, them are:

- Viscosity – How much effort is required to perform a single change? We assess the ease with which users of the tools can make changes to models. To undertake this assessment, we will build sample models and try to add or change the models within each of the modelling tool's editors.
- Visibility – How easy is it to view components? We assess whether the tools provide users with model editors where components are easily viewed for use in model construction. This assessment will be based on how we view the layout of modelling components being visible and accessible for each modelling task.
- Juxtaposition – Is it possible to place any two components side by side? We assess whether the tools provide modellers with a means to view models side by side. This may be deemed necessary when a given modelling activity is informed by another, e.g., construction of a class model based on a use case model. An important side to this dimension for this paper is whether or not one can derive parts of a given model from another directly, including traceability of information between models.
- Hidden dependencies – Is every dependency overtly indicated in both directions? Is the indication perceptual or symbolic? We assess whether there are any hidden dependencies between components.
- Premature commitment – Do modellers have to make decisions before they have the information they need? We assess the extent to which the tools require premature commitment during modelling.
- Secondary notation – Can modellers use layout, colour and other cues to convey meaning beyond the official notation of the language? We assess whether tools provide secondary notation to provide extra information about models. Although Petre considers secondary notation with reference to visual programming she believes that is generalised to user interfaces and environment. She believes that good graphics rely on secondary notation and that secondary notation is what makes the difference between experts and novices. Graphical features do not guarantee that a graphical representation will be clear. It is the use of secondary notation which gives the clarity and the poor use of secondary notation is what distinguishes novices from experts.[19]

The purpose of the framework is to encourage discussion on design decisions and as such there are always likely to be tradeoffs made. Blackwell [18] gives the example of changing the structure of a notation to reduce the viscosity which is likely to affect other dimensions such as introducing hidden dependencies or increasing abstraction. These tradeoffs can be shown in Figure 1.

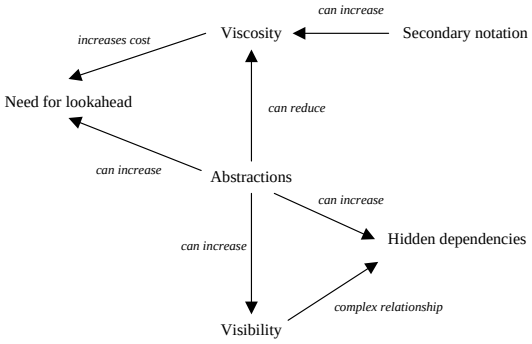

Figure 1: Cognitive Dimensions tradeoffs [20]

3.0 Application of cognitive dimensions

We consider software systems modelling tools that have an established support and user base. These tools are: Borland's Together [21] IBM's Rational Rose [22] and the Eclipse [23] toolset. These three tools also support many UML modelling activities central to MDA development and all three tools artefacts can be imported into a number of MDA tools. (For example Together: Rational Rose and XDE [22,24] and Model-in-Action [25]; Rational Rose: Real Time Studio [26] smartGenerator [27] and Model-in Action [25]; Eclipse: Bridgepoint [28], iQGen [29] and Adaptive [30]). In assessing tools according to our chosen cognitive dimensions, this paper provides examples and subsequent discussion of the way in which the tools represent use case diagrams and descriptions, class and activity models. We review the similarities and differences among the models and their environments across the tools with a view to highlighting the extent to which the tools provide users with facilities to amend or add to any of the models.

3.1 The modelling case

Our modelling case considers a hotel business where guests might make room reservations online, or may telephone the hotel to make such a reservation. The proprietors of the hotel may use a credit card system to obtain payments from guests during check-in to the rooms. Typically, prior to any of these activities, a guest want to determine whether or not there is a room available in the first place.

3.2 Use case models

In this section, we compare use case models as represented in each tool. We review the viscosity of use case models, the visibility of use case modelling components, and whether or not it is possible to juxtapose use case models with any other models (e.g., class models, or sequence diagrams) for a given development project.

3.2.1 Use case model – Together

The Together tool requires a user to be familiar with the notion of a project, and indeed, a model such as the use case in Figure 2 does belong to a UML type of project. Other project types include a Java project, and a Plug-in project among

others. Hence, the project to which the use case model belongs may consist of others, which are listed in a tree structure to the far left of the development pane. To the right of this, and to the left of the model editor window is a toolbar with components (e.g., subject, actor, use case, association line) for use case modelling. We did not experience any issues with regard to the visibility of use case modelling components in Together.

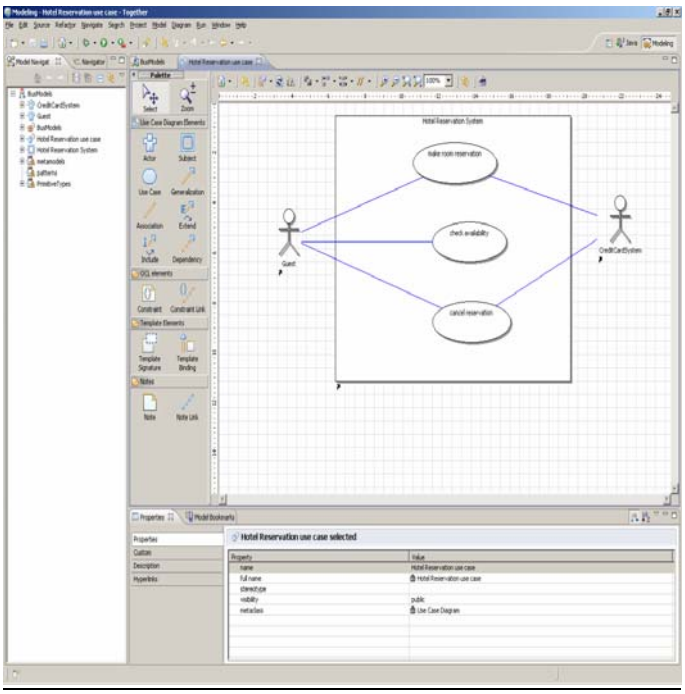

Figure 2: Use case model in Together 2007

It was apparent, however, that the model was viscous in that amending or adding to it required work on other parts of the model that did not require direct amendment. For example, if a user builds the use cases and associated actors, and then remembers to add a system boundary, it becomes necessary to move the use cases inside the system, and align the actors accordingly. The associations also require alignment. This is an illustration of knock-on viscosity of Together's use case editor.

The tool provides useful drag and drop functions, but a link or linked element would have to be moved to a desired position individually. There is no automatic relocation of links or use cases (or actors) when extra elements are added or some removed, or moved.

Whereas the use case elements in the model are visible, the use case components for selection in the palette are not particularly visible to the user. However, this has been addressed by the tool by providing different Layout mechanisms (e.g., List,

Columns, Details, etc). The Layout chosen in Figure 2 is Columns as it allowed greater visibility.

3.2.2 Use case model - Eclipse

In contrast to Together, Eclipse lays out use case modelling components at the top of the model editor space. An issue with building use case models in Eclipse is that it is not possible for a user to lay out associations in a style that they want. That is, moving an actor or a use case, or indeed adding an extra use case causes the Eclipse modeller to move any other use cases. Whereas the automatic moving of model elements ensures space for the newly added or moved use case, this may not be laid out the way a user would like. This makes modification of models difficult as some time may be spent trying to place model elements in positions that make the model more legible. It is important to allow such flexibility.

Hence, a modeller experiences both knock-on viscosity and repetition viscosity. Knock-on viscosity is experienced because additions to the model cause other model elements to be moved without user intervention, and repetition viscosity is experienced because a user who is not happy with the automatic layout of the model will have to try and alight model elements manually – a tedious task. There are also implications for the secondary notation dimension. A similarity between Together and Eclipse in use case modelling is their use of the notion of a project as a container of the use case model. Additionally, both tools provide a means to show the system boundary.

3.2.3 Use case model – Rational Rose

In Rational Rose the concept of a system boundary for the use cases is not explicitly represented. The use case modelling components are laid out in a toolbar to the left of the model editor window very much like Together. A tree structure of the various model objects is shown to the left of the toolbar. A modeller new to UML modelling with Rational Rose can use the tool-tip facility to find out what each component is.

Viscosity of use case models will be experienced when modifying a use case or actor that is on top or in the midst of several other use cases or actors. Such an amendment does have the knock-on effect of necessitating the movement of other use cases and actors below or in the neighbourhood. Rational Rose allows the modeller to write a specification for each use case element. Since descriptions are not part of the standard notation, one might argue that Rational Rose allows modellers the use of a secondary notation to provide more detail about the use case using unstructured text.

3.3 Class models

In this section we consider class models built using each of the tools. We try to find out whether or not a user is able to directly use some elements from a use case model in constructing a class model.

3.3.1 Class model - Together

Since Together requires models to belong to a project, we initiated the class design editor within the same project as the Together use case model seen in Figure 2. Again, this was done to determine whether a user would be able to directly obtain elements of a use case model to build a class model. This was not possible, and the class diagram was built without such direct derivation.

Class modelling components in Together are laid out in a vertical toolbar, which is common with the use case modelling components in the tool. There is a choice between Column and List layout. The components have clear visibility in the Columns layout but in List layout, however, (even taking into account a Detail component) the visibility of the class components diminishes markedly. Moving the class elements in the editor is relatively easy, but reorganisation becomes difficult when the amended class is at the top or middle of a hierarchy, which requires moving lower level classes and associations around to produce a legible model.

It is likely that modellers would want to derive some classes from a use case model. However, there is no way for a user to lay out a use case model and a class model side by side for such cross-referencing. In other words, the cognitive dimension of juxtaposition of models is not facilitated in Together. Again, there is no way to derive the classes (or other class model elements) from the use case model. Additionally, there is no way to display the use case model adjacent to the class editor for reference while building the class model.

3.3.2 Class model - Eclipse

Eclipse behaves quite differently from Together when building a class model. For example, as soon as a modeller makes an association between any two classes (e.g. Guest and Room), Eclipse assigns roles to both ends of the associations, and also assigns the cardinality of the roles.

It was also noticed that Eclipse automatically provides the getter and setter methods based on the associations. This is an interesting feature of Eclipse, and may be a useful feature when creating PIM (or PSM) classes for code generation but has no value for business stakeholders. Eclipse also lays out the classes in the model, based on the class size and decides where on the screen a class is positioned. For example, moving the Room class causes Eclipse to redraw the association between Guest and Credit Card System differently. This is not intuitive since a modeller might wish to place classes and association lines in screen areas where they feel most appropriate (see Figure 3).

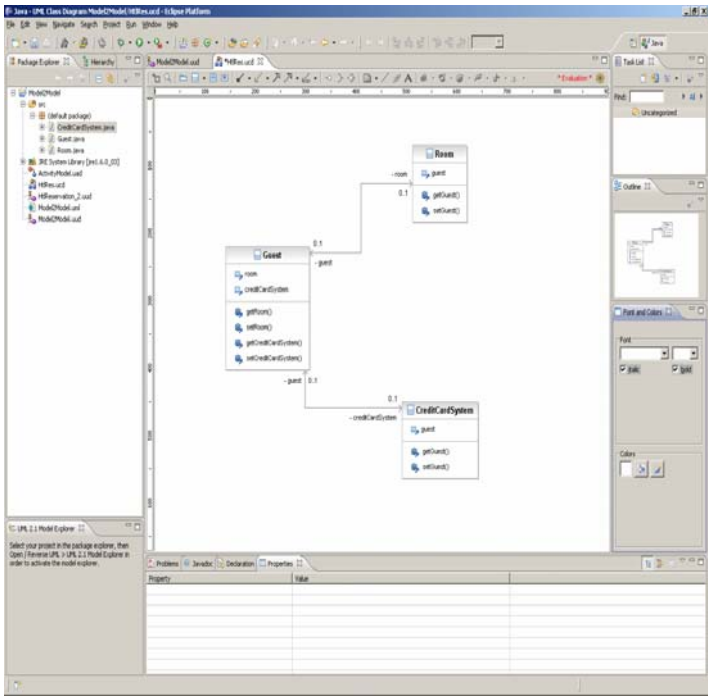

Figure 3: Class model in Eclipse

In Eclipse, drawing components for the class model or activity diagram, are laid out at the top of the drawing palette. A tree structure is produced to the left of the model to show the existing models for the current project. The classes are built from scratch as there is no facility to move any elements of a use case, or activity diagram into the class model. Eclipse allows a modeller to edit the roles, cardinality or the methods to depict a general class diagram. The Eclipse environment does not provide for juxtaposition of different models (e.g., use case and class models), nor is there a way to obtain class elements directly from a use case model in Eclipse.

3.3.3 Class model – Rational Rose

In Rational Rose, building class models is straightforward, but there is no facility to 'drag' a use case element and use it as a class or class property or method. However, if a modeller builds a class that is given the same name as an existing actor, Rational Rose gives the class an icon that has the look of an actor, with a stereotype indicating that the class is from the use case view.

It is possible to amend the class to have the standard look of a UML class notation, but one cannot remove the stereotype indicating the class is associated with an actor in the use case view. That is, Rational Rose does not allow the modeller to remove the stereotype implying that the class is associated to an actor in a use case model. This forces premature commitment on the modeller in deciding that such a relationship between objects in the different models exist. Moreover, there seems

to be a hidden dependency between the respective model elements, i.e., actor and class, but one is not sure which parts of the elements matter in the dependency (e.g., is it just the name, or the whole object?). That is, there seems to be little value in the forced stereotype since there is nothing more (e.g., methods or attributes) a modeller would get from the actor in building the class.

3.4. Activity Models

This section considers activity models built in each of the tools. The section attempts to determine whether a modeller is able to derive activities from a use case model. The creation of an activity diagram is particularly necessary in MDA in order to allow the developer to create a design model at the PIM level based on a rigorous understanding of the system behaviour.

3.4.1 Activity model - Together

The set of components for drawing an activity model in Together occupies more real estate than those for drawing use cases. Hence, the layout for the components (at the left of model editor window) can be changed from columns to list, to allow for most of them to be shown.

The icons in component view occupy considerable screen estate, but, the tool provides a means to scroll down should one need to. Another issue with the activity model itself (rather than drawing components) is the viscosity of the model. That is, if a modeller were to add a partition or lane, there is a knock-on to the existing model, and activities and control flow get shifted such that the model loses clarity. There is no facility within Together for directly exporting use case elements into an activity model.

3.4.2 Activity model - Eclipse

Unlike Together, the drawing components for activity models in Eclipse are spread across the top of the editor window. The components have no textual description attached to them, but there is a tool-tip describing what each component is. Eclipse does not provide any other layout for the drawing components. In addition, Eclipse does not allow the use of the vertical bar to merge activities (use of the vertical bar to split activities is allowed).

3.4.3 Activity model – Rational Rose

In Rational Rose, as with Eclipse and Together, it is not possible to obtain activities from either use case or class models.

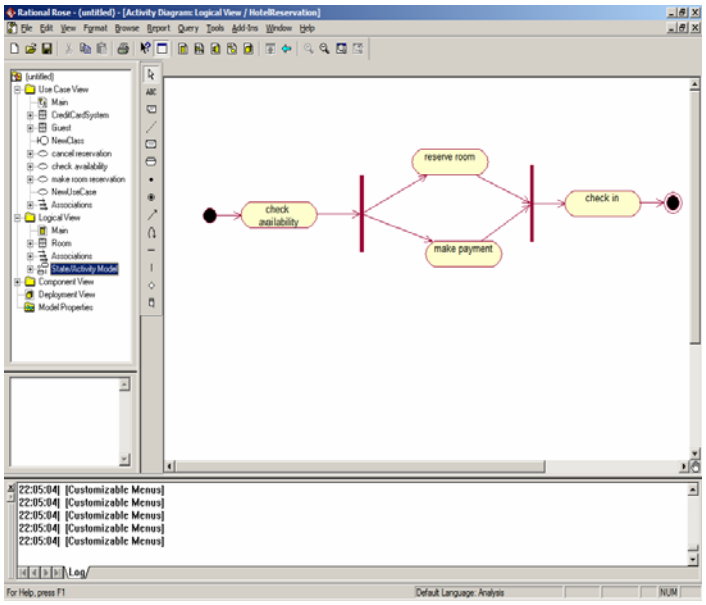

Figure 4: Activity model in Rational Rose

Rational Rose provides an intuitive activity modelling area where start and end states are easy to add to the model, and the vertical bar for splitting or merging activities is easy to use (see Figure 4).

4.0 Discussion

There are some modelling concepts that cross-cut the tools assessed so far. For example, all the tools tend to deploy the concept of a project as a container for software models. This is an important aspect of modelling or software development in general as several models may often be built to depict different views of a system.

Some tools (e.g., Together, Rational Rose and Objecteering) provide modelling components on a vertical toolbar to the left of the model editing window, while other tools (e.g. Eclipse) lay out such components just above the model editor window. The placement of the toolbar and the information that is provides is important, particularly with novices and new users of the tool. Eclipse did not provide any flexibility in laying out models in the model editor. It is important, particularly with respect to secondary notation to allow this to aid both the clarity and understanding of graphical representations for both novices and experts alike.

Table 1 provides an overview of the way in which the tools have been evaluated in terms of the cognitive dimensions framework which has been used to assess modelling activities.

Tools/ Dimension	Together	Eclipse	Rational Rose
Viscosity	Knock on viscosity experienced when additions are made to a model	Both knock on and repletion viscosity experienced when additions are made to a model	Knock-on viscosity experienced when additions are made to a model
Juxtaposition	Does not facilitate juxtaposition of models. One has to minimise one model to view another.	Does not facilitate juxtaposition of models. One has to minimise one model to view another.	Does not facilitate juxtaposition of models. One has to minimise one model to view another.
Visibility	Modelling components clearly visible and different layouts provided for laying out components on a vertical toolbar.	Small icons used for modelling components; seems reasonably visible to a user	Quite reasonable visibility of components
Hidden dependencies	None of this experienced in the modelling tasks performed	None of this experienced in the modelling tasks performed	Experienced when building a class model – classes named the same as an existing actor seem to be stereotyped accordingly.
Secondary notation	None for the modelling tasks performed	None for the modelling tasks performed	Use of text to provide descriptions of model elements, e.g., use case spec.
Premature commitment	None experienced so far	Forces naming of roles and specification of multiplicity	Forces implication of a class to be based on an actor where a similarly named actor exists.

Table 1: Summary of tools assessment with cognitive dimensions

An important point to reiterate is that whilst many modelling tools, including those that were assessed, allow forward and backward tracking between sequence, collaboration and class diagrams none of the assessed tools, provide a means to carry forward information from a use case or activity model to a class model.

5.0 Conclusions and further work

Our application of cognitive dimensions to MDA tools provided valuable insights into the user experience for modellers. For example, whereas some linkages are often maintained (such as sequence and collaboration diagrams) most tools do not provide a means to move from a use case or activity model to a class model. Despite the issues involved in supporting such a transition, it is desirable that such a feature is incorporated in a MDA development tool. Moreover, juxtaposition of models is important where a source model is used to inform the development of a target model. The authors have no knowledge of any UML modelling tool that provides such a feature, though we are of the view that this would be desirable for traceability purposes.

Other issues, such as the visibility of modelling components may not be straightforward to address, since working towards high visibility might in many cases also mean reducing the screen estate available for the building of models. The viscosity of models is an aspect that may also be relatively difficult to address since it is inevitable that changes at a high level in a model may cause many changes at a lower level of detail. These factors thus mean that tool vendors have to make pragmatic decisions of the cognitive measures they consider most important to tool users [16]. We are of the opinion that the visibility of the modelling components and the viscosity of models are the most important factors to be considered in any design, coupled with the use of secondary notation. However, by recognising that such trade-offs exist, and by explicit identification of the impact of design decisions on different cognitive dimensions, we hope that such decisions will be informed by the needs of a variety of user groups.

This work has been funded under the VIDE project by the European Commission within the Sixth Framework Programme (FP6-IST-2004-033606).

6.0 References

1. Mannio, M. and Nikula, U. Requirements Elicitation Using a Combination of Prototypes and Scenarios. in The 4th Workshop on Requirements Engineering, 2001. Buenos Aires, Argentina.
2. Elkoutbi, M., I. Khriss and Keller, K. "Automated Prototyping of User Interfaces Based on UML Scenarios." in *Automated Software Engineering*, 2005, 13(1).
3. Engels, G., Förster, A., Heckel, R. and Thone, S. *Process modeling using UML*. in *Process-Aware Information Systems*. M. Dumas, W. van der Aalst and A. ter Hofstede 2005, New York, Wiley Publishing**:** 85-117.
4. Garrido, J. L., Noguera, M., Gonzalez, M., Hurtado, M. V. and Rodriguez, M. L. "Definition and use of Computation Independent Models in an MDA-based groupware development process." in *Science of Computer Programming* 2007, 66(1).
5. VIDE. "Visualize all model driven programming. FP6-IST-2004-033606." 2006, from http://www.vide-ist.eu/index.html.
6. Phalp, K., Jeary, S., Vincent, J., Kanyaru, J. and Crowle, S. Supporting stakeholders in the MDA process. in 15th International Software Quality Management, 2007. Tampere, Finland.
7. Tariq, N. A. and Akhter, N. 2005. Comparison of Model Driven Architecture (MDA) based tools. Department of Computer and Systems Sciences. Stockholm, Royal Insititute if Technology (KHT). Masters.
8. Budinsky, F., Steinberg, D., Merks, E., Ellersick, R. and Grose, T. 2004. Eclipse Modelling Framework. E. Gamma, L. Nackman and J. Wiegand, Addison-Wesley.
9. Eshuis, R. and Wieringa, R. "Tool Support for Verifying UML Activity Diagrams." in *IEEE Transactions on Software Engineering*, 2004, 30(7).
10. Fabro, D., Bezivin, J. and Valduriez, P. Weaving Models with the Eclipse AMW Plugin. in Eclipse Modelling Symposium 2004. Esslingen, Germany.
11. OMG. "MDA Committed Products." Retrieved 20 December 2007, from http://www.omg.org/mda/committed-products.htm.
12. Mellor, S. J., Kendall, S., Uhl, A. and Weise, D. *MDA Distilled: Principles of Model-Driven Architecture*. Addison-Wesley, 2004.

13. Raistrick, C., Francis, P., Wright, J., Carter, C. and Wilkie, I. *Model Driven Development with Executable UML*. Cambridge, UK, Cambrideg University Press, 2004.
14. Green, T. R. G. and Petre, M. "Usability Analysis of Visual Programming Environments: A 'Cognitive Dimensions' Framework." in *Journal of Visual Languages and Computing*, 1996, 7: 131-174.
15. Green, T. R. G. *Cognitive dimensions of notations.* in *People and Computers V*. A. Sutcliffe and L. Macaulay 1989, Cambridge, UK, Cambridge University Press**:** 443-460.
16. Green, T. and Blackwell, A.*Cognitive Dimensions of Information Artefacts:a tutorial*. 1998, Cambridge University & Leeds University
17. Blackwell, A. F. Dealing with New Cognitive Dimensions. in Workshop on Cognitive Dimensions, 2000. University of Hertfordshire.
18. Blackwell, A. F., Whitley, K. N., Good, J. and Petre, M. "Cognitive Factors in Programming with Diagrams." in *Artificial Intelligence Review*, 2001, 15(1/2): 95-114.
19. Petre, M. "Why Looking Isn't Always Seeing: Readership Skills and Graphical Programming." in *Communications of the ACM*, 1996, 38(6): 33-44.
20. Blackwell, A. F. and Green, T. R. G. *Notational Systems - The Cognitive Dimensions Framework*. in *HCI Models, Theories and Frameworks: Towards a multidisciplinary science*. J. Carroll 2003, San Fransisco, USA, Morgan Kaufmann.
21. Borland. "Together Architect " Retrieved 8 July 2006, from http://www.borland.com/us/products/together/index.html.
22. IBM. "Rational XDE Developer " Retrieved 5 August 2006, from http://www-306.ibm.com/software/uk/rational/awdtools/swdeveloper.html.
23. Eclipse. "Eclipse project." Retrieved 7 August 2006, from http://www.eclipse.org.
24. IBM. "Rational Software Architect." Retrieved 20 December 2007, from http://www-306.ibm.com/siftware/awdtools/architect/swarchitect.
25. Mia-Software. "Model-In-Action." Retrieved 19 December 2007 2007, from http://www.mia-software.com.
26. Artisan. "Real Time Studio." Retrieved 20 December 2007, from http://www.artisansw.com.
27. BITPlan. "smartGenerator." Retrieved 20 December 2007, from http://bitplan.com.
28. MentorGraphics. "BridgePoint/xtUML or EDGE UML Suite." Retrieved 19 December 2007, from http://wwww.mentor.com.
29. innoQ. "iQgen." Retrieved 19 December 2007, from http://www.inoq.com/iqgen.
30. Adaptive. "Adaptive Software " Retrieved 20 December 2007, from http://www.adaptive.com.