SPECIFIC TARGETED RESEARCH PROJECT
INFORMATION SOCIETY TECHNOLOGIES

FP6-IST-2005-033606

VIsualize all moDel drivEn programming VIDE

WP 5	**Deliverable number 5.1 The visual user interface**

Project name:	Visualize all model driven programming
Start date of the project:	01 July 2006
Duration of the project:	30 months
Project coordinator:	Polish - Japanese Institute of Information Technology
Leading partner:	Bournemouth University
Due date of deliverable:	30 November 2007
Actual submission date	14 January 2008
Status	final
Document type:	Report
Document acronym:	DEL
Editor(s)	Sheridan Jeary, John Mathenge Kanyaru, Melanie Coles, Andreas Roth, Tomek Wadziak, Piotr Habela, Keith Phalp, Jonathan Vincent
Reviewer(s)	Sheridan Jeary, Andreas Roth, Piotr Habela, Keith Phalp, Jonathan Vincent
Accepting	Kazimierz Subieta
Location	http://www.vide-ist.eu
Version	14
Dissemination level	PU (Public)

Project supported by the European Commission within Sixth Framework Programme

Abstract:

This deliverable contains a summary of the work that has been carried out within Work Package 5. This has involved an in depth piece of research into MDA tools to ascertain their required features and to determine those features that could be useful within the VIDE project. Building on this work further research was carried out into the areas of Software Visualisation, Visual Programming and finally Diagramming, concentrating on aspects of class diagram layout to further generate requirements. Four modelling tools were then analysed using the Cognitive Dimensions framework to facilitate discussion, and some further requirements were thus obtained. An exploratory prototype of the CIM level user interface was then developed and feedback collected. This resulted in the specification of the definitive prototype for the CIM level interface. The Visual Code Editor is then specified before the Visual Expression Builder that allows for the construction of OCL expressions. Finally the work details the VIDE approach to dealing with legacy applications.

The VIDE consortium:

Polish-Japanese Institute of Information Technology (PJIIT)	**Coordinator**	**Poland**
Rodan Systems S.A.	Partner	Poland
Institute for Information Systems at the German Research Center for Artificial Intelligence	Partner	Germany
Fraunhofer	Partner	Germany
Bournemouth University	Partner	United Kingdom
SOFTEAM	Partner	France
TNM Software GmbH	Partner	Germany
SAP AG	Partner	Germany
ALTEC	Partner	Greece

History of changes

Date	*Version*	*Author*	*Change description*
1.11.2007	1	Sheridan Jeary, (PJIIT Grzgorz Falda, Tomas Wadziak)	document creation (addition of chapter from PJIIT)
5.11.2007	2	Sheridan Jeary	Document update
28.11.2007	3	Sheridan Jeary, John Mathenge Kanyaru, (SAP Andreas Roth)	Document update and addition of chapter from SAP
10.12.2007	4	Sheridan Jeary, John Mathenge Kanyaru, (SAP Andreas Roth)	Document update and initial review by Piotr Habela PJIIT
19.12.2007	5	Sheridan Jeary, John Mathenge Kanyaru, Melanie Coles (SAP Andreas Roth)	Document update throughout and update of SAP chapter
4.1.2008	12	Sheridan Jeary, John Mathenge Kanyaru, Melanie Coles (SAP Andreas Roth PJIIT Piotr Habela/Tomek Wadziak)	Document update throughout and update of PJIIT chapter
7.1.2008	14	Sheridan Jeary, John Mathenge Kanyaru, Melanie Coles (SAP Andreas Roth PJIIT Piotr Habela/Tomek Wadziak)	Document update throughout and update of PJIIT chapters

Table of contents

List of figures

List of tables

1. Introduction and Overview

The Model Driven Software Development (MDSD) process has changed the way that software development is approached. No longer is it expected that the user will be in possession of only programming skills. Users will come from a number of different backgrounds and will expect to be presented with both information and an interface that is commensurate with their ability, understanding, experience and expectation. This will involve business users expecting to be able to store and manage documents and parts of documents, that relate to their business and to link relevant sections together. Analysts and modellers will be expecting to add, edit, delete and refine the information and models they have, probably in some iterative process to create models of the business processes, and from there produce a requirements specification that will be of value to all parties. The specification models will then be refined to produce first cut activity diagrams and class models, prior to their modification and transformation as the next part of the process. Finally, the programmers will want to view the generated code and be able to trace the origins of the artefacts that produced it. This traceability is important at all steps of the process. If you are creating a model, then the source of the content is important as are the versions of it. This process needs to be supported at the interface to add value to the users' tasks. There has been considerable research into the design of the user interface and the use of computers by different classes of people, but there is no work relating to the interface and the related specific requirements for Model Driven Software Development.

1.1 User groups

In Deliverable 1 [1] the consortium examined the user groups that would be likely to use the VIDE environment and a number of different groups were defined. The following different users were found to have different views of the model and its related code that are relevant to Work Package 5.

- Domain users usually have either no or very little knowledge about business modelling but they will be able to produce requirements for a software application either alone or with assistance from the business or requirements analyst. It will be possible to produce CIM level models if the user is a Business Consultant. The language, the interface and the graphical representation should be easy to understand so that domain users can validate the correctness of the models.
- Business/Requirements Analysts are one of the main user groups of the VIDE environment. They hold interviews with domain users and analyse and model the proposed solution on the CIM level. They have knowledge of modelling of business processes as well as technical architectures and need to be comfortable with the process of refining the model. They need to have models that are understandable for the stage of the process they are at. There should be context friendly assistance with both the interface and the model they are using. All actions should be traceable and different versions should be stored.
- Analyst/Designers are likely to have a strong background in conceptual modelling and UML models that are applicable to different levels/stages of the development process. For reusing or composing new applications with or without the existence of pre-existing components the analyst/designer is likely to use the UML to understand the business logic that is implemented by a component or to define how multiple components may be composed. They are responsible for the conceptual Platform Independent Model (PIM) that is based on the Computational Independent Model (CIM) produced by the business analyst.
- Analyst/VIDE Programmers will have a strong background in behavioural modelling and will need tools to assist in that task. The Analyst is likely to use the graphical notation but may use textual notation if it is necessary, but the VIDE Programmer will use textual or graphical notation depending on which is most efficient for the task in hand. They will also implement components designed by software designers.

1.2 High level requirements

The high level requirements for the VIDE graphical user interface have come from a number of sources. Firstly the description of Work provided a useful starting point. Secondly, the research that underpinned Deliverable D1 was important.

1.2.1 Requirements from the Description of Work

1. A fully visual toolset to be used by both IT specialists and individuals with little or no IT experience such as specific domain experts, users and testers.

2. Make programming more user friendly
3. Create visual tools for prototyping, programming, debugging and testing. Make all stages of application development more accessible to non-IT professionals
4. VIDE will be an open and interoperable platform

1.2.2 Requirements from Work Package 1 work

Requirement	**Name**	**Priority**
REQ – NonFunc1	Accessibility at the CIM level - the VIDE environment should provide non-technical, business domain descriptions. Non-technical users working at the CIM level should be able to input, retrieve and understand their business domain descriptions in a notation that is non-technical and accessible.	SHOULD
REQ – NonFunc 2	CIM level collaboration - the VIDE environment MAY offer collaboration mechanisms. It may be possible for CIM or PIM users to collaboratively work on a shared CIM view through a communication mechanism (such as shared notes or links to shared views between stakeholders).	MAY
REQ – NonFunc 3	On-line support for CIM/PIM users - Users working at the CIM/PIM level should have immediate access to online/in-system, context sensitive help that describes how transformations between CIM, PIM and PSM levels are specified and used in the modelling activities supported by VIDE. Help should be expressed in non-technical terms wherever possible	SHOULD
REQ – NonFunc 4	Clear and unambiguous notation - the VIDE environment should use notation that has clear, comprehensible and unambiguous semantics suited for the user working at the CIM, PIM or PSM level.	SHOULD
REQ – NonFunc 5	Model view saliency - VIDE models views must be user-oriented. Views on CIM, PIM and PSM must be controllable depending on specific user interactions with the VIDE environment. It should be possible for users to dynamically control the scope and technical content of these views depending on their specification/comprehension needs.	SHOULD
REQ – NonFunc 6	Appropriate textual/graphical fidelity - VIDE must provide appropriate textual and graphical modalities for its users. They should be able to work with textual or graphical notations that offer the most effective expressiveness for CIM, PIM and PSM concerns.	SHOULD
REQ – NonFunc 7	Timely feedback and constraints - the VIDE environment should provide feedback on user actions at all modelling levels. Multiple users working on the same VIDE project should receive rapid feedback on their attempted actions within the VIDE environment. Such feedback should indicate their success or failure to complete an action or task; its impact on their local modelling level; its potential impact on other modelling levels; and any constraints that may impact on the success of their intended action.	SHOULD
REQ – NonFunc 8	Runnable and testable VIDE prototypes - the VIDE environment should allow execution of runnable models. VIDE users should be able to validate at any time (where possible) the models that can be automatically transformed into an executable form.	SHOULD
REQ – User 1	Flexibility and interoperability of VIDE language and tools - the VIDE language and tools should have flexibility and be interoperable with existing tools	SHOULD
REQ – User 2	Reuse of UML Standard - the VIDE tools for certain user groups SHOULD be informed by existing tools for the user groups. End users are very sensitive to using standards.	SHOULD
REQ – Tool 7	Meta-modelling Framework - VIDE SHOULD use GMF as it's graphical modelling framework	SHOULD
REQ – Tool 9	CIM modelling standards - VIDE may support CIM level modelling with BPMN; where there is inadequate or no support for BMPN, VIDE	MAY

	may provide CIM modelling capability with UML activity diagrams	
REQ – Tool 12	VIDE extensibility - the VIDE tools should be extensible via a plug-in mechanism.	SHOULD
REQ – Tool 14	Model driven approach - the VIDE tool must strictly follow a model driven approach.	MUST

Table 1: Requirements from Work Package 1

1.3 WP5 Summary

The objective of this Work Package is to provide a full set of requirements for the VIDE Graphical User Interface at the CIM and PIM levels. This deliverable shows how the requirements were obtained starting with research into the existing MDA tools that are covered in Chapter 2. The methods of evaluation are covered in Chapter 3. These will allow discussion of the requirements and evaluation of a prototype. Chapter 4 explores the rich research areas of Software Visualisation, Visual programming and Diagramming Research and shows the requirements that resulted from that work. Four major modelling toolsets were explored in Chapter 5 to see what functionality they had and to see which of their design features would be useful. In addition the work allowed discussion of general modelling tool functionality. The initial exploratory CIM level prototype is specified and evaluated in Chapter 6 before a fully detailed requirements summary is explained in Chapter 7 and the definitive CIM level prototype is specified in Chapter 8. Chapter 9 details the specification of the Visual Code Editor for State Visualisation and Chapter 10 the associated Visual Expression Builder which allows definition of Object Constraint language (OCL) expressions. Chapter 11 details the interface to legacy applications using Web services. The final chapter summaries this report and gives details of proposed future work.

2. Research into Model Driven Architecture tools

2.1 Introduction

What should a Model Driven Architecture (MDA) tool contain to meet the Object Management Group's (OMG) requirements for MDA tools? The OMG's MDA has been developed since 2000, so the tools to support this complex question are still in their infancy. Some tools pre-dated MDA and have been re-classified by the vendors/developers as MDA compliant tools (Objecteering [2] PathMATE [3]), whilst others have been developed since the inception of MDA (BoldExpress Studio [4]). Some tools have been developed as part of a research program (Generative Model Transformer project [5]) others are commercial tools (Rational Software Architect [6], Select Component Factory [7]). Some tools attempt to support the entire lifecycle (ArcStyler [8], Model-In-Action [9], XDE [10], OptimalJ [11]) and others focus upon a small section of the stages (CASSANDRA [12], Codagen [13], Tau [14]). Whilst some use all the standards and open source tools that are available to them (Together 2006 [15]); others develop their tool based on a proprietary approach (modelscope [16], Codeless [17]). Clearly, there is no tacit agreement about what features an MDA tool should contain. This issue has been recognised by OMG and an MDA Tool Capabilities Request For Information (RFI) [18] has been issued and also an MDA Tool Component Request For Proposal (RFP) [19].

The first stage of this review will explore the literature to develop a list of features that an MDA tool could potentially contain. This list of features will then be applied to the MDA products listed on OMG's website [20]. This will be an investigation of the literature available on each product from both datasheets and from vendors' websites.

2.2 MDA Tool Requirements Review

2.2.1 Introduction

The often cited primary goals of MDA are portability, interoperability and reusability [19, 21-24], therefore the MDA tools, at the very least, should support these three objectives.

Metadata is critical to the concept of interoperability, it is the primary means by which interoperability is achieved, and therefore an MDA tool must be able to store, manage and publish both application and system metadata [25]. Metadata management and integration is supported by the core MDA standards [25].. Platform Independent Models (PIMS) facilitate the creation of different Platform Specific Models (PSMs) corresponding to the same set of PIMs, which result in implementations that are easily (if not automatically) integrated [21] and therefore interoperability is achieved. Consequently the ability to integrate various components of an application on different platforms should result from the basic principles of MDA.

PIMs also play an important part in re-use of legacy applications; the integration can be carried out at the platform independent level, using reverse engineered PIMs that represent the legacy application [26].

2.2.2 Models

The OMG defines a model as "*a formal specification of the function, structure and/or behaviour of a system*" [27] and it is these models that are the primary artefact of an MDA [22], so an MDA tool has to, at the very least, be able to support model generation and manipulation. Also central to MDA is that many models can be created at different levels of abstraction, these models can be linked together and can be transformed and, ultimately, implemented [28].

2.2.2.1 Meta-models

Meta-models are used to support the definition of syntax and semantics of models; they are usually accompanied by natural language descriptions of concepts that correspond to elements of the meta-model, defining informally the semantics of the modelling elements [26]. Meta-models and meta-data are critical to interoperability, users need to be able to define their own meta-models for backwards compatibility and for integration to legacy systems to be achieved.

2.2.2.2 Types of model: CIM, PIM, PSM, Code

The models used in MDA tools as principally specified by the OMG are mainly the Platform Independent Model (PIM) and the Platform Specific Model (PSM). Other models that are discussed, although not as ubiquitously, are the Computational Independent Model (CIM) and the code itself.

The CIM, or Business Model or sometimes even the base PIM [22] is the most infrequently mentioned model although it is said to play an important role in bridging the gap between domain experts and system designers and developers [21]. Thus the CIM aids not only in understanding the problem and but also as a source of shared vocabulary for the other models [21]. However, the automatic transformation from CIM to PIM is considered *not feasible* as human intervention is always required in specifying the system [28]. It may be that CIM development is seen as problematic and complex so is frequently omitted from MDA tools.

There appears to be no clear definition of what a CIM, PIM, and PSM actually are, where the line is between each. There seem to be 'degrees' of *CIMness* and *PIMness* and it depends upon the user's point of view as to what sort of model it is [29] and many notions are still loosely defined [30]. This is obviously problematic, for vendors developing tools and for the variety of users using the tool. How can one tool be compared against another tool, if both sets of vendor documentation talk about PIMs, but in reality mean different levels of model?

2.2.2.3 Model Support

PIMs are seen as having two uses: by distilling the fundamental structure and meaning they make it easier to validate the correctness of the model and they make it easier to produce implementations on different platforms [27]. Consequently, MDA tools should have facilities for validating models and also allow development of a range of implementations.

The main standard for model development in MDA is the Unified Modelling Language. Most tools could therefore be expected to support model development using a full set, or subset, of the defined UML diagrams. They should also be expected to provide support for model import and export so that existing models may be imported into the tool and models may be exported to other tools. Elements of model merging and differencing, if you have multiple views of the same system, could reasonably be expected to support modelling tasks.

2.2.2.4 Traceability and model/code synchronisation

Traceability management is seen as being essential in an MDA-based approach. If not automated, models and implementation will become inconsistent, losing the majority of the benefit of MDA [26]. Information needs to be kept about which elements are related to which by transformation, i.e. tracing information or what has been termed a '*persistent transformation*' [31]. Such information needs to not only be stored, but also acted upon, thus changes in one model should be propagated to alter the referencing models, whether they are *upstream* or *downstream* of the changes. However, such bi-directionality and reverse engineering is seen as one of the ultimate goals of MDA tools, whether they are currently capable of supporting such fluidity remains to be seen.

2.2.2.5 Storage

Alongside model transformation is the fundamental functionality of model storage, the former depending heavily on the latter [32]. The OMG standard for model definition is via Meta-object facility (MOF) and the standard for exchanging models is XML Model Interchange (XMI). How the models are stored is not necessarily important for the tool user that the models can be stored and the transformations can be stored is what is important. However, that models can be exported and imported between tools is important, therefore an MDA tool should employ the current standards to support maximum interoperability of tools.

A set of consistent models means the models must be stored in a repository. Ideally such a repository should be accessible to multiple users, and must be coupled with version management tools. It must also have a centralized access point and administration tools [33]. It means that the repository will become the backbone of an MDA tool, supporting the range of functionality that could naturally be expected from a professional development tool.

2.2.2.6 Levels of Abstraction

The models used in MDA do appear to have contradictory requirements, on the one hand they must be abstract enough to help the designer model the domain and communicate with the user, but also detailed and semantically rich enough to specify business rules and to be used in code generation [29]. The MDA tool must be able to support these conflicting ideas, enabling models of high level abstraction to allow creativity and communication and then also the refinement or transformation of these models into models that are more detailed and semantically rich.

2.2.3 Standards

MDA represents a positive effort to make the OMG standards “churn-proof”[34], however for the MDA tools to support this they themselves need to be “churn-proof”. The use of a specific tool should not tie the users to that vendor, thus the implementation of standards across all tools, and the non-modification of these standards is essential. At the core of MDA are a number of OMG standards: UML, MOF, XMI and the Common Warehouse Meta-Model (CWM) [25]. It is this foundation of developed and developing standards that gives MDA its coherence in model development and management. Thus MDA tools could reasonably be expected to support these standards.

2.2.3.1 Unified Modelling Language (UML)

UML is OMG’s standard modelling tool for MDA, however the developer’s fluency with UML can have an impact on the successful use of an MDA tool [35], and thus a developer has to acquire more skills before MDA tools can be applied. Support for UML modelling and the validation of such models to support developers therefore must be a consideration of MDA tools.

2.2.3.2 Object Constraint Language (OCL)

UML defines a formal assertion language, OCL, which facilitates the specification of constraints [27] which can be used to define pre- and post-conditions on operations [26].

2.2.3.3 UML profiles

A UML profile is a set of extensions to UML, using primarily stereotypes and tagged values that enrich a model’s semantics [27]. Therefore UML profiles are useful for adding detail to models and also facilitating the transformation to a specific platform.

2.2.3.4 Meta Object Facility (MOF)

The MOF provides an abstract language and framework for describing and representing meta-information, it also defines a framework for implementing repositories to store the models [26]. Thus the MOF is integral to the model storage mentioned in Section 2.2.2.5.

The MOF is the unique meta-meta-model for all IT-related purposes, containing all the universal features that are not specific to a particular domain language [30]. It allows users to configure and customise the tool.

It is the MOF alignment with UML that means any tool intended to create UML models can easily be adapted to create MOF meta-models and thus be classified (or re-classified) as an MDA tool [30]. Tools in existence before the MDA standard was defined can use MOF to align the tool with the OMG standard.

2.2.3.5 XML Metadata Interchange (XMI)

XMI is the standard interchange mechanism used between various tools, repositories and middleware [27], and will support the import and export of models discussed in 2.2.2.3. With the implementation and use of a standard for such functionality vendor tie-in can be avoided.

2.2.3.6 Common Warehouse Metamodel (CWM)

CWM is the data warehouse standard, covering the full lifecycle of designing, building and managing data warehouse applications [27].

2.2.3.7 Other MDA standards

There are other models that are part of the OMG's MDA standard such as the Object Constraint Language (OCL) and Queries/Views/Transformations (QVT) in addition to Executable UML (xUML) and Action Semantic Language (ASL). These standards are less ubiquitous in the MDA tools available and are still under development.

2.2.4 Transformations

Four levels of transformations have been defined by the OMG: manual transformations, using a profile, patterns and marking, and automatic transformations [21]. The ultimate goal of MDA tools is to maximise the automatic transformation of models, however it is recognised that this depends upon the maturity of the tools and the users' and tool builders' experience [23].

It is difficult to see how manual transformation differs from 'traditional' development and how it can be classed as an MDA approach. However Seidewitz [36] argues that using MDA and manual transformations gives two benefits; the explicit distinction between models and a record of the transformations. Another advantage of the manual transformations is that there is no need to purchase or to learn transformation tools [36]. Manual transformation, whilst having some value, has no need for the support of a tool, therefore this form of transformation is outside the scope of this paper.

To be able to automatically transform a model it needs to be written in a *well defined language* [37]. An MDA model must have a formally defined syntax and semantics or it is not considered to be a model, just an *informal diagram* [19, 24, 27]. To totally automate transformations is complex and requires significant detail to be added by the user for example how would you add non-functional requirements, so that they can be transformed in the PIM-to-PSM transformation? [29]. Consequently there needs to be a mechanism by which users can enrich the models and add more detail about both functional and non-functional requirements. It has been accepted by OMG and practitioners that whilst complete automatic transformations are the goal, it will probably not be realised for the foreseeable future and transformations will have to be enhanced by humans [27]. MDA tools will have to support this 'human intervention', allowing users to manipulate the models at various stages and maintaining these changes in addition to retaining the added information when reverse engineering or propagating model change.

Transformations are about altering one model to another model; a CIM should be transformable to a PIM, a PIM to a PSM and a PSM to code. Therefore MDA tools that allow a user to model a PIM level model should support the transformation of that model to the next level, i.e. PSM. Also if an MDA tool is to support reverse engineering it should support code transformation to a PSM.

2.2.5 Lifecycle

Another key aspect of OMG's defined MDA is that a tool should support the complete lifecycle [24, 27] so analysis and design, programming (testing, component build or component assembly), deployment and management should be covered by the tool. As MDA covers the whole software development lifecycle of an application a tool should either cover the entire lifecycle, or position itself explicitly within the lifecycle giving information about imports and exports to make it part of a tool chain.

The typical lifecycle as described by OMG [27] and others [22] is shown in Figure 1.

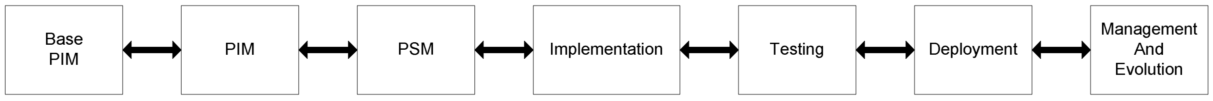


Figure 1: The lifecycle as described by the OMG

A tool's application to these stages needs to be explored. The OMGs' RFI on MDA Tools [18] has developed a taxonomy of the various types of tool many which could apply to the lifecycle stages, including a modelling tool, an analysis tool, a transformation tool, a test tool and a requirement tool. Clearly the lifecycle is an important issue in the development of a tool to support the MDA process. Related to the issue of lifecycle coverage, and with the rise in popularity of agile methods is the tool's support for methodologies. , Does a tool support agile development or are does it support a more 'traditional approach'.

2.2.6 Reverse engineering

If everything in MDA is about models, models generated by domain experts, refined by systems experts, and transformed into code, then it should be possible to take any model and transform it into any other – ie both backwards and forwards.

2.2.7 Technology support

There is a vast range of architectures, platforms and technologies that could be supported by MDA tools; however the majority support a few major ones. The most popular development languages are the most likely to be supported by the tools, Java and C are likely to be the main programming languages. Other technical support is likely to be for CORBA, J2EE, .NET and Web Services. The most likely databases to be supported are Oracle and MS SQLServer.

Generated code has, amongst programmers, generally had a poor reputation. It is seen as clumsy, long-winded and less efficient [29]and most developers believe they can develop better code than a code generator [35]. The OMG should consider emphasising additional features such as code debuggers, code configuration and regeneration of code that would give additional support to the programmers for MDA to ensure its take up in the wider market.

2.2.7.1 Technical Users

One of the often cited reasons for MDA development and MDA tool support is because the available technologies have multiplied and have become increasingly complex, it has become harder and harder for the 'technologists' to keep up. However, the MDA literature acknowledges that complete automatic transformation is, whilst the ultimate goal of MDA, still some way off. As a result the requirements are for programmers to actually have to know more, rather than less. They have to learn to use the MDA tool, to understand the patterns and profiles it applies and be able to look at, understand and modify the generated code. Thus they may need to know, Java, J2EE, struts, UML and MDA [29].

2.2.8 Tool functionality

2.2.8.1 Tool Features

A range of tool requirements could be specified, the most common set include business modelling, model transformation, artefact generation, integration of legacy applications and tool integration [22, 23, 26]. There are a range of other features that tool developers could provide such as; scalability and multi-user tools, version control, document generation, change management, re-usable components, model integrity/consistency checking.

2.2.8.2 Vendor dependence

One problem of using MDA tools is the potential for becoming dependent upon the vendor of that tool [29]. However, as long as the tool uses the available standards and allows for import and export some of this over-reliance can be ameliorated. This of course increases the importance of tools adhering to and applying the relevant standards. However many tools tend to operate on models that conform to their own internal standards, thus the models are not easily exportable. Users are therefore often locked into using their tool throughout the entire lifecycle [31]. It is the model transformations which are seen as the key to unlocking this vendor lock-in issue and allow the transfer of models from one tool to another [31].

2.2.8.3 Status of the company and the tool

Another issue surrounding the tool itself is its status. Is it a commercial application being sold to users with support and training? Is it a research application that is being used to explore the concept and application of MDA? Is it an Open Source tool? Also of interest is the Company's relationship to the OMG. Companies that are members of, or affiliated with, the OMG will be much more likely to reflect the language and principles of MDA as expressed in the OMG documentation, thus it is more likely that their MDA tools will be found to be '*more MDA'* than tools developed by companies that are not OMG members.

2.2.8.4 Tool Taxonomy

In the OMG's Request for Information about MDA Tool Capabilities [18] they develop a ten point taxonomy to classify the various types of MDA tools: Modelling Tool, Analysis Tool, Transformation Tool, Composition Tool, Test Tool, Simulation Tool, Metadata Management Tool, Reverse Engineering Tool, Requirements Tool and Version Control Tool. Whilst this list is indicative and intended as a guide for the respondents it is not clear how a tool that meets a subset of these criteria could be labelled an MDA tool. If for example; a product is a Test Tool and has no other functionality can it be an MDA tool? Does a tool have to meet more than one of the criteria? Can a tool be 'more' MDA than another tool by ticking more of the criteria? The OMG RFI does define an MDA tool as "a tool used to develop, interpret, and/or transform models", but this would also imply that an MDA tool has to do nothing more than model. This leaves the question of whether a modelling tool alone can be an MDA tool?

2.2.9 Summary of Requirements

Interoperability

Portability

Reusability

Models
MDA Specific Models
- Meta-models
- CIM
- PIM
- PSM
- Code (generated)
- Executable UML

Type of Model - UML
- Class
- Object
- Composite
- Package
- Component
- Deployment
- Use Case
- Communication
- Sequence
- Interaction
- Activity
- State
- Timing

Other Models
- BPMN
- BPEL
- ER Models
- Informal diagrams
- Dataflow diagrams
- Structure charts
- Architecture
- Logical model
- Physical model
- Business logic/rules
- Interface diagram

General model properties
- Model differencing and merging
- Model import and export
- Model consistency/integrity/testable
- Model navigation/browse
- Traceability
- Storage
- Model/code synchronisation
- Executable models

Standards
MDA Standards:
- UML (2.0)
- UML Profiles
- MOF
- XMI
- CWM
- OCL
- MDA
- OMG
- QVT
- xUML
- ASL

Transformations
MDA transformations
- CIM -> PIM
- PIM -> PSM
- PSM -> Code

General transformations
- Model -> model
- Model -> code
- Logical -> physical
- code -> model
- code generation

Lifecycle support
- SDLC Support
- Agile
- RAD
- Iterative development

Technology support
- List of technologies
- Component based development
- Pattern based development
- Template based development

Database Supported
- List of databases

Tool features	OMG's Tool Taxonomy
▪ Scalable	▪ Modelling
▪ Multi-user	▪ Analysis
▪ Document generation	▪ Transformation
▪ Change management	▪ Composition
▪ Regenerate code	▪ Test
▪ User interface	▪ Simulation
▪ Round-trip-engineering	▪ Metadata Management
▪ Status of the tool	▪ Reverse Engineering
▪ Status of the company	▪ Requirements
	▪ Version Control

Table 2: Summary of MDA Tool requirements

2.3 Review of Existing MDA Tools

The review has taken the form of exploring the data on the OMG website and of investigating the companies' websites and their publicity material for the MDA tools. Therefore this review cannot claim to be exhaustive or wholly objective; it is just an overview of the various 'publicised' capacities of the tools based on the companies' opinions. This will generate a reference point as to what companies' think are the important and 'sellable' components of an MDA tool and therefore help to gauge the perceived significant components of an MDA tool. It should also show areas of weakness, that is areas that have been largely ignored by the tool vendors and thus potential areas for development.

The OMG webpage containing the list of tools [38] and all the companies listed web sites (see Table 3) were accessed over a several months up to January 2008. The 57 '*Committed Companies and Their Products*' listed have been reviewed and compared iteratively against the summary of requirements listed above. In other words, one has informed the other and vice-versa. Listings have been removed where there is no MDA tool developed by the company, or the company appears to have ceased trading or there is too little information to properly assess the tools functionality, so only 46 tools remain to be reviewed.

This is a review more of the data sheets and the web sites than it is of the tools themselves consequently it will reflect the features mentioned and the language used by the respective companies not necessarily the functionality of the MDA tool itself.

2.3.1 Listing from the OMG Web Site

	Tool	Company	url
1	Adaptations [39]	Adaptive Inc	http://www.adaptive.com
2	Ameos [40]	Aonix	http://www.aonix.com
3	Real Time Studio [41]	ARTiSAN	http://www.artisansw.com
~~4~~	~~b+m ArchitectureWare~~		~~http://www2.architectureware.de/~~
5	smartGenerator [42]	BITPlan	http://www.bitplan.com
6	Together 2006 [15]	Borland	http://www.borland.com
7	Caboom [43]	Calkey Technologies	http://www.calkey.com/caboom.htm
8	SIMplicity [44]	Calytrix	http://www.calytrix.com
9	Codagen Architect [13]	Codagen Technologies	http://www.manyeta.com
10	Codeless [17]	Codeless Technology	http:www.codeless.com
~~11~~	~~No Tool~~	~~Consortium for Business Object Promotion~~	
12	REP ++ Studio [45]	Consyst	http://www.consyst-sql.com/a/WWW/Accueil/Accueil.html
13	OptimalJ [11]	Compuware	http://www.compuware.com
14	Component-X [46]	Data Access Technologies - Model Driven Solutions	http://www.enterprise-component.com
15	~~No Tool~~	~~David Frankel Consulting~~	~~http://www.davidfrankelconsulting.com~~
16	CodeGenie [47]	Domain Solutions	http://www.ooagenerator.com/codegenie.htm or http://www.domsols.com
17	Constructor/MDRAD [48]	Dot Net Builders	http://www.dotnetbuilders.com
~~18~~	~~TET~~	~~EDCubed~~	~~http://www.edcubed.com reloads to http://www.abovo.com/en/home~~
19	Bridge [49]	E2E	http://www.e2ebridge.com
20	e-GEN [50]	Gentastic	http://www.gentastic.com/e_GEN/e_GenApproach.html
21	~~MDE - DEFUNCT~~	~~M1 Global Solutions~~	~~http://www.m1global.org/index.html or http://www.m1global.com./~~
22	~~No Tool~~	~~Hendryx & Associates~~	~~http://www.hendryxassoc.com/index.html~~

	Tool	Company	url
23	~~No Tool~~	~~Herzum Software~~	~~http://www.herzumsoftware.com/~~
24	Rational Software Architect [6]	IBM	http://www-306.ibm.com/software/awdtools/architect/swarchitect/
25	Medini product family(was m2c) [51]	IKV++	http://www.ikv.de/
26	Rhapsody [52]	Telelogic (was I-Logix)	http://modeling.telelogic.com/
27	iQgen [53]	innoQ	http://www.innoq.com/iqgen/
28	ArcStyler [8]	Interactive Objects Software	http://www.interactive-objects.com/products
29	Kabira Transaction Platform and Kabira Accelerator [54]	Kabira Technologies, Inc	http://www.kabira.com/
30	CASSANDRA/xUML [12]	KnowGravity	http://www.knowgravity.com/eng/index.htm
31	iUML [55]	Kennedy Carter Ltd	http://www.kc.com/
31	iCCG [56]	Kennedy Carter Ltd	http://www.kc.com/
32	Xcoder [57]	LIANTIS	http://www.liantis.com/
33	~~No Tool~~	~~M2VP's MDA Consulting Services~~	~~http://www.m2vp.com/~~
34	~~No Tool~~	~~MASTER Project~~	~~http://www.esi.es/Master~~
35	BridgePoint/xtUML or EDGE UML Suite [58]	Mentor Graphics	http://www.mentor.com/
36	MetaMatrix Data Services Platform [59]	JBoss (MetaMatrix Commitment) bought April 2007	http://www.redhat.com/metamatrix/
37	modelscope [16]	Metamaxim	http://www.metamaxim.com/
38	Model-In-Action [9]	Mia-Software	http://www.mia-software.com
39	Innovator [60]	MID	http://www.mid.de/
40	~~No Tool~~	~~The MOD Group~~	~~http://www.themodgroup.com/~~
41	BoldExpress Studio [4]	Neosight Technologies	http://www.neosight.com./
42	Blu Age [61]	Netfective	http://www.bluage.com/
43	~~No Tool~~	~~OCI's MDA Services~~	~~http://www.ociweb.com/consulting/mda.html~~
44	FrontierSuite [62]	ObjectFrontier	http://www.objectfrontier.com/
45	PowerRAD [63]	Outline Systems Inc.	http://www.outlinesys.com/

	Tool	Company	url
46	PathMATE [3]	Pathfinder Solutions	http://www.pathfindermda.com/
47	Agora Plastic 2005 [64]	Plastic Software	http://www.plasticsoftware.com/
48	Framework [65]	realMethods	http://www.realmethods.com/index.html
49	Select Component Factory or Select Solution for MDA [7]	Select Business Solutions	http://www.selectbs.com/
50	MetaBoss [66]	Softaris Pty. Ltd.	www.metaboss.com
51	Generative Model Transformer project [5]	SoftMetaWare	http://www.softmetaware.com/
52	Objecteering [2]	Softeam	http://www.objecteering.com/
53	OlivaNova Model Execution Sys [67]	CARE Technologies S.A. / SOSY Inc's	http://www.sosyinc.com
54	Enterprise Architect [68]	Sparx Systems	http://www.sparxsystems.com
55	MasterCraft [69]	Tata Consultancy Services	http://www.tatamastercraft.com/index.htm / http://www.tata.com/index.htm
56	TAU Generation2 [14]	Telelogic	http://www.telelogic.com/
57	ACE [70]	TechOne	http://www.techone.com/

Table 3: Summary of MDA committed tools

2.3.2 Review Results

2.3.2.1 Models

	MDA Specific Models							
		Meta-models	*CIM*	*PIM*	*PSM*	*Code (generated)*	*xUML*	
1	Adaptations		✓	✓	✓	✓		4
2	Ameos	✓		✓	✓	✓		4
3	Real Time Studio			✓	✓	✓		3
5	smartGenerator			✓	✓	✓		3
6	Together 2006			✓	✓	✓		3
7	Caboom					✓		1
8	SIMplicity			✓	✓	✓		3
9	Codagen Architect			✓	✓	✓		3
10	Codeless			✓				1
12	REP ++ Studio	✓		✓	✓	✓		4
13	OptimalJ			✓	✓	✓		3
14	Component-X					✓		1
16	CodeGenie	✓		✓	✓	✓		4
17	Constructor/MDRAD			✓	✓	✓	✓	4
19	Bridge							0
20	e-GEN	✓		✓	✓	✓		4
24	Rational Software Architect					✓		1
25	Medini product family(was m2c)	✓				✓		2
26	Rhapsody			✓		✓		2
27	iQgen	✓		✓	✓	✓		4
28	ArcStyler					✓		1
29	Kabira Transaction Platform and							0
30	CASSANDRA/xUML			✓	✓	✓	✓	4
31	iUML	✓		✓	✓	✓	✓	5
31	iCCG					✓	✓	2
32	Xcoder			✓		✓		2
35	BridgePoint/xtUML or EDGE UML Suite	✓		✓	✓	✓	✓	5
36	MetaMatrix Data Services Platform					✓		1
37	modelscope							0
38	Model-In-Action							0
39	Innovator					✓		1
41	BoldExpress Studio					✓		1
42	Blu Age					✓		1
44	FrontierSuite					✓		1

45	PowerRAD	✓	✓	✓		✓		4
46	PathMATE	✓		✓	✓	✓		4
47	Agora Plastic 2005	✓		✓	✓	✓		4
48	Framework			✓	✓	✓		3
49	Select Component Factory or Select		✓	✓	✓	✓		4
50	MetaBoss			✓		✓		2
51	Generative Model Transformer			✓	✓			2
52	Objecteering					✓		1
53	OlivaNova Model Execution System					✓		1
54	Enterprise Architect			✓	✓	✓		3
55	MasterCraft	✓				✓		2
56	TAU Generation2					✓		1
57	ACE				✓	✓		2
47		12	3	27	23	41	5	

Table 4: MDA specific models

As can be seen in Table 4 one of the most interesting of these results is the very low number of tools that mention a CIM model, this reflects the problem both with the modelling and the transformation of CIM to PIM models, such that most tools avoid the concept altogether. Another interesting result is that very few tools use meta-models. These are seen by MDA as the foundation of transformations and of interoperability. Also very few tools use executable UML.

Pre-CIM	Users: **All**	**MUST**
GUI REQ ID 1	Development of an informal Pre-CIM model should be supported	
Related requirement IDs: 1, 2, 3, 16, 17, 18, 19, 20, 21 ,22, 23, 24, 25, 26, 27, 28, 35, 60, 64, 69		

CIM	Users: **All**	**MUST**
GUI REQ ID 2	Development of a CIM model should be supported	
Related requirement IDs: 1, 2, 3, 16, 17, 18, 19, 20, 21 ,22, 23, 24, 25, 26, 27, 28, 35, 60, 64, 69		

PIM	Users: **All**	**MUST**
GUI REQ ID 3	Development of a PIM model should be supported	
Related requirement IDs: 1, 2, 3, 16, 17, 18, 19, 20, 21 ,22, 23, 24, 25, 26, 27, 28, 35, 60, 64, 69		

UML Models																
		UML	***Class***	***Object***	***Composite***	***Package***	***Component***	***Deployment***	***Use Case***	***Communication***	***Sequence***	***Interaction***	***Activity***	***State***	***Timing***	
1	Adaptations	✓														1
2	Ameos	✓														1
3	Real Time Studio	✓									✓					2
5	smartGenerator	✓														1
6	Together 2006	✓														1
7	Caboom	✓		✓					✓			✓	✓			5
8	SIMplicity	✓	✓	✓				✓								4
9	Codagen Architect	✓	✓						✓		✓		✓	✓		6
10	Codeless	✓	✓											✓		3
12	REP ++ Studio	✓														1
13	OptimalJ	✓														1
14	Component-X	✓														1
16	CodeGenie	✓	✓											✓		3
17	Constructor/MDRAD	✓	✓	✓												3
19	Bridge	✓	✓					✓	✓		✓		✓	✓		7
20	e-GEN	✓														1
24	Rational Software Architect	✓	✓					✓	✓		✓		✓	✓		7
25	Medini product family(was m2c)	✓														1
26	Rhapsody	✓		✓							✓			✓		4
27	iQgen	✓														1
28	ArcStyler	✓	✓										✓			3
29	Kabira Transaction Platform and	✓	✓										✓	✓		4
30	CASSANDRA/xUML	✓	✓						✓		✓			✓		5
31	iUML	✓														1
31	iCCG	✓														1
32	Xcoder	✓														1
35	BridgePoint/xtUML or EDGE UML Suite	✓			✓		✓		✓		✓					5
36	MetaMatrix Data Services Platform	✓														1
37	modelscope	✓												✓		2
38	Model-In-Action	✓		✓	✓		✓	✓	✓		✓		✓	✓		9
39	Innovator	✓	✓	✓	✓	✓	✓		✓		✓		✓	✓		10
41	BoldExpress Studio	✓														1
42	Blu Age	✓														1
44	FrontierSuite	✓														1
45	PowerRAD	✓		✓												2
46	PathMATE	✓														1
47	Agora Plastic 2005	✓														1
48	Framework	✓														1
49	Select Component Factory	✓	✓	✓	✓	✓	✓	✓	✓		✓		✓			10
50	MetaBoss	✓												✓		2

51	Generative Model Transformer															0
52	Objecteering	✓							✓				✓			3
53	OlivaNova Model Execution System	✓	✓	✓												3
54	Enterprise Architect	✓	✓	✓	✓	✓	✓	✓	✓		✓		✓	✓		11
55	MasterCraft	✓														1
56	TAU Generation2	✓														1
57	ACE	✓														1
		46	14	10	5	3	5	6	11	0	11	1	11	13	0	

Table 5: UML Models

All tools use some UML models as can be seen in Table 5, however, the 'Generative Model Transformer' has very little information available and it is not clear if the tool has actually been developed yet. Some tools do state that all UML models are supported; Enterprise Architect, Select and Innovator cite the most comprehensive list. Class and State diagrams are the most often mentioned models used in the tools.

Any MDA tool should support UML, this is the foundation of the OMG's MDA Standards and it would be expected that an MDA tool was UML based. Whether a tool supported all 13 of the UML models and also if UML is the only notation used is a different matter. Class diagrams are the most popular and typically the most often used by developers.

CIM | **PIM** | **PSM**	Users: **All**	**MUST**
GUI REQ ID 4	UML must be supported.	
Related requirement IDs: 4, 5, 11		

CIM | **PIM** | **PSM**	Users: **All**	**MUST**
GUI REQ ID 5	Class Diagram modelling, creation, amendment and deletion.	
Related requirement IDs: 4, 5, 11		

		Other Models											
		BPMN	***BPEL***	***ER Models***	***Informal diagrams***	***Dataflow diagrams***	***Structure charts***	***Architecture***	***Logical model***	***Physical model***	***Business logic/rules***	***interface diagram***	
1	Adaptations												0
2	Ameos												0
3	Real Time Studio							✓					1
5	smartGenerator												0
6	Together 2006	✓	✓						✓	✓			4
7	Caboom										✓		1
8	SIMplicity												0
9	Codagen Architect												0
10	Codeless												0
12	REP ++ Studio												0
13	OptimalJ										✓		1

14	Component-X										✓		1
16	CodeGenie												0
17	Constructor/MDRAD										✓		1
19	Bridge												0
20	e-GEN												0
24	Rational Software Architect			✓	✓								2
25	Medini product family(was m2c)												0
26	Rhapsody												0
27	iQgen												0
28	ArcStyler				✓								1
29	Kabira Transaction Platform and		✓										0
30	CASSANDRA/xUML												0
31	iUML												0
31	iCCG												0
32	Xcoder												0
35	BridgePoint/xtUML or EDGE UML Suite											✓	1
36	MetaMatrix Data Services Platform												0
37	modelscope										✓		1
38	Model-In-Action												0
39	Innovator	✓		✓		✓	✓						4
41	BoldExpress Studio												0
42	Blu Age												0
44	FrontierSuite			✓									1
45	PowerRAD										✓		1
46	PathMATE												0
47	Agora Plastic 2005												0
48	Framework												0
49	Select Component Factory	✓											1
50	MetaBoss										✓		1
51	Generative Model Transformer												0
52	Objecteering	✓											1
53	OlivaNova Model Execution System										✓		1
54	Enterprise Architect			✓									1
55	MasterCraft										✓		1
56	TAU Generation2												0
57	ACE												0
		4	2	4	2	1	1	1	1	1	9	1	

Table 6: Other models

Some tools do support other models although there is no clear trend as can be seen in Table 6. Business rules and logic, although how expressed is not always clear, are the most cited ‘other model’ supported by the tools.

General Model Properties											
		Model differencing	***Model merging***	***Model import and export***	***Model consistency/integrity/testable***	***Model Navigation/traversable***	***Traceability***	***Model Repository***	***Model/code synchronisation***	***Executable models***	
1	Adaptations			✓				✓			2
2	Ameos							✓			1
3	Real Time Studio				✓	✓			✓		3
5	smartGenerator										0
6	Together 2006	✓	✓		✓						3
7	Caboom				✓						1
8	SIMplicity										0
9	Codagen Architect										0
10	Codeless				✓						1
12	REP ++ Studio			✓				✓	✓		3
13	OptimalJ										0
14	Component-X										0
16	CodeGenie			✓	✓	✓					3
17	Constructor/MDRAD			✓							1
19	Bridge			✓	✓			✓			3
20	e-GEN			✓							1
24	Rational Software Architect		✓			✓	✓				3
25	Medini product family(was m2c)		✓				✓				2
26	Rhapsody			✓			✓				2
27	iQgen			✓	✓						2
28	ArcStyler			✓	✓	✓	✓				4
29	Kabira Transaction Platform and										0
30	CASSANDRA/xUML			✓	✓	✓					3
31	iUML				✓	✓					2
31	iCCG										0
32	Xcoder			✓							1
35	BridgePoint/xtUML or EDGE UML Suite				✓						1
36	MetaMatrix Data Services Platform			✓				✓			2
37	modelscope				✓					✓	2
38	Model-In-Action			✓							1
39	Innovator					✓		✓			2
41	BoldExpress Studio										0
42	Blu Age				✓						1
44	FrontierSuite			✓			✓				2
45	PowerRAD				✓						1
46	PathMATE			✓	✓	✓					3
47	Agora Plastic 2005			✓	✓	✓					3
48	Framework			✓							1

49	Select Component Factory or Select						✓		✓		2
50	MetaBoss				✓				✓		2
51	Generative Model Transformer										0
52	Objecteering	✓	✓		✓	✓	✓	✓	✓		7
53	OlivaNova Model Execution System										0
54	Enterprise Architect	✓		✓	✓	✓	✓		✓		6
55	MasterCraft			✓	✓						2
56	TAU Generation2				✓	✓	✓				3
57	ACE										0
		3	4	19	21	12	9	7	6		

Table 7: General model properties

During the review carried out in Table 7 very few products mention tool differencing or merging although clearly a useful feature. Model import and export is obviously important, for tools to be fully useful they should allow at least models to be exported so that the models can be used in other tools and thus users are not tied into the product. Many tools support model import so that other modelling software (such as Rational Rose, Together, ARIS etc) can be used and the models then imported. This reflects the concept of tool chaining that many MDA products aim at, not necessarily to offer the full functionality of an MDA tool, but of a section of it.

Not quite half the tools mention model consistency, integrity or testing, this is surprisingly low given the importance MDA places upon the modelling process. It could be that the literature does necessarily cover enough detail to discuss this type of functionality; however this has to be a critical feature for an MDA tool to support the MDA process.

It is also surprising how few tools mention traceability of models, that is the ability to trace ideas through the multiple abstractions (CIM, PIM, PSM, Code). Related to this is the idea of model/code synchronisation, clearly if changes are made to the PIM or to the code, these changes should be propagated through the multiple abstractions to maintain consistency. Also how few mention any kind of model storage, although again many more of the tools must be storing the models it is just not clear from the documentation.

CIM | **PIM** | **PSM**	Users: **All**	**MUST**
GUI REQ ID 6	Import and export of models.	
Related requirement IDs: 6, 68, 84		

CIM | **PIM** | **PSM**	Users: **All**	**MUST**
GUI REQ ID 7	Model integrity, verification, testing.	
Related requirement IDs: 7, 52, 53, 54, 55, 56		

CIM | **PIM** | **PSM**	Users: **All**	**MUST**
GUI REQ ID 8	Traceability of requirements through multiple abstractions.	
Related requirement IDs: 8, 9, 10, 15, 34a, 49, 50, 51, 58, 67, 72, 73, 74, 79, 80, 81, 83		

CIM | PIM | PSM	Users: **All**	**MUST**
GUI REQ ID 9	Propagation of changes to all levels of abstraction	
Related requirement IDs: 8, 9, 10, 15, 34a, 49, 50, 51, 58, 67, 72, 73, 74, 79, 80, 81, 83		

CIM | PIM | PSM	Users: **All**	**MUST**
GUI REQ ID 10	Model repository.	
Related requirement IDs: 8, 9, 10, 15, 34a, 49, 50, 51, 58, 67, 72, 73, 74, 79, 80, 81, 83		

2.3.2.2 Standards

MDA Standards

		UML	*UML (2.0)*	*UML Profiles*	*MOF*	*XMI*	*CWM*	*OCL*	*QVT*	*xUML*	*ASL*	*MDA*	*OMG*	
1	Adaptations	✓	✓		✓	✓	✓					✓		5
2	Ameos	✓	✓	✓		✓						✓		5
3	Real Time Studio	✓	✓	✓		✓						✓	✓	6
5	smartGenerator	✓				✓						✓	✓	4
6	Together 2006	✓	✓	✓	✓	✓		✓	✓			✓		8
7	Caboom	✓		✓										2
8	SIMplicity	✓		✓										2
9	Codagen Architect	✓		✓		✓						✓		4
10	Codeless	✓				✓								2
12	REP ++ Studio	✓												1
13	OptimalJ	✓	✓		✓	✓	✓					✓		6
14	Component-X	✓		✓			✓							3
16	CodeGenie	✓				✓								2
17	Constructor/MDRAD	✓			✓	✓						✓		4
19	Bridge	✓				✓						✓	✓	4
20	e-GEN	✓			✓	✓						✓		4
24	Rational Software Architect	✓	✓	✓								✓		4
25	Medini product family(was m2c)	✓			✓			✓	✓					4
26	Rhapsody	✓	✓	✓		✓								4
27	iQgen	✓		✓		✓						✓	✓	5
28	ArcStyler	✓		✓	✓	✓						✓	✓	6
29	Kabira Transaction Platform and	✓												1
30	CASSANDRA/xUML	✓								✓				2
31	iUML	✓								✓	✓			3
31	iCCG	✓												1
32	Xcoder	✓				✓								2

35	BridgePoint/xtUML or EDGE UML Suite	✓	✓							✓	✓			4
36	MetaMatrix Data Services Platform	✓			✓	✓	✓						✓	5
37	modelscope	✓												1
38	Model-In-Action	✓		✓	✓	✓							✓	5
39	Innovator	✓	✓	✓	✓	✓	✓						✓	7
41	BoldExpress Studio	✓		✓				✓						3
42	Blu Age	✓			✓	✓		✓				✓		5
44	FrontierSuite	✓										✓		2
45	PowerRAD	✓			✓							✓		3
46	PathMATE	✓	✓			✓						✓	✓	5
47	Agora Plastic 2005	✓		✓		✓						✓	✓	5
48	Framework	✓				✓								2
49	Select Component Factory	✓												1
50	MetaBoss	✓												1
51	Generative Model Transformer	✓										✓	✓	3
52	Objecteering	✓	✓	✓								✓		4
53	OlivaNova Model Execution System	✓		✓										2
54	Enterprise Architect	✓	✓	✓		✓						✓		5
55	MasterCraft	✓		✓										2
56	TAU Generation2	✓	✓									✓		3
57	ACE	✓				✓						✓		3
		47	13	19	12	25	5	4	2	3	2	23	11	

Table 8: MDA standards

All tools' documentation mentions the UML standard (see Table 8), which is to be expected, given that the modelling support is in all cases UML. Some mention UML 2 (both 2.0 and 2.1) and many rely on UML Profiles to tailor and annotate the models. The MOF standard is only mentioned in 12 of the tools, this matches with the few (7) tools that mention the repository given that the two are closely linked. XMI is more widespread (25), often because tools are importing models from other modelling software and XMI is the standard for exchanging such UML models and it corresponds to the 19 tools that can import models. The other standards of CWM, OCL, QVT, xUML, ASL are not well supported. The most notable of this list is CWM, which is sited as the other core MDA standard for data warehouse modelling, is hardly supported at all.

The MDA and OMG standards were added simply to see how many tools claimed to support the 'MDA standard' or the 'OMG standard' and it was interesting to note that not all tools claimed to. Virtually all documentation mentioned MDA (but they would as many of the data sheets were prepared for the OMG MDA page), but not necessarily making the claim to meeting the 'standard'.

CIM	Users: **All**	**MUST**
GUI REQ ID 11	An open and interoperable platform that meets as many standards as possible	
Related requirement IDs: 11		

Other Standards						
		EDOC	***RAS***	***SYSML***	***JMI***	
1	Adaptations	✓	✓			2
2	Ameos					0
3	Real Time Studio			✓		1
5	smartGenerator					0
6	Together 2006					0
7	Caboom					0
8	SIMplicity					0
9	Codagen Architect					0
10	Codeless					0
12	REP ++ Studio					0
13	OptimalJ					0
14	Component-X	✓				1
16	CodeGenie					0
17	Constructor/MDRAD					0
19	Bridge					0
20	e-GEN					0
24	Rational Software Architect		✓			1
25	Medini product family(was m2c)					0
26	Rhapsody			✓		1
27	iQgen					0
28	ArcStyler				✓	1
29	Kabira Transaction Platform and					0
30	CASSANDRA/xUML					0
31	iUML					0
31	iCCG					0
32	Xcoder					0
35	BridgePoint/xtUML or EDGE UML Suite					0
36	MetaMatrix Data Services Platform					0
37	modelscope					0
38	Model-In-Action					0
39	Innovator					0
41	BoldExpress Studio					0
42	Blu Age					0
44	FrontierSuite					0
45	PowerRAD					0
46	PathMATE					0
47	Agora Plastic 2005					0
48	Framework					0

49	Select Component Factory or Select					0
50	MetaBoss					0
51	Generative Model Transformer					0
52	Objecteering					0
53	OlivaNova Model Execution System					0
54	Enterprise Architect					0
55	MasterCraft					0
56	TAU Generation2					0
57	ACE					0
		2	2	2	1	

Table 9: Other standards

Other standards were mentioned in tools, but there were none of any significance (see Table 9).

2.3.2.3 Transformations

MDA Transformations

		CIM -> PIM	*PIM -> PSM*	*PSM -> Code*	
1	Adaptations				0
2	Ameos				0
3	Real Time Studio			✓	1
5	smartGenerator		✓		1
6	Together 2006		✓		1
7	Caboom		✓		1
8	SIMplicity		✓		1
9	Codagen Architect		✓		1
10	Codeless				0
12	REP ++ Studio				0
13	OptimalJ		✓	✓	2
14	Component-X				0
16	CodeGenie				0
17	Constructor/MDRAD		✓	✓	2
19	Bridge				0
20	e-GEN		✓	✓	2
24	Rational Software Architect				0
25	Medini product family(was m2c)				0
26	Rhapsody				0
27	iQgen				0
28	ArcStyler				0
29	Kabira Transaction Platform and				0
30	CASSANDRA/xUML				0
31	iUML				0
31	iCCG		✓		1
32	Xcoder				0

35	BridgePoint/xtUML or EDGE UML Suite		✓	✓	2
36	MetaMatrix Data Services Platform				0
37	modelscope				0
38	Model-In-Action				0
39	Innovator				0
41	BoldExpress Studio				0
42	Blu Age				0
44	FrontierSuite				0
45	PowerRAD				0
46	PathMATE		✓	✓	2
47	Agora Plastic 2005		✓	✓	2
48	Framework		✓	✓	2
49	Select Component Factory	✓	✓	✓	3
50	MetaBoss				0
51	Generative Model Transformer				0
52	Objecteering				0
53	OlivaNova Model Execution System				0
54	Enterprise Architect		✓	✓	2
55	MasterCraft		✓		1
56	TAU Generation2				0
57	ACE		✓	✓	2
		1	17	11	

Table 10: MDA Transformations

Table 10 shows that the number of tools that stated they had CIMs, PIMs and PSMs transformation was quite low in number. However, it is interesting to note that of the 3 tools that reported the use of a CIM model only 1 tool explicitly mentions the transformation of CIM to PIM. 27 tools stated they used PIMs, however only 17 transform these PIMs to PSMs and 23 tools mention PSM, but only 11 transform PSMs to code. Clearly there is not a one to one relationship of using a model and transforming the model to the next level of abstraction. Although as stated previously this could be a restriction of the data sheets and web site information more than of the tool itself.

Of those tools that do perform MDA transformations a high number also support meta-modelling, however there are some tools that apparently transform PIMs to PSMs and PSMs to code without the support of meta-models.

Other Transformations							
		Model -> model	***Model -> code***	***Logical -> physical***	***code -> model***	***code generation***	
1	Adaptations						0
2	Ameos		✓				1
3	Real Time Studio		✓				1
5	smartGenerator						0
6	Together 2006			✓			1
7	Caboom			✓			1
8	SIMplicity						0
9	Codagen Architect	✓	✓				2
10	Codeless						0
12	REP ++ Studio						0

13	OptimalJ						0
14	Component-X						0
16	CodeGenie		✓			✓	2
17	Constructor/MDRAD						0
19	Bridge		✓				1
20	e-GEN						0
24	Rational Software Architect		✓		✓	✓	3
25	Medini product family(was m2c)	✓	✓		✓		3
26	Rhapsody		✓				1
27	iQgen		✓				1
28	ArcStyler	✓	✓				2
29	Kabira Transaction Platform and		✓				1
30	CASSANDRA/xUML						0
31	iUML						0
31	iCCG						0
32	Xcoder		✓				1
35	BridgePoint/xtUML or EDGE UML Suite		✓				1
36	MetaMatrix Data Services Platform		✓				1
37	modelscope		✓				1
38	Model-In-Action	✓	✓		✓		3
39	Innovator		✓				1
41	BoldExpress Studio		✓				1
42	Blu Age		✓				1
44	FrontierSuite	✓	✓				2
45	PowerRAD		✓				1
46	PathMATE		✓				1
47	Agora Plastic 2005					✓	1
48	Framework						0
49	Select Component Factory or Select						0
50	MetaBoss		✓				1
51	Generative Model Transformer						0
52	Objecteering	✓	✓				2
53	OlivaNova Model Execution System		✓				1
54	Enterprise Architect						0
55	MasterCraft	✓					1
56	TAU Generation2		✓				1
57	ACE						0
47		7	26	2	3	3	

Table 11: Other transformations

Of course it could be a matter of how companies report the transformations, i.e. not using the phrasing ‘PSM to code’ but a more generic ‘model to code’ transformation. Many tools are reported as doing model to code transformations, in some there are duplications (i.e. tools mention both PSM to code and model to code), however many state either one or the other, thus most tools do claim to generate code from models. Although some make a feature of not generating code e.g. CodeLess which *“Unlike the others, it chooses not to generate code at all.”* [17]

What is still surprisingly low is the model to model transformations. Only 7 tools claim to do this, so it is not clear how each level of abstraction is realised if there is no model transformation, i.e. how do you get from CIM to PIM or from PIM to PSM?

Pre-CIM | **CIM** |	Users: **All**	**MUST**

GUI REQ ID 12	Transformation of the Pre-CIM to CIM model
Related requirement IDs: 12, 12a, 25	

CIM | **PIM** |	Users: **All**	**MUST**
GUI REQ ID 12a	Transformation of the CIM to PIM model	
Related requirement IDs: 12, 12a, 25		

2.3.2.4 Methodology

Methodology Support					
		SDLC Support	***Agile***	***Iterative development***	
1	Adaptations				0
2	Ameos				0
3	Real Time Studio				0
5	smartGenerator				0
6	Together 2006				0
7	Caboom				0
8	SIMplicity				0
9	Codagen Architect				0
10	Codeless				0
12	REP ++ Studio				0
13	OptimalJ		✓		1
14	Component-X				0
16	CodeGenie				0
17	Constructor/MDRAD		✓		1
19	Bridge				0
20	e-GEN	✓			1
24	Rational Software Architect	✓			1
25	Medini product family(was m2c)				0
26	Rhapsody	✓	✓		2
27	iQgen				0
28	ArcStyler				0
29	Kabira Transaction Platform and				0
30	CASSANDRA/xUML	✓			1
31	iUML				0

31	iCCG				0
32	Xcoder				0
35	BridgePoint/xtUML or EDGE UML Suite				0
36	MetaMatrix Data Services Platform				0
37	modelscope		✓		1
38	Model-In-Action	✓			1
39	Innovator				0
41	BoldExpress Studio				0
42	Blu Age				0
44	FrontierSuite	✓			1
45	PowerRAD	✓	✓		2
46	PathMATE				0
47	Agora Plastic 2005				0
48	Framework				0
49	Select Component Factory or Select				0
50	MetaBoss			✓	1
51	Generative Model Transformer				0
52	Objecteering	✓			1
53	OlivaNova Model Execution System				0
54	Enterprise Architect	✓			1
55	MasterCraft	✓			1
56	TAU Generation2	✓			1
57	ACE				0
47		11	5	1	

Table 12: Methodology support

As can be seen in Table 12 the majority of tools don't mention methodology support or process support at all. The most mentioned, at only 11, is the general SDLC support, and most tools that do mention it are not suggesting a process to be followed, just that the tool covers a range of steps in the development process. This is to be expected the MDA approach does not specify a method and users are left to follow their own chosen process. Whether tools expect, suggest or work best with a specific methodology is not clear from the documentation and beyond the scope of this paper.

2.3.2.5 Technology

Technology – programming (Part 1)																	
		Java	***J2EE***	***EJB***	***Struts***	***JSP***	***Java Data Objects (JDO)***	***Java Transparent Persistence Framework***	***Java Server Faces***	***JDBC***	***C***	***C++***	***VisualBasic/VB***	***COBOL***	***Delphi-Pascal***	***Fortran***	***Ada/Spark Ada***
1	Adaptations	✓				✓					✓		✓				
2	Ameos	✓		✓							✓	✓					✓
3	Real Time Studio	✓									✓	✓					✓

5	smartGenerator	✓	✓	✓							✓	✓		✓	✓		
6	Together 2006	✓										✓					
7	Caboom	✓	✓	✓		✓				✓			✓				
8	SIMplicity	✓										✓					
9	Codagen Architect	✓	✓	✓	✓	✓	✓	✓				✓	✓				
10	Codeless																
12	REP ++ Studio	✓	✓			✓						✓	✓				
13	OptimalJ		✓	✓	✓												
14	Component-X	✓	✓	✓													
16	CodeGenie	✓	✓									✓					
17	Constructor/MDR AD												✓				
19	Bridge	✓															
20	e-GEN		✓														
24	Rational Software Architect	✓	✓		✓				✓		✓	✓					
25	Medini product family(was m2c)	✓	✓														
26	Rhapsody	✓									✓	✓					✓
27	iQgen	✓	✓			✓					✓	✓		✓			
28	ArcStyler		✓	✓	✓												
29	Kabira Transaction Platform and	✓															
30	CASSANDRA/xUML																
31	iCCG/iUML			✓													
32	Xcoder	✓	✓	✓										✓			
35	BridgePoint/xtUML or EDGE UML Suite										✓	✓					
36	MetaMatrix Data Services Platform	✓								✓							
37	modelscope																
38	Model-In-Action	✓	✓	✓	✓	✓						✓	✓				✓
39	Innovator	✓										✓	✓	✓			
41	BoldExpress Studio														✓		
42	Blu Age		✓														
44	FrontierSuite	✓	✓	✓			✓										
45	PowerRAD		✓														
46	PathMATE	✓									✓	✓					
47	Agora Plastic 2005	✓	✓									✓					
48	Framework		✓		✓												
49	Select Component Factory or Select	✓		✓									✓				
50	MetaBoss																
51	Generative Model Transformer																
52	Objecteering	✓										✓				✓	
53	OlivaNova Model Execution System			✓		✓			✓				✓				
54	Enterprise Architect	✓		✓								✓	✓				
55	MasterCraft		✓	✓		✓						✓					
56	TAU Generation2	✓									✓	✓					

57	ACE		✓	✓		✓											
		28	21	16	6	9	2	1	2	2	10	20	10	4	2	1	4

Technology – programming (Part 2)

		.NET	*C#*	*ASP/ASP+*	*PHP*	*Web Services*	*HTML*	*SOAP*	*UDDI*	*XML*	*ebXML*	*Cold Fusion*	
1	Adaptations							✓		✓			6
2	Ameos	✓											6
3	Real Time Studio									✓			5
5	smartGenerator	✓					✓						9
6	Together 2006	✓											3
7	Caboom	✓	✓	✓				✓	✓	✓			12
8	SIMplicity												2
9	Codagen Architect	✓	✓										11
10	Codeless	✓											1
12	REP ++ Studio	✓	✓	✓		✓							9
13	OptimalJ												3
14	Component-X	✓						✓		✓	✓		7
16	CodeGenie												3
17	Constructor/MDRAD	✓	✓										3
19	Bridge	✓											2
20	e-GEN	✓				✓							3
24	Rational Software Architect		✓			✓							8
25	Medini product family(was m2c)												2
26	Rhapsody												4
27	iQgen	✓	✓							✓			9
28	ArcStyler	✓	✓			✓							6
29	Kabira Transaction Platform and	✓											2
30	CASSANDRA/xUML									✓			1
31	iCCG/iUML						✓			✓			3
32	Xcoder	✓	✓										6
35	BridgePoint/xtUML or EDGE UML Suite												2
36	MetaMatrix Data Services Platform					✓		✓		✓			5
37	modelscope												0
38	Model-In-Action	✓	✓	✓	✓			✓		✓			14
39	Innovator												4
41	BoldExpress Studio					✓	✓	✓		✓			5
42	Blu Age	✓											2
44	FrontierSuite												4
45	PowerRAD	✓											2
46	PathMATE												3
47	Agora Plastic 2005	✓	✓										5
48	Framework												2
49	Select Component Factory or Select		✓			✓				✓			6
50	MetaBoss												0

51	Generative Model Transformer												0
52	Objecteering		✓										4
53	OlivaNova Model Execution System	✓								✓		✓	7
54	Enterprise Architect		✓		✓	✓							7
55	MasterCraft												4
56	TAU Generation2												3
57	ACE							✓		✓			5
46		20	13	3	2	8	3	7	1	13	1	1	

Table 13: Technology (programming)

The most supported technology is Java, which includes a range of features such as J2EE which is the most often used platform, which includes EJB and others. .NET and C++ are the second most often mentioned technologies. The results in Table 13 conform to general industry trends about the popularity and usage of languages and platforms so it is not surprising these technologies are popular with the MDA tools.

What is interesting is the relatively high number of tools that support very few languages (3 or less). Given that one of the most often cited advantages of MDA is that you can build the models and then transform them into multiple languages, this low number constrains this advantage.

Technology - general

		CORBA	***Jboss***	***Eclipse***	***Spring***	***Hibernate***	***SQL-DDL***	***ADO/ADO+***	***Data Access Objects (DAO)***	***MVC***	***Pattern-based code generation***	***template based code generation***	***State Machine generation***	***Mainframe***	***Palm OS***	***Linux***	***Windows***	***win32***	***VxWorks***	***COM+***	***Component based development***	
1	Adaptations																					0
2	Ameos	✓																			✓	2
3	Real Time Studio										✓	✓	✓					✓	✓			5
5	smartGenerator	✓					✓					✓		✓	✓							5
6	Together 2006	✓									✓	✓										3
7	Caboom							✓			✓									✓		3
8	SIMplicity										✓	✓	✓								✓	4
9	Codagen Architect										✓	✓										3
10	Codeless																					0
12	REP ++ Studio	✓																		✓		2
13	OptimalJ		✓						✓		✓											3
14	Component-X	✓																				1
16	CodeGenie				✓	✓																2
17	Constructor/MDRAD																					0
19	Bridge																					0
20	e-GEN	✓									✓											3
24	Rational Software Architect			✓							✓											2
25	Medini product family(was m2c)	✓									✓											2
26	Rhapsody	✓		✓																✓		3
27	iQgen	✓		✓								✓										3

28	ArcStyler		✓								✓											2
29	Kabira Transaction Platform and																					0
30	CASSANDRA/xUML																					0
31	iUML/iCCG	✓									✓											2
32	Xcoder						✓													✓		2
35	BridgePoint/xtUML or EDGE UML Suite		✓	✓																		2
36	MetaMatrix Data Services Platform			✓																		1
37	modelscope																					0
38	Model-In-Action			✓			✓				✓	✓										4
39	Innovator	✓					✓					✓										3
41	BoldExpress Studio																					0
42	Blu Age										✓											1
44	FrontierSuite										✓										✓	2
45	PowerRAD																					0
46	PathMATE			✓							✓	✓									✓	4
47	Agora Plastic 2005										✓											1
48	Framework					✓			✓		✓	✓										4
49	Select Component Factory or Select										✓											1
50	MetaBoss										✓											1
51	Generative Model Transformer																					0
52	Objecteering	✓		✓			✓														✓	4
53	OlivaNova Model Execution System									✓										✓		2
54	Enterprise Architect	✓		✓			✓				✓	✓										5
55	MasterCraft																				✓	1
56	TAU Generation2			✓																	✓	2
57	ACE	✓								✓	✓	✓										4
		14	3	10	1	2	6	1	2	2	21	12	2	1	1	0	0	1	1	5	7	

Table 14: Technology general

CORBA is the most often quoted middleware standard, this is not surprising as it is also an OMG standard (see Table 14. Also the use of patterns and templates for code development are often used in MDA tools, which reflects the second and third levels of transformations as discussed earlier.

Coding functionality									
		code: quality	***code: configurable***	***code: debugger***	***code: regeneration***	***code: customisable***	***iterative development***	***Reusable Component Library***	
1	Adaptations								0
2	Ameos								0
3	Real Time Studio								0
5	smartGenerator								0

6	Together 2006								0
7	Caboom								0
8	SIMplicity				✓			✓	2
9	Codagen Architect				✓				1
10	Codeless								0
12	REP ++ Studio								1
13	OptimalJ							✓	1
14	Component-X								0
16	CodeGenie							✓	1
17	Constructor/MDRAD								0
19	Bridge							✓	1
20	e-GEN				✓			✓	3
24	Rational Software Architect	✓							1
25	Medini product family(was m2c)				✓				1
26	Rhapsody								0
27	iQgen								0
28	ArcStyler	✓							1
29	Kabira Transaction Platform and								0
30	CASSANDRA/xUML								0
31	iUML/iCCG		✓						1
32	Xcoder								0
35	BridgePoint/xtUML or EDGE UML Suite			✓					1
36	MetaMatrix Data Services Platform							✓	1
37	modelscope								0
38	Model-In-Action	✓			✓				2
39	Innovator								0
41	BoldExpress Studio								0
42	Blu Age								0
44	FrontierSuite		✓			✓			2
45	PowerRAD								0
46	PathMATE		✓			✓		✓	3
47	Agora Plastic 2005		✓						1
48	Framework		✓					✓	2
49	Select Component Factory or Select							✓	1
50	MetaBoss					✓	✓		2
51	Generative Model Transformer								0
52	Objecteering		✓						1
53	OlivaNova Model Execution System								0
54	Enterprise Architect								1
55	MasterCraft		✓					✓	2
56	TAU Generation2								0
57	ACE								0
		3	7	1	5	3	1	10	

Table 15: Coding functionality

Of the code features there is little functionality mentioned (see Table 15), possibly because this is too low a level for many of tool descriptions to consider, but also because the main focus of MDA is on the modelling aspects. The most often cited coding feature is a reusable component library.

2.3.2.6 Database Supported

Database Technology														
		MS SQL Server	***Oracle***	***DB2***	***Sybase***	***MySQL***	***MS Access***	***Informix***	***Cloudscape***	***PointBase***	***JDBC compliant RDBMS***	***Object-Based database***	***Object-relational mapping***	
1	Adaptations													0
2	Ameos													0
3	Real Time Studio											✓		1
5	smartGenerator													0
6	Together 2006	✓	✓	✓	✓									4
7	Caboom	✓	✓	✓	✓								✓	5
8	SIMplicity													0
9	Codagen Architect													0
10	Codeless													0
12	REP ++ Studio		✓											1
13	OptimalJ													0
14	Component-X													0
16	CodeGenie		✓	✓		✓								3
17	Constructor/MDRAD	✓												1
19	Bridge													0
20	e-GEN													0
24	Rational Software Architect			✓										1
25	Medini product family(was m2c)													0
26	Rhapsody													0
27	iQgen													0
28	ArcStyler	✓	✓											2
29	Kabira Transaction Platform and													0
30	CASSANDRA/xUML													0
31	iUML/iCCG													0
32	Xcoder													0
35	BridgePoint/xtUML or EDGE UML Suite													0
36	MetaMatrix Data Services Platform	✓	✓	✓	✓	✓								5
37	modelscope													0
38	Model-In-Action													0
39	Innovator	✓	✓	✓				✓						4
41	BoldExpress Studio													0
42	Blu Age													0
44	FrontierSuite	✓	✓	✓	✓				✓	✓	✓			7
45	PowerRAD													0
46	PathMATE													0
47	Agora Plastic 2005													0

48	Framework													0
49	Select Component Factory or Select													0
50	MetaBoss													0
51	Generative Model Transformer													0
52	Objecteering													0
53	OlivaNova Model Execution System	✓	✓			✓								3
54	Enterprise Architect	✓	✓				✓							3
55	MasterCraft		✓											1
56	TAU Generation2													0
57	ACE		✓											1
		9	12	7	4	3	1	1	1	1	1	1	1	

Table 16: Database support

The results in Table 16 show that database support does not feature too highly in MDA tool requirements and functionality literature, again this could be to do with the level of detail data sheets and web sites work at. Most developed applications must have some form of persistent storage, thus the majority probably use a database. Some tools such as MetaMatrix [59] refer to being database focused ; it *“provides uniform access to disparate, heterogeneous data sources.”*, therefore they cite a number of potential databases.

2.3.2.7 Tool features

OMG’s Tool Taxonomy												
		Modelling	*Analysis*	*Transformation*	*Composition*	*Test*	*Simulation*	*Metadata Management*	*Reverse Engineering (Legacy)*	*Requirements*	*Version Control*	
1	Adaptations	✓		✓		✓		✓			✓	5
2	Ameos	✓		✓				✓			✓	4
3	Real Time Studio	✓						✓	✓			3
5	smartGenerator	✓				✓						2
6	Together 2006	✓		✓					✓		✓	4
7	Caboom	✓										1
8	SIMplicity	✓		✓								2
9	Codagen Architect	✓		✓					✓			3
10	Codeless	✓									✓	2
12	REP ++ Studio					✓		✓				2
13	OptimalJ	✓		✓						✓		3
14	Component-X	✓				✓	✓					3
16	CodeGenie			✓				✓			✓	3
17	Constructor/MDRAD			✓					✓			2
19	Bridge	✓	✓			✓						3
20	e-GEN	✓				✓						2
24	Rational Software Architect	✓		✓		✓			✓		✓	5
25	Medini product family(was m2c)	✓	✓	✓			✓		✓		✓	6
26	Rhapsody	✓				✓	✓		✓	✓		5
27	iQgen	✓		✓								2
28	ArcStyler	✓	✓			✓			✓	✓	✓	6
29	Kabira Transaction Platform and	✓				✓						2
30	CASSANDRA/xUML					✓	✓					2
31	iUML/iCCG	✓		✓			✓					3
32	Xcoder	✓		✓								2
35	BridgePoint/xtUML or EDGE UML Suite	✓		✓			✓	✓				4
36	MetaMatrix Data Services Platform	✓		✓				✓				3
37	modelscope	✓		✓								2
38	Model-In-Action	✓		✓				✓	✓			4
39	Innovator	✓		✓				✓	✓	✓	✓	6
41	BoldExpress Studio	✓		✓								2
42	Blu Age	✓		✓								2
44	FrontierSuite	✓		✓					✓			3
45	PowerRAD	✓	✓	✓						✓		4
46	PathMATE	✓		✓		✓		✓				4
47	Agora Plastic 2005	✓		✓		✓			✓		✓	5
48	Framework	✓		✓		✓		✓				4

49	Select Component Factory or Select	✓		✓		✓			✓		✓	5
50	MetaBoss	✓				✓						2
51	Generative Model Transformer											0
52	Objecteering	✓	✓	✓						✓	✓	5
53	OlivaNova Model Execution System	✓		✓								2
54	Enterprise Architect	✓	✓	✓		✓			✓		✓	6
55	MasterCraft	✓		✓		✓		✓				4
56	TAU Generation2	✓	✓			✓	✓		✓	✓		6
57	ACE	✓		✓								2
		41	7	31	0	19	7	12	15	7	13	

Table 17: OMG's tool taxonomy

Unsurprisingly as can be seen in Table 20 the majority of tools label themselves as having modelling functionality, which is not surprising given the focus on UML and models in MDA. What is more surprising is that 5 don't list modelling as part of their functionality, maybe because they take it as a given assumption. Also with the focus in MDA on the transformation of models, many data sheets list transformations as part of the tool functionality. The other elements of the OMG's taxonomy are sometimes mentioned, but not in very high numbers. The most interesting feature is the Composition element – which no tool claimed.

Other tool capabilities														
		Software Design	***Configuration Management***	***Deploy Applications***	***Scalable***	***Multi-user***	***Document generation***	***Change management***	***Regenerate code***	***user interface***	***round-trip-engineering***	***Traceability***	***open integration***	
1	Adaptations	✓			✓	✓		✓						4
2	Ameos				✓	✓								2
3	Real Time Studio		✓			✓							✓	3
5	smartGenerator	✓												1
6	Together 2006						✓							1
7	Caboom	✓				✓		✓						3
8	SIMplicity								✓					1
9	Codagen Architect						✓		✓				✓	3
10	Codeless						✓			✓				2
12	REP ++ Studio					✓	✓			✓			✓	4
13	OptimalJ									✓			✓	2
14	Component-X									✓				1
16	CodeGenie													0
17	Constructor/MDRAD									✓				1
19	Bridge			✓			✓							2
20	e-GEN	✓				✓			✓					3
24	Rational Software Architect					✓	✓			✓		✓		4
25	Medini product family(was m2c)								✓			✓		2
26	Rhapsody					✓	✓					✓		4

27	iQgen													1
28	ArcStyler			✓		✓	✓					✓		5
29	Kabira Transaction Platform and			✓										2
30	CASSANDRA/xUML									✓				1
31	iUML/iCCG													0
32	Xcoder										✓			1
35	BridgePoint/xtUML or EDGE UML Suite													0
36	MetaMatrix Data Services Platform				✓	✓		✓						5
37	modelscope									✓				2
38	Model-In-Action			✓			✓		✓	✓				5
39	Innovator					✓	✓				✓			4
41	BoldExpress Studio			✓						✓				3
42	Blu Age			✓						✓				3
44	FrontierSuite			✓	✓							✓		4
45	PowerRAD				✓		✓	✓		✓				4
46	PathMATE													0
47	Agora Plastic 2005					✓								2
48	Framework						✓				✓			3
49	Select Component Factory or Select					✓					✓	✓		4
50	MetaBoss													0
51	Generative Model Transformer													0
52	Objecteering		✓			✓	✓	✓				✓		6
53	OlivaNova Model Execution System						✓			✓				2
54	Enterprise Architect				✓	✓	✓					✓		5
55	MasterCraft					✓	✓			✓				4
56	TAU Generation2		✓			✓	✓	✓		✓		✓		6
57	ACE													1
		4	3	7	6	17	17	6	5	15	4	9	4	

Table 18: Other tool capabilities

A number of other general tool features were explored (see Table 18), and none of these were reported in high numbers. Multi-user tool support and document generation were the most popular at 17. Although as far as document generation support goes it is surprising that more tools do not report supporting this feature, as it seems so essential to the design and development process.

CIM | **PIM** | **PSM**	Users: **All**	**MUST**
GUI REQ ID 14	Document generation	
Related requirement IDs: 12, 14, 26, 27, 28, 29, 32, 33, 58, 59, 62, 63, 66, 75, 76		

Also the reported support for traceability is very low, it is difficult to see how an MDA tool can be fully utilised and support development if requirements cannot be traced both backwards and forwards through the levels of abstraction.

CIM | PIM | PSM	Users: **All**	**MUST**
GUI REQ ID 15	Traceability of requirements through all models and levels of abstraction.	
Related requirement IDs: 8, 9, 10, 15, 34a, 49, 50, 51, 58, 67, 72, 73, 74, 79, 80, 81, 83		

2.3.2.8 General Properties

General						
		interoperable	***Legacy***	***Trial Software available***	***Status of the company***	
1	Adaptations			✓	✓	2
2	Ameos			✓		1
3	Real Time Studio	✓		✓	✓	3
5	smartGenerator					0
6	Together 2006			✓	✓	2
7	Caboom					0
8	SIMplicity			✓		1
9	Codagen Architect					0
10	Codeless			✓		1
12	REP ++ Studio			✓		1
13	OptimalJ	✓		✓	✓	3
14	Component-X	✓	✓		✓	3
16	CodeGenie			✓	✓	2
17	Constructor/MDRAD					0
19	Bridge			✓		1
20	e-GEN					0
24	Rational Software Architect			✓	✓	2
25	Medini product family(was m2c)					0
26	Rhapsody			✓		1
27	iQgen			✓		1
28	ArcStyler		✓	✓	✓	3
29	Kabira Transaction Platform and					0
30	CASSANDRA/xUML			✓		1
31	iUML/iCCG				✓	1
32	Xcoder			✓		1
35	BridgePoint/xtUML or EDGE UML Suite				✓	1
36	MetaMatrix Data Services Platform		✓			1
37	modelscope			✓		1
38	Model-In-Action		✓			1
39	Innovator				✓	1
41	BoldExpress Studio			✓		1
42	Blu Age				✓	1

44	FrontierSuite					0
45	PowerRAD				✓	1
46	PathMATE				✓	1
47	Agora Plastic 2005					0
48	Framework			✓		1
49	Select Component Factory or Select			✓	✓	2
50	MetaBoss					0
51	Generative Model Transformer					0
52	Objecteering			✓	✓	2
53	OlivaNova Model Execution System			✓		1
54	Enterprise Architect			✓	✓	2
55	MasterCraft					0
56	TAU Generation2			✓		1
57	ACE					0
		3	4	24	17	

Table 19: General properties

From the research it is interesting to note how few tools mention legacy systems and integrating them into new developments in their data sheets. Many of the tools have 30 or 15 day trial software that can be downloaded and some have online tutorials and demos .As can be seen in Table 19 seventeen of the tool vendors have some kind of link with the OMG, such as they are members that can be defined as contributing, domain, platform or influencing. This may well have implications for some of the results found, for instance a large proportion of the tools that support a high number of MDA models, support the highest number of UML models and the most MDA standards are developed by OMG members.

2.4 Summary

This Chapter explored the research into MDA tools with the aim of producing a list of features that an MDA tool should contain. These features were then applied to the OMG listed MDA tools and the results published. A number of requirements were outlined for the VIDE project additional to those discussed in D1.

3. Methods of Evaluation

3.1 Introduction

Evaluation is reported in [71] as being goal oriented, with the primary goal of checking results of actions or interventions in order to improve the quality of those actions. Or, it could be to choose the best alternative action, which would be dependent on the current knowledge of science and standards. One of the most important aspects of the work in Work Package 5 is identifying the requirements for the VIDE Graphical User Interface and evaluation of existing Graphical User Interfaces and various other modelling environments. These interface evaluations, in addition to those of the Work Package prototype will inform the requirements for the specification of the final system. There are a number of different views of the way to evaluate a User Interface and these will be explored in this chapter and a method will be selected which will give the greatest value to the package design team.

The evaluation of a software application may be focussed in a number of different areas, for example system performance, usability, re-use, security, personalisation attributes, accessibility etc. However, the issue at the User Interface is likely to be concerned with the way the user interacts with the system and the value that they get from that interaction.

This chapter will first, in Section 3.2, summarise the evaluation standards that are available that impinge on the general area. It will then investigate a number of evaluation methods in Section 3.3. An evaluation method determines the frame of the evaluation by choosing techniques appropriate for the life cycle and setting the focus of the study and the measurement criteria. [71]. The Cognitive Dimensions Framework will be explored in Section 3.4 and finally Section 3.5 discusses the selection of an evaluation paradigm for the Work Package.

3.2 Evaluation standards

Standards are concerned with prescribing the way an activity is done to achieve consistency across a group of products [72]. There are a number of standards relating to the users interaction with a computer, usability and the ease of use with computers, starting with ISO 9241 which is currently being updated.

3.2.1 ISO 9241

ISO 9241 was originally produced in 1999 and entitled 'Ergonomic requirements for office work with visual display terminals'. A number of the sections were found to be outdated and have been or are being remodelled to take into account the move of technology toward the web and mobile services. The new standard title is 'Ergonomics of Human System Interaction' and a number of the sections are being renumbered and retitled. The original Section 11 relates to Usability and defines it
'*as the extent to which a computer system enables users, in a given context of use, to achieve specified goals effectively and efficiently while promoting feelings of satisfaction'.*
ISO 9241 has had a number of criticisms levelled at it because there are 82 pages of guidelines in a number of different sections and designers need to be able to understand not only the goals and benefits of each guideline, but also when the guideline should be applied. In addition there is not always clear information on how to apply each guideline.[73]

3.2.2 ISO/IEC 9126-1 (2000)

Other standards that have some bearing in the area include ISO/IEC FDIS 9126-1 entitled Software Engineering Product Quality. Part one defines a quality model and has a slightly different definition of usability
'*the capability of the software product to be understood, learned, used and attractive to the user, when used under specified conditions'.*
The model reflects a quality model that defines three views, the internal view which is measured by the static properties of the code and is done by inspection. The external view which is measured by the dynamic properties of the code when it is executed and the third view relates to quality in use, which is measured by the extent to which the software meets the needs of the user in the working environment (such as productivity). Quality in use is defined as 'the capability of the software product to enable specified users to achieve specified goals with effectiveness, productivity, safety and satisfaction in specified contexts of use'.[74]
ISO 9126 states that a product has six categories of software quality that are applicable to software development:

- functionality: the capability of the software to provide functions which meet stated and implied needs when the software is used under specified conditions.
- reliability: the capability of the software to maintain its level of performance when used under specified conditions.
- usability: the capability of the software to be understood, learned, used and liked by the user, when used under specified conditions.
- efficiency: the capability of the software to provide the required performance, relative to the amount of resources used, under stated conditions.
- maintainability: the capability of the software to be modified. Modifications may include corrections, improvements or adaptation of the software to changes in environment, and in requirements and functional specifications.
- portability: the capability of software to be transferred from one environment to another. [75]

Parts two and three provide metrics for evaluation of these criteria.

3.2.3 ISO 14915/IEC 61997

This standard contains details of the expected behaviour and appearance of multimedia user interfaces. It provides guidelines for multimedia control and navigation and media selection and combination, including a section on domain specific multimedia interfaces such as kiosk applications and computer aided co-operative work.

3.2.4 ISO/IEC 11581

This standard contains detail relating to icons. The first part relates to the design and development of an icon set and gives generic information relating to icon design. The subsequent five parts relate to specific types of icons – object, pointer, control, tool and action.

3.2.5 ISO/IEC 10741

This standard relates to how the cursor should move across the screen in response to the use of cursor control keys.

3.2.6 ISO/IEC 15910

This standard provides a guide to the process for development of both paper and online user documentation and is particularly for software that has a user interface, including printed documentation (e.g. user manuals and quick-reference cards), on-line documentation, help text and on-line documentation systems.

3.2.7 ISO/IEC WD 18019 (2000)

This standard describes how to establish what information users need, how to determine the way in which that information should be presented to the users, and how then to prepare the information and make it available. It covers both on-line and printed documentation .[74]

3.2.8 ISO 25062 or Common Industry Format (CIF)

This ISO describes the Common Industry Format for Usability Test Reports and is a standard for reporting Usability test findings. It is specifically targeted at usability professionals and stakeholders in organisations that are responsible for software procurement. It is intended to show that good practice in usability evaluation has been followed, there is sufficient information for a usability expert to judge the results, the results may be replicated and it assumes that effectiveness and efficiency metrics are used as well as performance and satisfaction. It is based on the definitions of usability found in ISO 9241:11 as detailed in 3.2.1.The CIF is primarily designed for summative testing rather than formative testing, and is thus considered useful for comparative testing where formal usability testing results in quantitative measurements.

3.3 Usability evaluation methods

3.3.1 Introduction

Neilsen [76] considers usability as part of system acceptability and believes it relates to how well users can use the functionality or utility of a system. System acceptability is determined by both social and practical acceptability. Usability however, is not a one dimensional problem and is associated with five attributes:

- Learnability – a system should be easy to learn so that users can quickly start to use it.
- Efficiency – the system should support a high level of productivity
- Memorability – if a user returns to the system after a period away from it, it should be intuitive and should not have to be re-learned
- Errors – the system should be as error free as possible, make it possible to recover form errors and prevent errors from occurring.
- Satisfaction – users should find the system subjectively pleasant to use

This is illustrated in Figure 2.

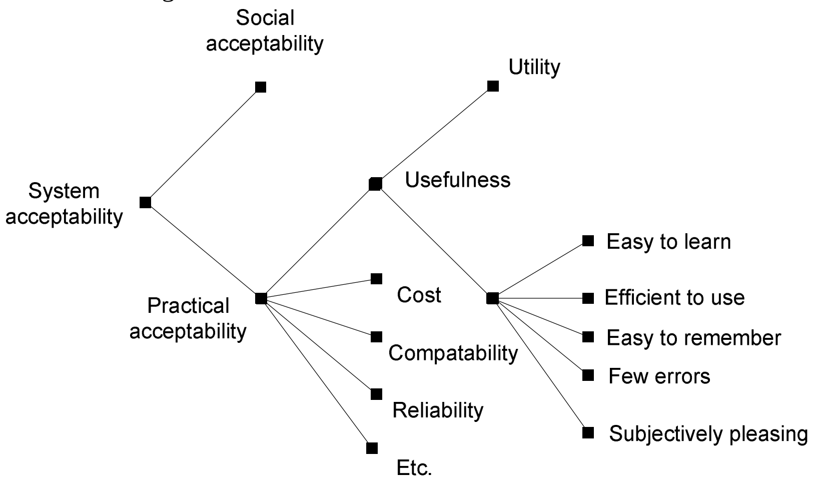


Figure 2: System accessibility and usability (adapted from [76] and [72])

Ivory and Hearst [77] classify methods in terms of their class, type, automation and effort level where the method class describes the evaluation at a high level for example usability or simulation. The method type describes how the evaluation is carried out within the method class such as question-asking protocol which is part of the usability testing class, or Petri-net modelling which is part of the simulation class. Their class and type taxonomy is detailed in Table 20. Their survey relating to the automation of usability evaluation of user interfaces surveyed 132 different evaluation methods, and interested readers are referred to their work.

Method class	**Method type**	**Description**
Testing	Thinking aloud protocol	User provides commentary during test
	Question-asking protocol	Tester asks user questions
	Shadowing method	Expert explains user actions to tester
	Coaching method	User can ask an expert questions
	Teaching method	Expert user teaches novice user
	Co-discovery Learning	Two users collaborate
	Performance measurement	Tester records usage data during test
	Log file analysis	Tester analyses usage data
	Retrospective testing	Tester reviews video with user
	Remote testing	Tester and user are not co-located during testing
Inspection		
	Guideline review	Expert checks guideline conformance
	Cognitive walkthrough	Expert simulates user’s problem solving
	Pluralistic walkthrough	Multiple people conduct cognitive walkthrough
	Heuristic evaluation	Expert identifies heuristic violation
	Perspective-based Inspection	Expert conducts narrowly focussed heuristic evaluation
	Feature Inspection	Expert evaluates product features
	Formal Usability Inspection	Expert conducts formal heuristic evaluation
	Consistency Inspection	Experts checks consistency across products
	Standards Inspection	Expert checks for standard compliance
Inquiry		
	Contextual Inquiry	Interviewer questions users in their environment
	Field Observation	Interviewer observes system use in users environment

	Focus Groups	Multiple users participate in a discussion session
	Interviews	One user participates in a discussion session
	Surveys	Interviewer asks user specific questions
	Questionnaires	User provides answers to specific questions
	Self-reporting logs	User records user-interface operations
	Screen snapshots	User captures user interface screens
	User feedback	User submits comments
Analytical modelling		
	GOMS Analysis	Predict execution and learning time
	UIDE Analysis	Conduct GOMS analysis in a UIDE
	Cognitive Task Analysis	Predict usability problems
	Task-environment Analysis	Assess mapping of user goals into UI tasks
	Knowledge Analysis	Predicts learnability
	Design Analysis	Assess design complexity
	Programmable User Models	Write program that acts like a user
Simulation		
	Information process modelling	Mimic user interaction
	Petri-net modelling	Mimic user interaction from usage data
	Genetic algorithm modelling	Mimic novice user interaction
	Information Scent modelling	Mimic web site navigation

Table 20: Classification and descriptions of user interface evaluation methods from [77]

3.3.2 The Usability Methods

This section will focus solely on some of the more commonly used methods [72] in an effort to identify which would be more suitable for assisting in the development of the VIDE system.

3.3.2.1 Heuristic evaluation

Heuristic evaluation is a method of finding usability problems in an interface design so that the problems can be dealt with as part of an iterative design process. It has been identified as one of the most widely adopted usability evaluation approaches in practice[72]. It is successful because of the advantages it has, which include that it is easy to administer and learn; it has a low cost; it can be applied in the early phases of the development life cycle and does not need professional evaluators [72, 78, 79]. Heuristic evaluation takes place when a group of evaluators investigate the system with a set of accepted guidelines and evaluate whether the system meets those guidelines. There have been a number of guidelines published, for example [78, 80, 81] but the Usability Body of Knowledge has taken many of the heuristic guidelines and created the following guidelines [82]:

- Usefulness
    - Value: The system should provide necessary utilities and address the real needs of users.
    - Relevance: The information and functions provided to the user should be relevant to the user's task and context.
- Consistency
    - Consistency and standards: Follow appropriate standards/conventions for the platform and the suite of products. Within an application (or a suite of applications), make sure that actions, terminology, and commands are used consistently.
    - Real-world conventions: Use commonly understood concepts, terms and metaphors, follow real-world conventions (when appropriate), and present information in a natural and logical order.
- Simplicity
    - Simplicity: Reduce clutter and eliminate any unnecessary or irrelevant elements.
    - Visibility: Keep the most commonly used options for a task visible (and the other options easily accessible).
    - Self-evidency: Design a system to be usable without instruction by the appropriate target user of the system: if appropriate, by a member of the general public or by a user who has the appropriate subject-matter knowledge but no prior experience with the system. Display data in a manner that is clear and obvious to the appropriate user.
- Communication

  - Feedback: Provide appropriate, clear, and timely feedback to the user so that he sees the results of his actions and knows what is going on with the system.
  - Structure: Use organization to reinforce meaning. Put related things together, and keep unrelated things separate.
  - Sequencing: Organize groups of actions with a beginning, middle, and end, so that users know where they are, when they are done, and have the satisfaction of accomplishment.
  - Help and documentation: Ensure that any instructions are concise and focused on supporting the user's task.
- Error Prevention and Handling
  - Forgiveness: Allow reasonable variations in input. Prevent the user from making serious errors whenever possible, and ask for user confirmation before allowing a potentially destructive action.
  - Error recovery: Provide clear, plain-language messages to describe the problem and suggest a solution to help users recover from any errors.
  - Undo and redo: Provide "emergency exits" to allow users to abandon an unwanted action. The ability to reverse actions relieves anxiety and encourages user exploration of unfamiliar options.
- Efficiency
  - Efficacy: (For frequent use) Accommodate a user’s continuous advancement in knowledge and skill. Do not impede efficient use by a skilled, experienced user.
  - Shortcuts: (For frequent use) Allow experienced users to work more quickly by providing abbreviations, function keys, macros, or other accelerators, and allowing customization or tailoring of frequent actions.
  - User control: (For experienced users) Make users the initiators of actions rather than the responders to increase the users’ sense that they are in charge of the system.
- Workload Reduction
  - Supportive automation: Make the user’s work easier, simpler, faster, or more fun. Automate unwanted workload.
  - Reduce memory load: Keep displays brief and simple, consolidate and summarize data, and present new information with meaningful aids to interpretation. Do not require the user to remember information. Allow recognition rather than recall.
  - Free cognitive resources for high-level tasks: Eliminate mental calculations, estimations, comparisons, and unnecessary thinking. Reduce uncertainty.
- Usability Judgment
  - It depends: There will often be tradeoffs involved in design, and the situation, sound judgment, experience should guide how those tradeoffs are weighed.
  - A foolish consistency...: There are times when it makes sense to bend or violate some of the principles or guidelines, but make sure that the violation is intentional and appropriate.

In general, heuristic evaluation is difficult to do as a sole evaluator. Different people find different usability problems; therefore it is possible to improve the effectiveness of heuristic evaluation by using multiple evaluators. Nielsen [78] has conducted considerable research into heuristic evaluation and recommends the use of three to five evaluators. These can be experts, developers or even novices trained to complete evaluation [72]. The process has three stages:

a. the briefing session when the evaluators are told what to do
b. The evaluation period when the evaluator spends 1-2 hours inspects the interface independently using the heuristics as guidelines. They should take two passes, firstly to get a feel of the produce and its’ scope. The second pass allows for specific elements to be investigated and to identify usability problems. If the system is either screen mock-ups or paper prototypes then the method has to be adapted. A second person may record the findings as the evaluator explores the system.
c. The debriefing session when the experts come together to discuss their findings [81].

There are problems with heuristic evaluation however, in that different evaluation approaches will find different problems and sometimes evaluation will miss severe problems [83]. Bailey [84] shows that over a number of studies 36% of identified problems were actually problems, 21% of problems were missed, and 43% of identified problems were not problems at all.

3.3.2.2 Cognitive walkthrough

In a cognitive walkthrough experts examine and discover usability problems of the system by 'walking through' the system and pretending they are the users, particularly novice users [81]. It involves the expert trying to create a users mental model and simulating their problem solving process, and gauging whether the user goals and memory processes in interaction with the system lead to the next step in the sequence [78].
Sharp, Rogers and Preece [81] describe the steps for completing a cognitive walkthrough as:

a. Before commencing the walkthrough, the experts will define the task to be done, the context and their assumptions about the user [72]. A description or prototype of the system is also produced, along with a clear sequence of actions needed for a use to complete a task.
b. The designer and an expert evaluator do the analysis of the system together.
c. The evaluators proceed through the task sequences asking the following questions:
    i. Will the correct action be sufficiently evident to the user?
    ii. Will the user notice that the correct action is available?
    iii. Will the user associate and interpret the response from the action correctly?
d. A record is completed which details the problems found, the assumptions made and the design changes required.
e. The design is revised.

It takes much longer to complete a cognitive walkthrough than a heuristic evaluation.[81]

3.3.2.3 Pluralistic walkthrough

A pluralistic walkthrough is a walkthrough the system that involves users, developers and usability experts stepping through a scenario discussing usability issues that arise[78]. Bias [85] outlines the steps that need to be completed as:

a. Scenarios are developed in the form of paper or screen prototypes.
b. The scenarios are presented to a panel of evaluators and the panellists are asked to write down the sequence of action they would take to move from one screen to another. They do this individually without conferring.
c. The panellists then discuss the actions they suggested. The users usually present first, followed by the usability experts with developer comments last.
d. This process continues until all screens have been evaluated.

The benefits of this type of walkthrough are that they focus on the user tasks, and will provide quantitative data that can be analysed. The limitations are that it can be difficult to get the range of evaluators together and the work can be slow, as the panel have to proceed at the speed of the slowest member. The number of paths that can be analysed are also limited [81].

3.3.2.4 Focus groups

Focus groups are a commonly used method of obtaining user reaction to samples, products and prototypes in marketing. They have been found to be useful in such areas as requirements elicitation, where a focus group can assist developers in ascertaining user requirements in Joint Application Development. They can be used in interaction design by bringing together a group of six to nine users to discuss issues about features of a user interface. [86] As an alternative to questionnaires they are very economical and will generally generate a wide range of issues [87].
The benefits to the VIDE project particularly in the early stages of interface design are that the requirements are also being identified, and any focus group will highlight concerns about early design decisions as well as helping to refine requirements.

3.4 The Cognitive Dimensions Framework

3.4.1 Introduction

Green and Petre [88] believe that the evaluation of programming environments presents a number of challenges and that many of the techniques such as those summarised in Table 20 concentrate on a low level of detail of the interaction between the user and the device. These low level evaluations are not suitable for evaluating programming environments and the accompanying notational design issues. Green [89] stresses the importance of the relationship between notation and the support environment, and the Cognitive Dimensions Framework has been formulated as a discussion tool and as such is more useful to users who are not Human Computer Interaction (HCI) specialists [88]. It is a framework that allows developers to think about the nature of the notational system, the way that people interact with them and a structure in which to understand the vocabulary [90]. Blackwell describes a notational system as consisting of marks made on some medium and using the example of a computer screen there may be multiple notations such as the main notation and the notation in surrounding menu dialogues.

3.4.2 The Cognitive Dimensions

These are described in [88-91] and may be summarised as:

- Viscosity: resistance to change
  A viscous system needs many user actions to accomplish one goal. How many user actions are necessary to make one change?

- Visibility: ability to view components easily
  Systems that have low visibility bury information in encapsulations. Is every part of the notation visible or is it at least possible to juxtapose two parts?

- Premature commitment: constraints on the order of doing things
  Does the user have to make decisions before having the information they need?

- Hidden dependencies: important links between entities are not visible
  Is every dependency clearly indicated in both directions? Is the indication perceptual or symbolic?

- Role-expressiveness: the purpose of an entity is readily inferred
  Can the user see how each of the components relates to the whole?

- Error proneness: the notation invites mistakes and the system gives little precaution

- Abstraction gradient: Types and availability of abstraction mechanisms

- Secondary notation: extra information in means other than formal syntax
  Can users use layout, colour and other cues to convey extra meaning, beyond the semantics of the notation or language?
- Closeness of mapping: closeness of representation to domain.

- Consistency: similar semantics are expressed in similar syntactic forms
  When some of the notation or language has been learned, can the rest be inferred?

- Diffuseness: verbosity of language
  How many symbols or graphic entities are required to express a meaning

- Hard mental operations: high demand on cognitive resources
  Are there places where the user needs pencil and paper to track what is happening?

- Provisionality: degree of commitment to actions or marks

- Progressive evaluation: work to date can be checked at any time
  Can a partially completed operation be executed to obtain feedback?

3.4.3 Cognitive dimensions trade-off

The purpose of the cognitive dimensions framework is to open discussion of the effects on cognition of different design decisions, and as such these will always involve trade-offs and these can be illustrated in Figure 3. Blackwell [91] gives the example of changing the structure of a notation to reduce viscosity is likely to affect other dimensions such as introducing hidden dependencies or increasing the abstraction.

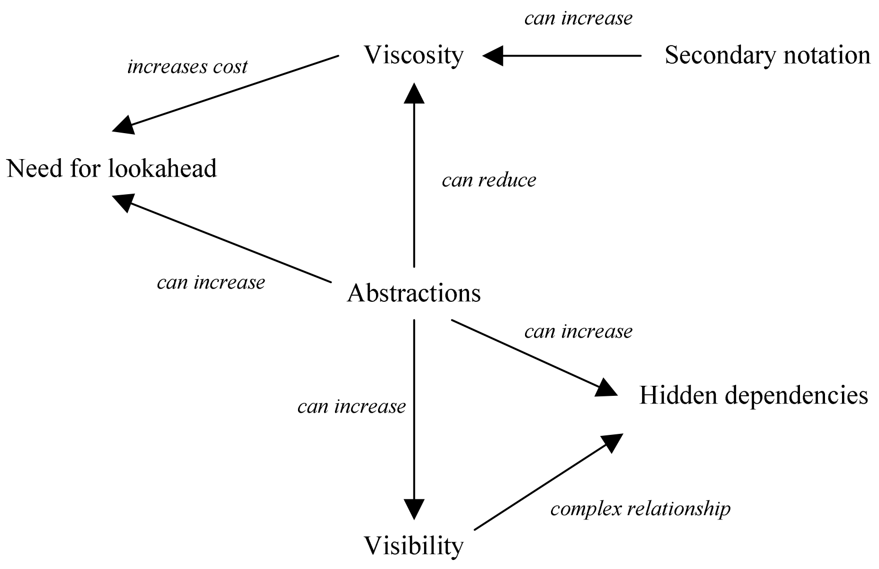


Figure 3: Cognitive Dimensions trade-offs [92]

3.5 Summary

Neilsen comments in [78] that many developers find usability evaluation methods to be intimidating, too difficult and time consuming to use therefore this work needs to define (a) method(s) that is both easy to learn and use and that will give sound results in a short development time scale. Wixom believes that that the growing body of research on evaluating usability methods is both unhelpful and irrelevant to the practitioner [93]. He shows that most of the available literature available to date is based on three points. Firstly that one of the most useful criteria for evaluating a method is the number of problems that can be detected using it. Secondly, that, a method can be evaluated in isolation from the goals and the context of that method. Finally, that using some kind of 'quasi-scientific' framework is the most effective way to resolve issues about selecting the best method [93]. In practice he believes that problem detection is a first step to improving the product, it is not sufficient on its' own for product improvement or as a criteria for method evaluation. The isolation of the method from its context means that important considerations such as development team adoption are lost. Finally, Wixom believes that a scientific framework for evaluating methods is inconsistent with the pragmatic philosophy and the engineering approach to product development.

To this end, this work has discussed a number of standards and approaches but will use Wixoms ideas for the selection of a usability method. He believes that the maxim should be 'how much can we improve the product in the shortest time with the least effort?' [93]. To this we would add '*the quality of* ' to make the maxim 'how much can we improve *the quality of* the product in the shortest time with the least effort? Nielsen advises that a single evaluation method in a project will not produce the best results, and that two different methods should be considered. One to evaluate a first attempt, and using the results of this evaluation a second version can be completed which is likely to be more usable [78].

The decision has therefore been made to use focus groups throughout the initial design stages, and the Cognitive Dimensions Framework to facilitate discussion of the early prototype other useful user interfaces.

4. Software Visualisation, Visual Programming and Diagramming Research

4.1 Introduction

There has been no work on the pre-CIM environment and very little on the CIM environments in relation to MDA in academic literature, therefore the work in this chapter draws heavily from a number of areas and applies it to VIDE. There is a rich stream of research in the area of visual programming and diagramming and their respective environments. We understand visual programming to mean the specification of a computer system using graphics and program visualisation to mean the use of graphics to enhance the understanding of a computer program [94]. In addition there is considerable work in software visualisation which Price defines as 'the use of interactive compute graphics, typography, graphic design, animation, and cinematography to enhance the interface between the software engineer or the computer science student and their programs'[94]. A visualisation is a collection or configuration of representations (and other information) which together make a higher level component [95].

Because data and text appear as visual images they allow the developer to make use of the brains' ability to link between ideas and pictorial representations [94]. A representation is a graphical (and other media) depiction of a single component [95]. Typical tasks which are assisted by software visualisation include design and development activities, debugging, testing, and maintenance [96].Work during early stages of development [97] at the 'pre-CIM' level are particularly usefully served with software visualisation assisting in the capturing of requirements and providing a useful discussion tool for developer collaboration at the user interface. Requirements visualisation has been a research topic for a number of years particularly using scenarios, but it is emerging as a growing area of interest and has resulted in a Requirements Visualisation Workshop series [98]. The VIDE project emphasises the visual aspects of development thus the use of research as the basis for deciding features which can be incorporated into the requirements for the VIDE CIM level design.

This chapter explores the software visualisation, visual programming and natural programming research areas, before finally looking at some of the tools that are currently available in the area.

4.2 Requirements from software visualisation research

Development tasks completed at different stages of the life cycle will require different sets of information to be available [96, 97] and no single visualisation tool as yet has been able to address all software engineering tasks simultaneously [96]. This is because there are two major levels that need to be addressed, the design and the coding. The design requires a full understanding of the problem and the problem domain, including how the developer interprets those problems and addresses them in the solution. It requires representations that are conceptual. At the software level it is more about addressing how the code works, how components interact and what services are available [97]. This means that Work Package 5 needs to take this into account as it is catering for the pre-CIM and CIM level, through design to the PIM level which is ultimately the code level.

In the search for requirements a good starting point is the taxonomies and descriptions that are available in software visualisation. Price [94] lists six areas that need to be considered when describing visualisations and these include scope, content, form, method, interaction and effectiveness. Young and Munro [95] describe a slightly different set of criteria with representation, abstraction, navigation, correlation, automation and interaction. Maletic, Marcus and Collard produce a further task oriented view [96] with tasks (why is the visualisation needed?), audience (who are the users?), target (what are the data sources?), representation (what should it look like?) and medium (how should it look?). They believe that too much is made in the visualisation field of tools that do not match requirements, and believe that more should be done to highlight the strengths of tools instead of weaknesses.

Young and Munro [95] give a list of desirable properties for a software representation and a further list for a software visualisation. It is an ambitious list which is not pragmatic [97] and although discussing three dimensional software is certainly a starting point for VIDE requirements. The work of Young and Munro [95] is therefore used heavily in the sections below:

A number of the properties are mutually exclusive, and therefore a requirement for one attribute is likely to be at the expense of another. A successful system will achieve a balance between the different attributes.

4.2.1.1 Individuality

Representations of different components should appear differently, and identical components displayed in the same context should appear identical.

Level :All	Users: All	**SHOULD**
GUI REQ ID 16	Different components should appear differently	
Related requirement IDs: 1, 2, 3, 16, 17, 18, 19, 20, 21 ,22, 23, 24, 25, 26, 27, 28, 35, 60, 64, 69		

Level: All	Users:	**SHOULD**
GUI REQ ID 17	Identical components in the same context should appear identical	
Related requirement IDs: 1, 2, 3, 16, 17, 18, 19, 20, 21 ,22, 23, 24, 25, 26, 27, 28, 35, 60, 64, 69		

4.2.1.2 Distinctive appearance

Differing representations should appear as different as possible; they should be easily recognisable as being either similar or different.

Level: All	Users: All	**SHOULD**
GUI REQ ID 18	Different components should be distinctive	
Related requirement IDs: 1, 2, 3, 16, 17, 18, 19, 20, 21 ,22, 23, 24, 25, 26, 27, 28, 35, 60, 64, 69		

4.2.1.3 High information content

Representations should provide as much information as possible about the component. This can be an issue because increasing information content is likely to increase visual complexity.

Level: All	Users: All	**SHOULD**
GUI REQ ID 19	Visual representations should provide as much information as possible about the component.	
Related requirement IDs: 1, 2, 3, 16, 17, 18, 19, 20, 21 ,22, 23, 24, 25, 26, 27, 28, 35, 60, 64, 69		

4.2.1.4 Low visual complexity

Representations should be visually simple. This will aid both the information system and the users understanding of the information. There should be as little visual complexity as possible. The trade-off is likely to be between whether the user is expected to distinguish between a large number of simple representations or a smaller number of complex visualisations. Granularity, abstraction, information content and type of information should vary to accommodate users' interest [97].

Level: All	Users: All	**SHOULD**
GUI REQ ID 20	Visual representations should be as simple as possible	
Related requirement IDs: 1, 2, 3, 16, 17, 18, 19, 20, 21 ,22, 23, 24, 25, 26, 27, 28, 35, 60, 64, 69		

4.2.1.5 Scalability of information content and visual complexity

The amount of information that is visible, or its complexity, should depend upon the context in which it is used. This will allow detailed displays of a small number of components giving maximum information as well as overviews of large numbers of components displaying less information about the individual components.

Level: All	Users: All	**SHOULD**
GUI REQ ID 21	The amount of information that is visible, or its complexity, should depend upon the context in which it is used.	
Related requirement IDs: 1, 2, 3, 16, 17, 18, 19, 20, 21 ,22, 23, 24, 25, 26, 27, 28, 35, 60, 64, 69		

4.2.1.6 Flexibility for integration into visualisations

Representations have both intrinsic and extrinsic dimensions which can be used for encoding information [99]. Extrinsic dimensions refer to the position or location of an object and intrinsic dimensions refer to the nature of the object such as colour, size and shape. Whilst Young and Munro use this attribute in particular to discuss problems in 3 dimensions, some parts are still valid in the VIDE project. They discuss the fact that any representation that uses more intrinsic dimensions becomes less flexible for integration into a visualisation [95]. Any use of colour in differentiating between representations or groups or classes of components can cause conflict because different components may use the same colour to represent different information. Size is also a factor, in that irregular sized components will be more complex to place and view.

Level: All	Users: All	**SHOULD**
GUI REQ ID 22	Use colour sparingly and consider extrinsic and extrinsic dimensions	
Related requirement IDs: 1, 2, 3, 16, 17, 18, 19, 20, 21 ,22, 23, 24, 25, 26, 27, 28, 35, 60, 64, 69		

4.2.1.7 Suitability for automation

Any representation should have the ability to be automatically generated, possibly with some user intervention. Irregular shapes and variable sizes can complicate the automation process.

Level: All	Users: All	**SHOULD**
GUI REQ ID 23	Use regular shapes and consider extrinsic and extrinsic dimensions in addition to facilitating automation	
Related requirement IDs: 1, 2, 3, 16, 17, 18, 19, 20, 21 ,22, 23, 24, 25, 26, 27, 28, 35, 60, 64, 69		

4.2.1.8 Simple navigation with minimum distortion

Any visualisation should be clearly structured and have features that assist in the navigation through the system.

Level: All	Users: All	**SHOULD**
GUI REQ ID 24	The components should be structured and have features to aid navigation	
Related requirement IDs: 1, 2, 3, 16, 17, 18, 19, 20, 21 ,22, 23, 24, 25, 26, 27, 28, 35, 60, 64, 69		

4.2.1.9 Resilience to change

Small additions or changes to the information in response to changes in users' interests should not result in major changes to the visualisation.

Level: All	Users: All	**SHOULD**
GUI REQ ID 25	The system should allow minor additions or changes which will not cause major changes to the overall system.	
Related requirement IDs: 1, 2, 3, 16, 17, 18, 19, 20, 21 ,22, 23, 24, 25, 26, 27, 28, 35, 60, 64, 69		

4.2.1.10 Good use of visual metaphor

Metaphors are useful in that they help the user understand a situation using familiar concepts.

Level: All	Users: All	**SHOULD**
GUI REQ ID 26	The system should use metaphor where possible	
Related requirement IDs: 1, 2, 3, 16, 17, 18, 19, 20, 21 ,22, 23, 24, 25, 26, 27, 28, 35, 60, 64, 69		

4.2.1.11 Approachable user interface

The user interface should encourage intuitive navigation and should not create unnecessary overhead.

Level: All	Users: All	**SHOULD**
GUI REQ ID 27	The system should be intuitive with regard to navigation and control.	
Related requirement IDs: 1, 2, 3, 16, 17, 18, 19, 20, 21 ,22, 23, 24, 25, 26, 27, 28, 35, 60, 64, 69		

4.2.1.12 Integration with other information sources

Because representations are usually an abstraction of the original information that they represent, it should be possible for the user to return to the original information source and to link to other views on the same information.

Level: All	Users: All	**SHOULD**
GUI REQ ID 28	The user should be able to return to the original information that the system is representing.	
Related requirement IDs: 1, 2, 3, 16, 17, 18, 19, 20, 21 ,22, 23, 24, 25, 26, 27, 28, 35, 60, 64, 69		

Level: All	Users: All	**SHOULD**
GUI REQ ID 29	The user should be able to link to other views of the same piece of information.	
Related requirement IDs: 12, 14, 26, 27, 28, 29, 32, 33, 58, 59, 62, 63, 66, 75, 76		

4.2.1.13 Good use of interaction

Visualisations will be improved by allowing user input to gain more information and to retain their attention.

Level: All	Users: All	**SHOULD**
GUI REQ ID 30	The system should allow the user to record input and additional information.	
Related requirement IDs: 12, 21, 24, 25, 27, 28, 29, 30, 31, 35, 51, 58, 59, 60, 64, 65, 66		

4.3 Requirements from visual programming research

There is considerable research into visual programming much of it discussing benefits or otherwise of following a visual programming paradigm. However, there has been little empirical work that justifies the various design decisions that are contained within either visual programming or visualisation systems [100]. Pane and Myers [101] believe that many visual programming systems are designed with technical objectives and have very little consideration of usability issues whilst Petre argues that there is an acceptance amongst authors that ‘graphics are better simply because they are graphical’[102]. Petre focuses on a number of issues that need to be considered in a graphical programming environment, but believes they generalise to other areas such as user interfaces and environment [102]. The main point she raises is that good graphics rely on secondary notation, and that the use of secondary notation is what makes a major difference between novices and experts. Secondary notation means that graphical representations complement other notation. An example of this is layout cues in programming code. The layout gives a graphical representation and represents information that may well be less accessible than the original symbolic notation. Petre’s findings show that graphical features do not guarantee that a representation will be clear. It is the use of the secondary notation which gives the clarity and poor use of secondary notation is what distinguishes novices from experts [102].

The continued interest in graphical representation is due to the results of a survey reported in [102] when programmers of all levels showed their preference for graphical representation because they are richer and provide more information, provide an overview and making structure more visible, have a higher level of abstraction, are more accessible, comprehensible, memorable and ‘fun’.

Level: All	Users: All	**SHOULD**
GUI REQ ID 31	Consider using secondary notation to improve comprehension	
Related requirement IDs: 12, 21, 24, 25, 27, 28, 29, 30, 31, 35, 51, 58, 59, 60, 64, 65, 66		

Work involving the design of a visual diagramming and programming environment will ask a number of questions such as: who is it for? What are they doing [88]. This has already been addressed in Deliverable D1, but the information is built upon here for in greater detail and for completeness it is also built upon to discuss the way that expert and novice programmers visualise and conceptualise.

4.3.1 Analysis of user characteristics

Computer systems should be built to suit the needs of people; consequently it is important to discover what types of people will be using the interface [103]. A broad characterisation of the general qualities of typical VIDE stakeholders, working at the levels of CIM, PIM and PSM is presented in Table 21. They are described in terms of the levels of system engagement specified by [104] (i.e. *mandatory* and *discretionary* types and *direct*; *indirect*; *remote*; *support* contact) and technical expertise (*novice*; *intermediate* and *expert*).

	CIM	**PIM**	**PSM**
Engagement	*Discretionary indirect/remote*	*Mandatory direct/indirect*	*Mandatory direct*
Examples	Business end-users, business consultants or business analysts,	System analysts/architects, usability engineers, VIDE Programmers	Software developer and software engineers
Expertise	*Novice*	*Novice/Intermediate*	*Expert*
Characteristics	Low levels of experience with computers; but high business domain expertise that is only easily expressed in terms of business goals rather than system requirements.	Low to medium technical experience but have expertise in mapping business or user requirements to high-level, system oriented models.	High levels of technical expertise but with typically no *direct* experience of (or access to) the problem domain.

Table 21: VIDE stakeholder expertise from [105]

Novice system users require robust systems that clearly represent their work domain, using visual cues and help to allow them to successfully perform their activities through interaction with the computer [104]. Indeed, VIDE stakeholders contributing at the CIM level are unlikely to be able to express their needs in a technical sense and may not even directly engage with the tool but instead enlist the help of a proxy (perhaps a PIM stakeholder). Studies of novice performance show that their strategies are both 'chaotic' and 'rigid' [102]. They tend to lack reading and search strategies of graphics and thus graphics can become more difficult to access because they are unsure which are important or relevant. Novices tend to believe that if a graphical representation is present then it must be relevant and will try to make sense of explicit connections, even if it is not relevant[102].

Level: All	Users: Novices	**SHOULD**
GUI REQ ID 32	Keep all graphical representations relevant.	
Related requirement IDs: 12, 14, 26, 27, 28, 29, 32, 33, 58, 59, 62, 63, 66, 75, 76		

Level: All	Users: Novices	**SHOULD**
GUI REQ ID 33	All explicit connections should be relevant	
Related requirement IDs: 12, 14, 26, 27, 28, 29, 32, 33, 58, 59, 62, 63, 66, 75, 76		

Those individuals who engage in PIM-level modelling activities are likely to be heterogeneous both in terms of their engagement with the VIDE tool and also their technical expertise. As 'bridge builders' between the disparate views of CIM and PSM, this group may contain stakeholders who are predisposed to either the CIM or the PSM perspective but are capable of expressing these concepts to some degree at the PIM level. For this reason, users at this level are likely to require support in their transformation activities, in particular direct access to on-line help and a clear view of the broader aspects of the VIDE architecture.

Most studies of expert focus on the strategies they use to solve problems and how they represent their knowledge and [106] provides a good summary. They describe experts as being able to

- Have efficient organisation of knowledge
- Use functional characteristics such as the nature of the underlying algorithm for organising that knowledge
- Use both generalised and specialised problem solving strategies
- Use strategies such as a high level approach to program decomposition and understanding
- Be flexible in their approach to comprehension and demonstrating the ability to adapt and change hypotheses.
- Recognise, use and adapt patterns
- Complete tasks with speed, accuracy and have a supply of examples, sources and strategies for dealing with situations.

Whilst specific to expert programmers, a number of these attributes are common across experts in any field [106].

It is anticipated that PSM contributors are likely to have high levels of technical expertise and harbour expectations of the system that matches their own experience of system software construction. As a consequence, these users will require visual representations of their modelling and development activities to have significant parallels with contemporary notations, such as the UML.

If a programmer completes a set of tasks with a specific set of goals in mind then the success or otherwise of that task depends on the use of representations that are graphical or textual and how useful the information that is displayed, and how close it is to the users information requirements [102]. This is a transferable skill, and essential not just to programmers, but anyone that is aiming to complete a task or solve a problem.

4.4 Requirements from the diagramming environment

The most commonly used PIM modelling notation is the UML class diagram. Given that VIDE aims to target a range of users (e.g., experienced developers, and business people with less experience), it may become necessary to consider some of the issues below when building the modelling component of VIDE. Most of the guidelines focus on the layout of the class diagram, especially, where different tools are used by development teams who share class models. We have not seen however, guidelines regarding how class layouts may be maintained when the class model is derived from say, parts of a CIM model. In other words, some of the graph theory-based algorithms proposed for managing “best-look” class designs do not focus on a typical MDA development process where artefacts are often a result of transformations. This does not mean that of these guidelines are not useful for VIDE. It may be that, some of the guidelines might be applied to CIM modelling and PIM modelling, even to interaction modelling. For example, the concept of minimising the number of bends, or crossing of edges might be considered relevant for business process models, use cases, and even sequence charts. We may need to be able to consider how derived PIMs (e.g., from CIMs) may also be made to adhere to the same guidelines applied to the source model, or vice versa.

4.4.1 UML class diagram layout

The UML is clear on the structure of the class diagram notation (class name, attributes, and operations). The UML, however, does not prescribe how modellers should layout class diagrams in order to enhance readability [107] [108]. It has been argued that the quality of a design can be linked to the readability of the class model since an unreadable model may not convey much to other designers who may be part of the development team, nor the potential users of the system.

Considerations have been made regarding aesthetic criteria for laying out class diagrams, especially, to ensure when class models are shared among different tools, the class model retains the initial layout. For example, some researchers (e.g., [109], [110]) have provided automatic layout mechanisms to be used to assure given aesthetic criteria such as distance between inheriting and inherited classes are maintained. The traditional manual layout of diagrams is not considered optimal since sharing of diagrams across tools often results in different layouts of the same model. Automatic layout algorithms, though well thought out and sensible, may have their drawbacks. Modellers will often lay diagrams in a manner that depicts their conceptualisation of the design they are building. Automatic calculations that result in a different layout might mean that modellers may obtain a different understanding of the model from the initial understanding. One expert may well draw another experts diagram to make it more meaningful to them [102]. We are therefore of the view that, where automatic layout mechanisms are deployed, there should be a choice for the modeller whether or not to adhere to automatic layout functions.

CIM | **PIM** | **PSM**	Users: **End** | **Analyst** | **Designer** | **Arch** | **Programmer**	**SHOULD**
GUI REQ ID 34	It should be possible for users to arrange their models themselves.	
Related requirement IDs: 12,19, 25, 27, 34, 47, 48, 58, 59, 60, 61, 63		

CIM | **PIM** | **PSM**	Users: **End** | **Analyst** | **Designer** | **Arch** | **Programmer**	**MUST**
GUI REQ ID 34a	The graphical layout of VIDE models must be stored persistently.	
Related requirement IDs: 8, 9, 10, 15, 34a, 49, 50, 51, 58, 67, 72, 73, 74, 79, 80, 81, 83		

CIM | **PIM** | **PSM**	Users: **End** | **Analyst** | **Designer** | **Arch** | **Programmer**	**MUST**
GUI REQ ID 35	It must be possible to turn on or off any automatic layout functionality within VIDE.	
Related requirement IDs: 12, 21, 24, 25, 27, 28, 29, 30, 31, 35, 51, 58, 59, 60, 64, 65, 66		

4.4.2 An overview of UML class diagram layout guidelines

Various aesthetic guidelines have been proposed based on knowledge from graph theory ([111] [112]). For example, it is considered good practice to ensure that links among classes (e.g., associations, inheritance, containments) are such that arrows point to one general direction. To enhance visibility of model elements, especially where structuring has been done using packages, spatial considerations should be made to ensure clarity of relations among classes across the packages. Heavy inter-package relations mean tight coupling, hence a bad design. For generalisation and containment associations, parents and containing models should be as near as possible to the child or contained classes. There are various proposed algorithms (e.g., see [113], [111]) for ensuring these distances are maintained across different tools. Where possible, it is suggested that the area occupied by a class should be minimal and that association edges should not be allowed to overlap or cross. Some guidelines appear purely subjective as there is no obvious way to judge the appropriate measure. For example, it is suggested that association edges should not be too short nor too long [114]. Furthermore, where comments are attached to classes or packages, the comment node should be as near as possible to the class or package. Additional information such as association names, and multiplicity indicators, should be clearly assigned to their model elements.

PIM | **PSM**	Users: **End** | **Analyst** | **Designer** | **Arch** | **Programmer**	**SHOULD**
GUI REQ ID 36	The graphical space used to represent a class should be minimal.	
Related requirement IDs: 12, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 31, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 61		

CIM | **PIM** | **PSM**	Users: **End** | **Analyst** | **Designer** | **Arch** | **Programmer**	**SHOULD**
GUI REQ ID 37	Where automatic layout is used, it should keep compositional elements close together.	
Related requirement IDs: 12, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 31, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 61		

PIM | **PSM**	Users: **End** | **Analyst** | **Designer** | **Arch** | **Programmer**	**SHOULD**
GUI REQ ID 38	Where automatic layout is used, it should keep aggregated elements near by.	
Related requirement IDs: 12, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 31, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 61		

PIM | **PSM**	Users: **End** | **Analyst** | **Designer** | **Arch** | **Programmer**	**SHOULD**
GUI REQ ID 39	Where automatic layout is used, relational arcs should not cross.	
Related requirement IDs: 12, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 31, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 61		

PIM | **PSM**	Users: **End** | **Analyst** | **Designer** | **Arch** | **Programmer**	**SHOULD**
GUI REQ ID 40	Where automatic layout is used, bends in arcs should be kept to a minimum.	
Related requirement IDs: 12, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 31, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 61		

The general thrust of the various considerations for class layout is not only based on readability, but also on production of quality design. For example, it is considered a poor design where a class has a high number of outgoing relations, meaning that the class depends on many other classes, hence, high coupling. Furthermore, empty classes are to be discouraged because they add no value to the model.

CIM | **PIM** | **PSM**	Users: **End** | **Analyst** | **Designer** | **Arch** | **Programmer**	**MUST** | **SHOULD** | **COULD** | **WONT**
GUI REQ ID 41	Where it is possible to automatically detect, 'design smells' could be visually highlighted within a model view.	
Related requirement IDs: 12, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 31, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 61		

Similar observations are made by [115] who argue that it is preferable to minimise crossing of association edges in a class design. Other guidelines suggest that where labels exist, they should be horizontal, and inheritance lines should clearly indicate the involved classes [116]. In addition, crossing minimization, bend minimization, orthogonal layout, maintaining a uniform direction of associations (e.g. arcs of a class hierarchy pointing in a consistent direction), and that there shouldn't be any nesting of one class hierarchy within another. Further to these guidelines, [116] adds that shared inheriting edges should be merged prior to reaching the super class. Also, edges should be labelled appropriately, preferably on either end of the class association.

PIM | **PSM**	Users: **End** | **Analyst** | **Designer** | **Arch** | **Programmer**	**SHOULD**
GUI REQ ID 42	Where automatic layout is used, labelled arcs should be kept horizontal.	
Related requirement IDs: 12, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 31, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 61		

PIM | **PSM**	Users: **End** | **Analyst** | **Designer** | **Arch** | **Programmer**	**SHOULD**
GUI REQ ID 43	Where automatic layout is used, an inherited class path should be easy to follow.	
Related requirement IDs: 12, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 31, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 61		

PIM | **PSM**	Users: **End** | **Analyst** | **Designer** | **Arch** | **Programmer**	**SHOULD**
GUI REQ ID 44	Separate class inheritance paths should not overlap.	
Related requirement IDs: 12, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 31, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 61		

PIM | **PSM**	Users: **End** | **Analyst** | **Designer** | **Arch** | **Programmer**	**SHOULD**
GUI REQ ID 45	Inheritance arcs that share the same parent should merge before reaching the parent class representation.	
Related requirement IDs: 12, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 31, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 61		

CIM | **PIM** | **PSM**	Users: **End** | **Analyst** | **Designer** | **Arch** | **Programmer**	**SHOULD**
GUI REQ ID 46	Is should be possible to label entity relationships in both directions.	
Related requirement IDs: 12, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 31, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 61		

It is interesting that most work on the aesthetics of class diagram construction has produced similar guidelines. For example, work by [109] also indicates the need to minimize crossings between edges, minimizing the number of bends, minimizing overlaps between nodes (classes) and edges, maximizing number of orthogonal edges, minimizing edge length, and minimizing area occupied by classes. It is also argued (e.g., see [115]) that modellers should construct associations such as generalisations for a class in the same direction. Others have also suggested that various class hierarchies, including generalisations should he highlighted using different colours.

4.4.3 A note on guideline support issues within MDA tools

A problem with most of these recommendations is that, it is not possible to see how they may be supported by all tool vendors, given that some of the suggestions to implement a common algorithm might not be commercially viable to some of the vendors. That is, it may not matter to some MDA tool providers whether or not these guidelines are adhered to or not. The introduction of algorithms for class layout from Graph theory may also not interest some tool vendors.

4.4.4 Requirements for class layout

The size of class models depends mainly on the size of the problem being modelled. Since class models are an Object Oriented construct, one might want to deploy concepts such as inheritance hierarchies, composition or aggregation to break down complex class models into more readable artefacts. Furthermore, others have argued that association lines should not cross since "messy" association lines curtail the readability of the class diagram. It is notable though that no CASE tool has yet implemented such a guideline, even at the level of notifying the modeller where such associations are crossing. We argue that, the VIDE environment could provide a means for instructing modellers to avoid such issues where they arise.

PIM | **PSM**	Users: **End** | **Analyst** | **Designer** | **Arch** | **Programmer**	**COULD**
GUI REQ ID 47	The VIDE environment could provide on-line guidance or interactive heuristics for class layout.	
Related requirement IDs: 12,19, 25, 27, 34, 47, 48, 58, 59, 60, 61, 63		

PIM | **PSM**	Users: **End** | **Analyst** | **Designer** | **Arch** | **Programmer**	**COULD**
GUI REQ ID 48	The VIDE environment could visually highlight crossing relation arcs.	
Related requirement IDs: 12,19, 25, 27, 34, 47, 48, 58, 59, 60, 61, 63		

Software development (with visual models or traditional source code-based development) is an intellectual activity that often takes an iterative process across many versions of the same product. An important aspect of a modelling IDE would be the provision of a version management function in order for development participants to track changes or the evolution of models. This is important because as new information regarding the problem comes to light, models will need to be changed to reflect learned information. Class models might also be needed purely for aesthetic reasons - to provide neater, more readable artefacts.

PIM	Users: **End** | **Analyst** | **Designer** | **Arch** | **Programmer**	**COULD**
GUI REQ ID 49	The VIDE environment could persistently store early versions of PIM models and their layout.	
Related requirement IDs: 8, 9, 10, 15, 34a, 49, 50, 51, 58, 67, 72, 73, 74, 79, 80, 81, 83		

PIM	Users: **End** | **Analyst** | **Designer** | **Arch** | **Programmer**	**COULD**
GUI REQ ID 50	It could be possible for PIM modellers to interactively navigate through PIM model history.	
Related requirement IDs: 8, 9, 10, 15, 34a, 49, 50, 51, 58, 67, 72, 73, 74, 79, 80, 81, 83		

Many MDA tools support both M2M and M2Text transformations. For VIDE, there seems to have emerged a view that the main development process will comprise of PIM modelling, and the generation of counterpart textual VIDE code from VIDE PIM models. This is a central development activity, and experienced developers might want to effect changes to the textual code, which, in our view should be reflected in the visual code.

PIM	Users: **End** | **Analyst** | **Designer** | **Arch** | **Programmer**	**SHOULD**
GUI REQ ID 51	Changes made to the textual model should automatically update the visual model, and visa-versa.	
Related requirement IDs: 8, 9, 10, 15, 34a, 49, 50, 51, 58, 67, 72, 73, 74, 79, 80, 81, 83		

The development of a PIM model will result in analysis steps that require validation with stakeholders. Such validation often requires demonstration of some application logic either at the early stage of application modelling (PIM model development), or based on generated application code. We argue that as part of the validation process, VIDE might consider providing a means to provide visual simulation of implied behaviour, either by use of throw-away prototypes or by providing early behavioural mock-ups of the application.

PIM	Users: **End** | **Analyst** | **Designer** | **Arch** | **Programmer**	**SHOULD**
GUI REQ ID 52	It should be possible to interactively simulate PIM models.	
Related requirement IDs: 7, 52, 53, 54, 55, 56		

PIM	Users: **End** | **Analyst** | **Designer** | **Arch** | **Programmer**	**SHOULD**
GUI REQ ID 53	It should be possible to interactively mark models that fail a validation step.	
Related requirement IDs: 7, 52, 53, 54, 55, 56		

PIM	Users: **End** | **Analyst** | **Designer** | **Arch** | **Programmer**	**SHOULD**
GUI REQ ID 54	It should be possible to interactively mark models that pass a validation step.	
Related requirement IDs: 7, 52, 53, 54, 55, 56		

PIM	Users: **End** | **Analyst** | **Designer** | **Arch** | **Programmer**	**SHOULD**
GUI REQ ID 55	It should be possible to label or graphically annotate a validation step.	
Related requirement IDs: 7, 52, 53, 54, 55, 56		

CIM | **PIM** | **PSM**	Users: **End** | **Analyst** | **Designer** | **Arch** | **Programmer**	**SHOULD**
GUI REQ ID 56	It should be possible to link a validation step with specific CIM, PIM or PSM models.	
Related requirement IDs: 7, 52, 53, 54, 55, 56		

Whereas VIDE situates itself within a revolutionary spectrum of the current MDA support environments, VIDE cannot afford to isolate its users from other well known support environments such as Eclipse, Rational Rose, Objecteering, Together, etc. One way for VIDE to exploit modelling features available in these environments (or for VIDE to be exploited by such environments) is for the VIDE IDE to provide interfaces for these other tools to access VIDE modelling features. Another way, it could be possible for VIDE models to be persisted such that they are accessible by such tools.

CIM | **PIM** | **PSM**	Users: **End** | **Analyst** | **Designer** | **Arch** | **Programmer**	**COULD**
GUI REQ ID 57	It could be possible for VIDE GUI elements to be used in other development environments.	
Related requirement IDs: 11, 57, 68		

4.4.5 General diagramming comments

Whilst much of the diagramming literature detailed in this work highlights the issues with layout a study by Hahn and Kim [117] identifies a number of other factors which influence the use of diagrams in the design environment. Firstly the way diagram allows decomposition into process components is important as is layout organisation. Both factors reduce analysis and design errors.

4.5 Natural and code-less programming environments

4.5.1 Introduction

The final strand of research examines natural and code-less programming environments and the associated work to see what can be learnt from the work done in the area. Some of the applications such as HANDS [118] and SCRATCH [119] are aimed at children and/or novice programmers, whilst other applications such as Limnor [120] and MyDesk [121] are aimed to assist non-programmers to create running applications often in specialist areas.

4.5.2 Examples

4.5.2.1 Human-centred Advances for the Novice Development of Software (HANDS)

The work in the HANDS project aimed to explore how users naturally program, use graphics and process data. The result of the studies would guide the development of a new language and environment called HANDS [118]. The arguments for natural programming are strong given that the transformation from a real world operation such as adding 3 numbers together in the example given in [118] use a single operator in a spreadsheet, but in C code take three kinds of brackets and three kinds of assignment operators in five lines of code.

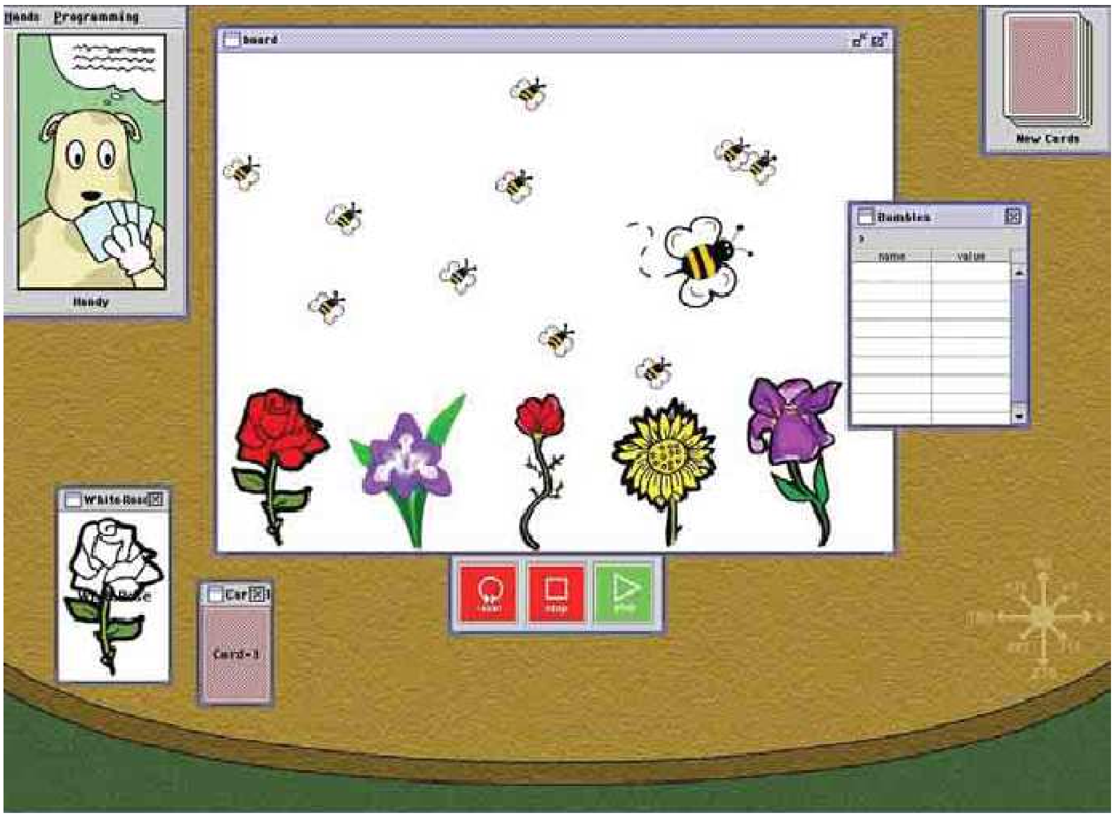
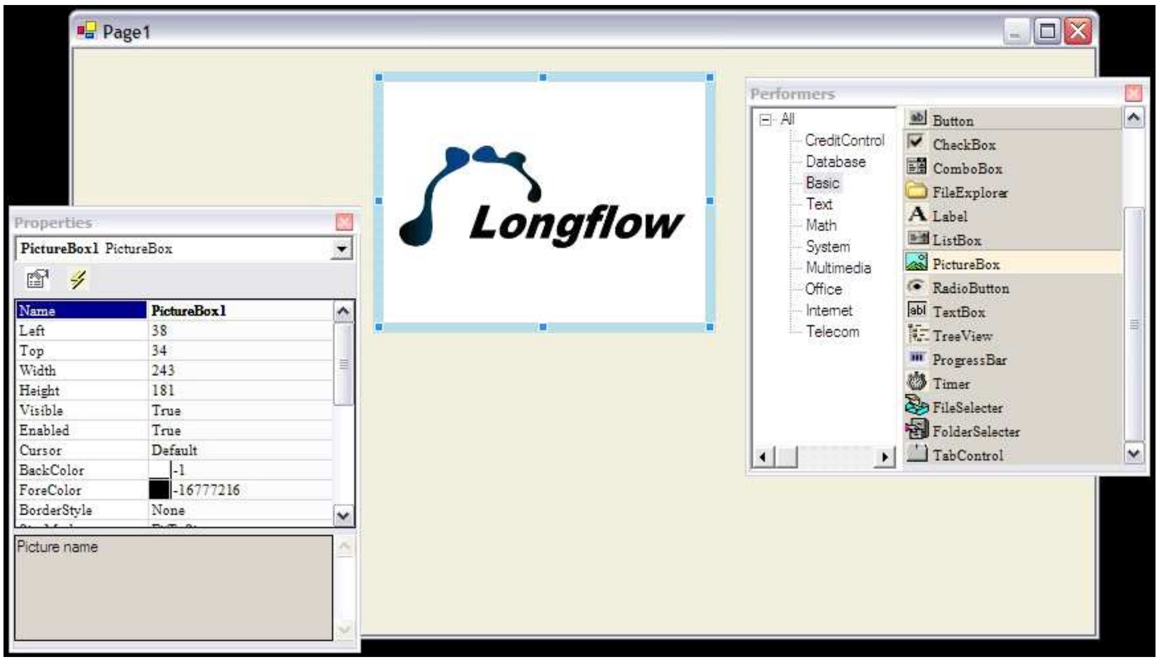


Figure 4: The HANDS programming environment [118]

The dog named Handy sits at a table and manipulates a set of cards. The cards store the data. The application uses an event-based language .

4.5.2.2 Limnor

Limnor allows the developer to drag and drop components called performers into a project and then change their properties as shown in Figure 5.Useful features are the provision of dropdown boxes which give the user choices as to what components are available.



Name	PictureBox1
Left	38
Top	34
Width	243
Height	181
Visible	True
Enabled	True
Cursor	Default
BackColor	-1
ForeColor	-16777216
BorderStyle	None

Figure 5. A Limnor PictureBox 'performer' showing properties [120]

The Limnor performers can then be assigned Actions and those actions can be assigned Events as shown in Figure 6. These are in different screens and formats. The different screens are not anchored and can be placed to suit the working style of the user.

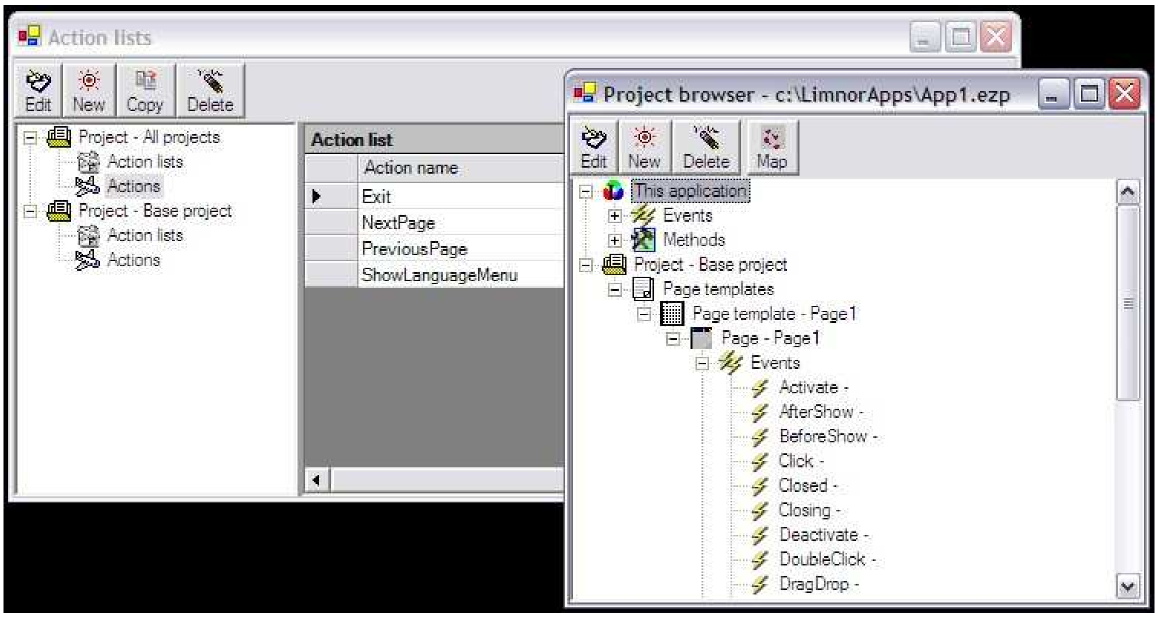


Figure 6. Limnor Actions and Events [120]

Finally the Events can be connected to show program flow as shown in Figure 7

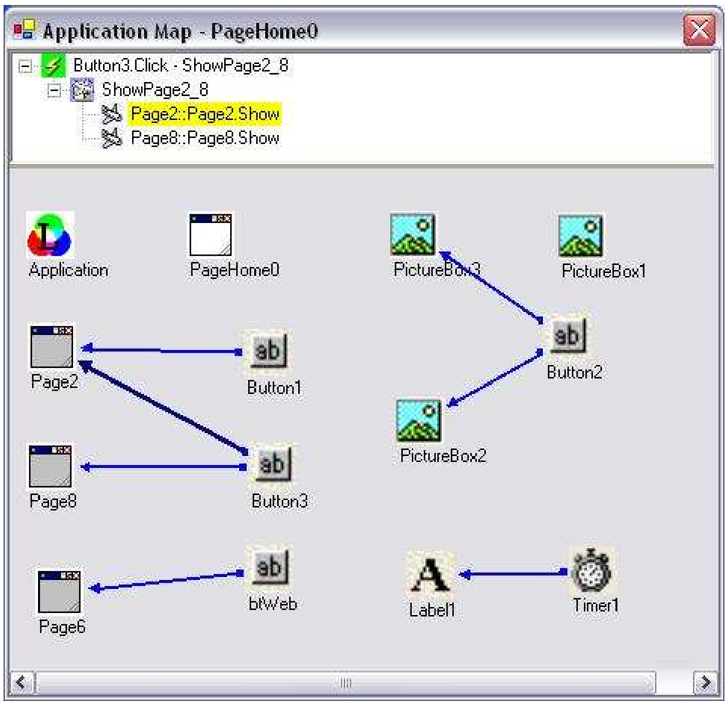


Figure 7. Limnor program control flow

The merits or otherwise of the Limnor visual programming language are not at issue here, but the environment and the process can be examined to identify areas that are useful.

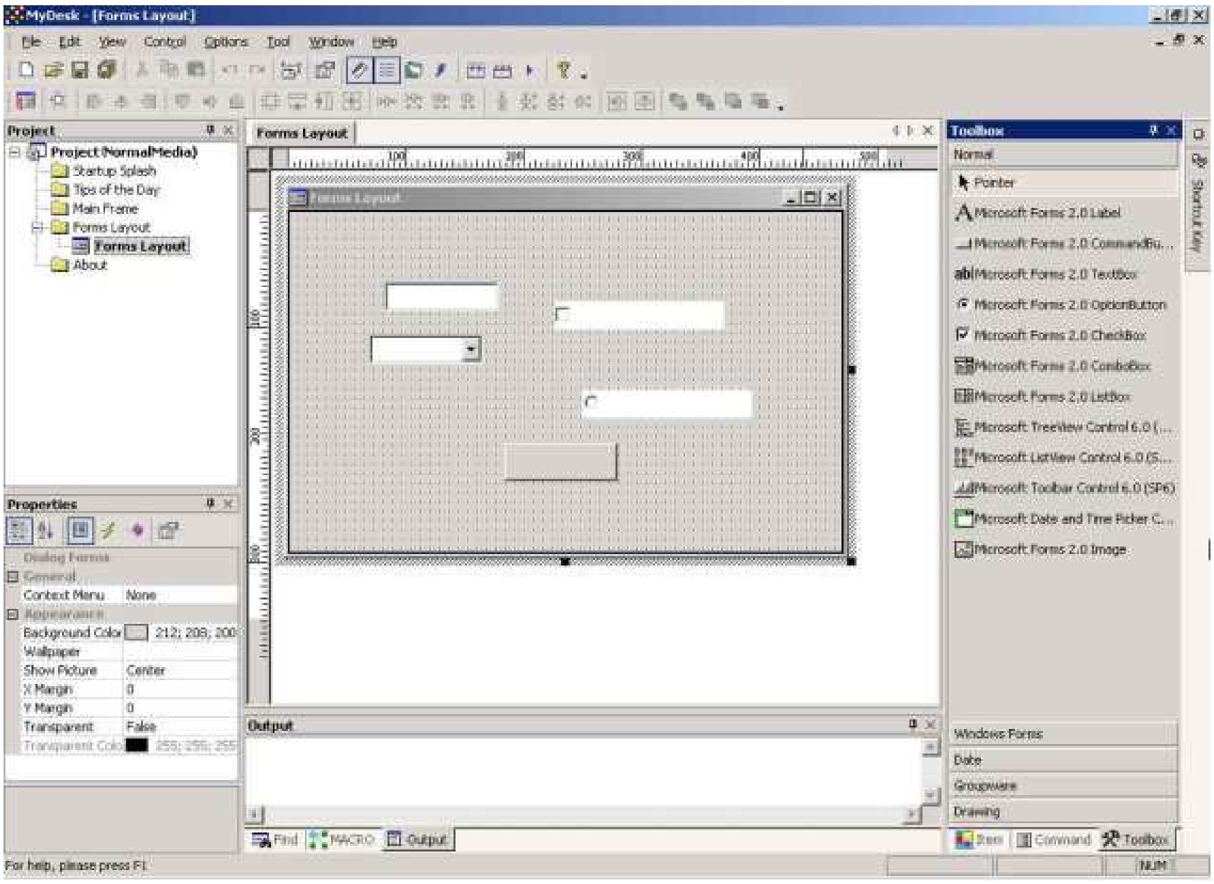
CIM	Users	**SHOULD**
REQ ID 58	The components and contents of any project should be easy to find and identify	
Related requirement IDs: 8, 9, 10, 15, 34a, 49, 50, 51, 58, 67, 72, 73, 74, 79, 80, 81, 83		

CIM	Users	**SHOULD**
REQ ID 59	There should be a standard ‘look and feel’ to the screens at the CIM level.	
Related requirement IDs: 12, 21, 24, 25, 27, 28, 29, 30, 31, 35, 51, 58, 59, 60, 64, 65, 66		

CIM LEVEL	Users	**SHOULD**
REQ ID 60	Providing drop down lists of component types and actions will give the user choices about what is available.	
Related requirement IDs: 12, 21, 24, 25, 27, 28, 29, 30, 31, 35, 51, 58, 59, 60, 64, 65, 66		

4.5.2.3 My Desk 2.0

MyDesk 2.0 by Genusoft [122]is a visual development environment based on ActiveX controls that it can be used in a number of different areas, such as network diagramming and forms layout (see Figure 8).

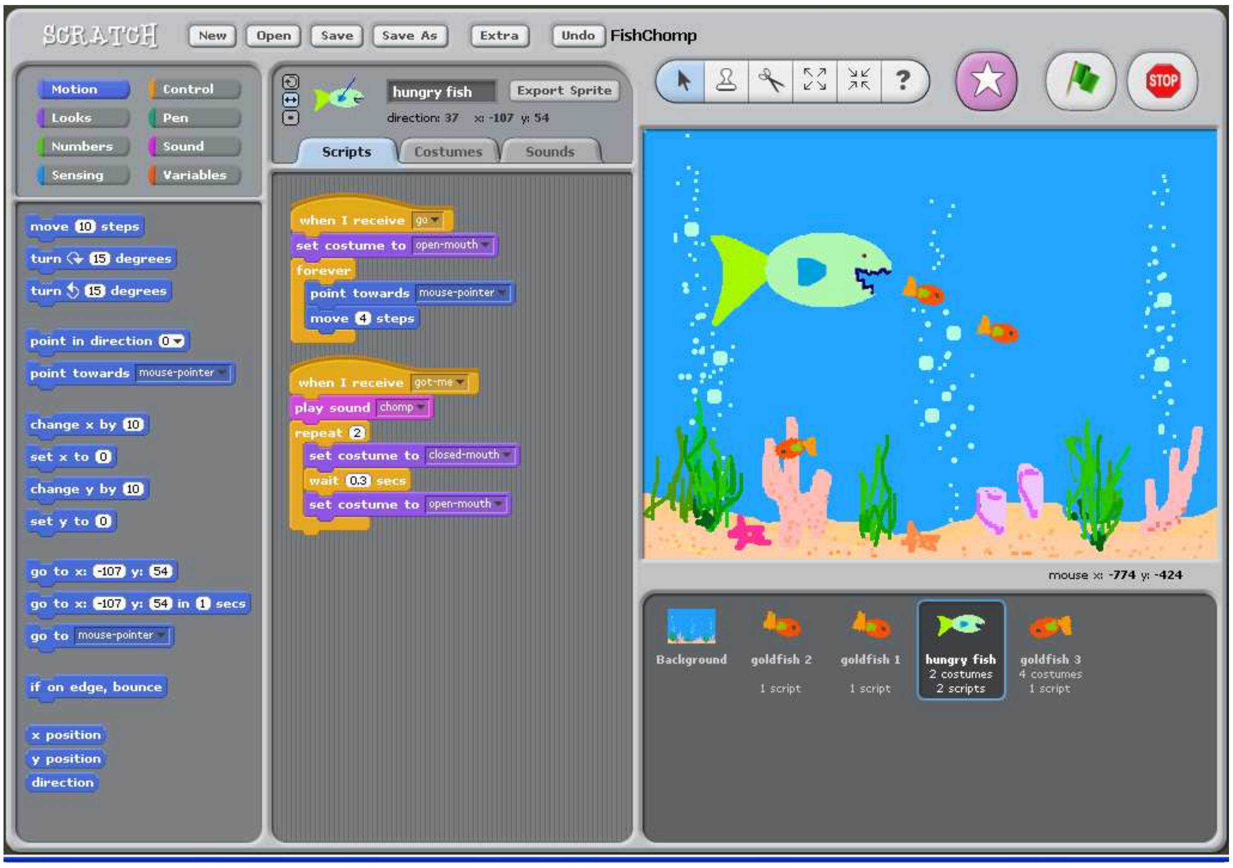

MyDesk - [Forms Layout]
File Edit View Control Options Tool Window Help
Project
Project (NormalMedia)
Startup Splash
Tips of the Day
Main Frame
Forms Layout
Forms Layout
About
Forms Layout
Forms Layout
Toolbox
Normal
Pointer
Microsoft Forms 2.0 Label
Microsoft Forms 2.0 CommandBu...
Microsoft Forms 2.0 TextBox
Microsoft Forms 2.0 OptionButton
Microsoft Forms 2.0 CheckBox
Microsoft Forms 2.0 ComboBox
Microsoft Forms 2.0 ListBox
Microsoft TreeView Control 6.0 (...
Microsoft ListView Control 6.0 (S...
Microsoft Toolbar Control 6.0 (SP6)
Microsoft Date and Time Picker C...
Microsoft Forms 2.0 Image
Windows Forms
Date
Groupware
Drawing
Item
Command
Toolbox
Shortcut Key
Properties
Output
Find
MACRO
Output
For help, please press F1
NUM

Dialog Forms	
General	
Context Menu	None
Appearance	
Background Colo	212; 208; 200
Wallpaper	
Show Picture	Center
X Margin	0
Y Margin	0
Transparent	False
Transparent Colo	255; 255; 255

Figure 8: MyDesk forms layout [121]

4.5.2.4 Scratch

Scratch is described as a programming environment for young people and has been created using a building block metaphor where users have to snap together graphical elements to create a script [123]. The graphical elements can only be connected in a syntactically correct way [119].The layout of the screen shows that the available components and services are on the left, with the current script in the centre and the graphical representation of the script on the right.



```
when I receive go
set costume to open-mouth
forever
    point towards mouse-pointer
    move 4 steps

when I receive got-me
play sound chomp
repeat 2
    set costume to closed-mouth
    wait 0.3 secs
    set costume to open-mouth
```

Figure 9: Example Scratch user interface[119]

4.5.2.5 Denim

Based on research at Berkeley University by Newman and others [124], Denim is an interactive environment for the design and visualisation of web sites using a pen and ink metaphor. It aims to capture the informal tasks that a developer has when designing a web site. The screen layout shows the increasing detail highlighted using the left hand ‘slider’. Options for the user tasks are given using a ‘pie-chart’ wheel with up to six services available.

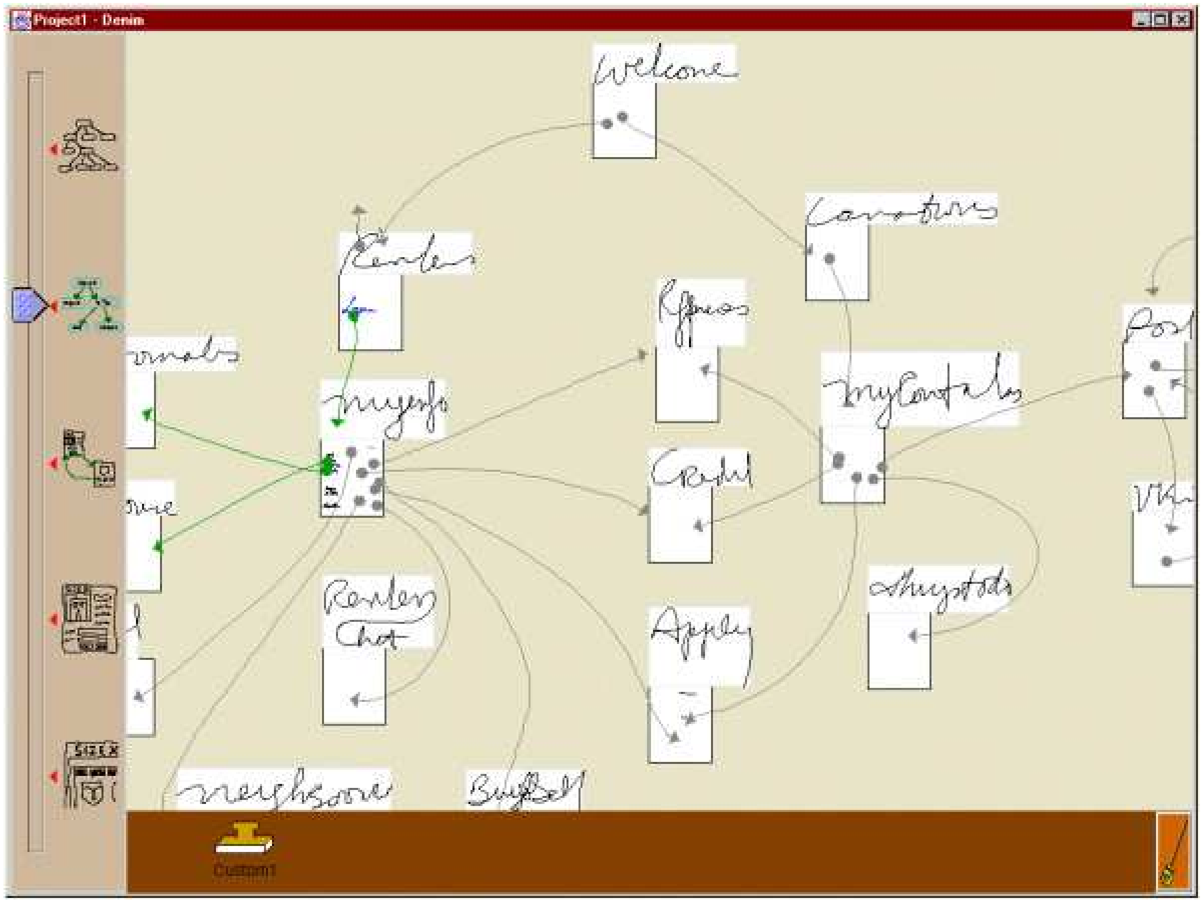


Figure 10: Denim user interface [124]

4.6 Summary

This chapter showed that the research areas of Software Visualisation, Visual Programming, Diagramming and code-less programming have all got features that are useful when creating a list of requirements for an MDA environment. Software Visualisation has much to offer in the area of individual component design whilst the areas of Visual programming and Diagramming have rich research agendas in the design and layout of visual environments. This has enabled the VIDE consortium to use many proven features in the design of the VIDE interface.

5. Exploration of graphical user interfaces

5.1 Introduction

In this chapter, an exploration of graphical user interfaces is undertaken. We focus on some of the mainstream, industrial-strength UML tools such as Objecteering [125], Together [126] and Eclipse [127]. The selection of the tools studied is mainly based on their ubiquity in the modelling arena, and their apparent use in MDA development. Our investigation is based on some of the cognitive dimensions discussed in [128]. In using these assessment tools, we attempt to provide an overview of the extent to which the studied tools support software development tasks within an MDA development process. For example, the cognitive dimension of viscosity (see [128], pg. 12) helps to assess the extent to which models are resistant to change. On the other hand, the dimension of visibility concerns itself with the ability of a modeller to view modelling components easily. The reader is referred to [128, 129] for a full discussion of the cognitive dimensions framework.

For our assessment purposes, this chapter considers the first six cognitive dimensions. The reason for is that they seemed most relevant for the type of activities pertinent to software modelling. Additionally, these dimensions have been extensively discussed with institutive examples that lend themselves to the type of assessment we deem sensible for MDA-based development in [91]. We briefly describe the dimensions used for assessing modelling with the four tools below:

- Viscosity – we will assess the ease with which users can make changes to models within the studied tools. To do this assessment, we will build sample models (e.g., behaviour models) in each of the tools, and try to add or change the models within each of the modelling tool's editors.
- Visibility – 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 – we assess whether the tools provide modellers with a means to view models side by side. This assessment will typically be based on whether tools allow users to view different 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 chapter is whether or not one can derive parts of a given model from another directly, including traceability of information between models.
- Hidden dependencies – we assess whether there are any hidden dependencies between components.
- Premature commitment – we assess the extent to which the tools require premature commitment during modelling on the part of the users.
- Secondary notation – we assess whether tools provide secondary notation to provide extra information about models.

5.2 Modelling tools

The UML comprises notations for describing various aspects of a software system. For example, the use case notation is primarily for expressing software requirements and specifications. Tool vendors have developed support environments for this UML modelling. Some of the longstanding tools with thriving user bases, include Borland's Together [126], IBM's Rational Rose [10], the Objecteering UML modeller [125], and Eclipse [127].

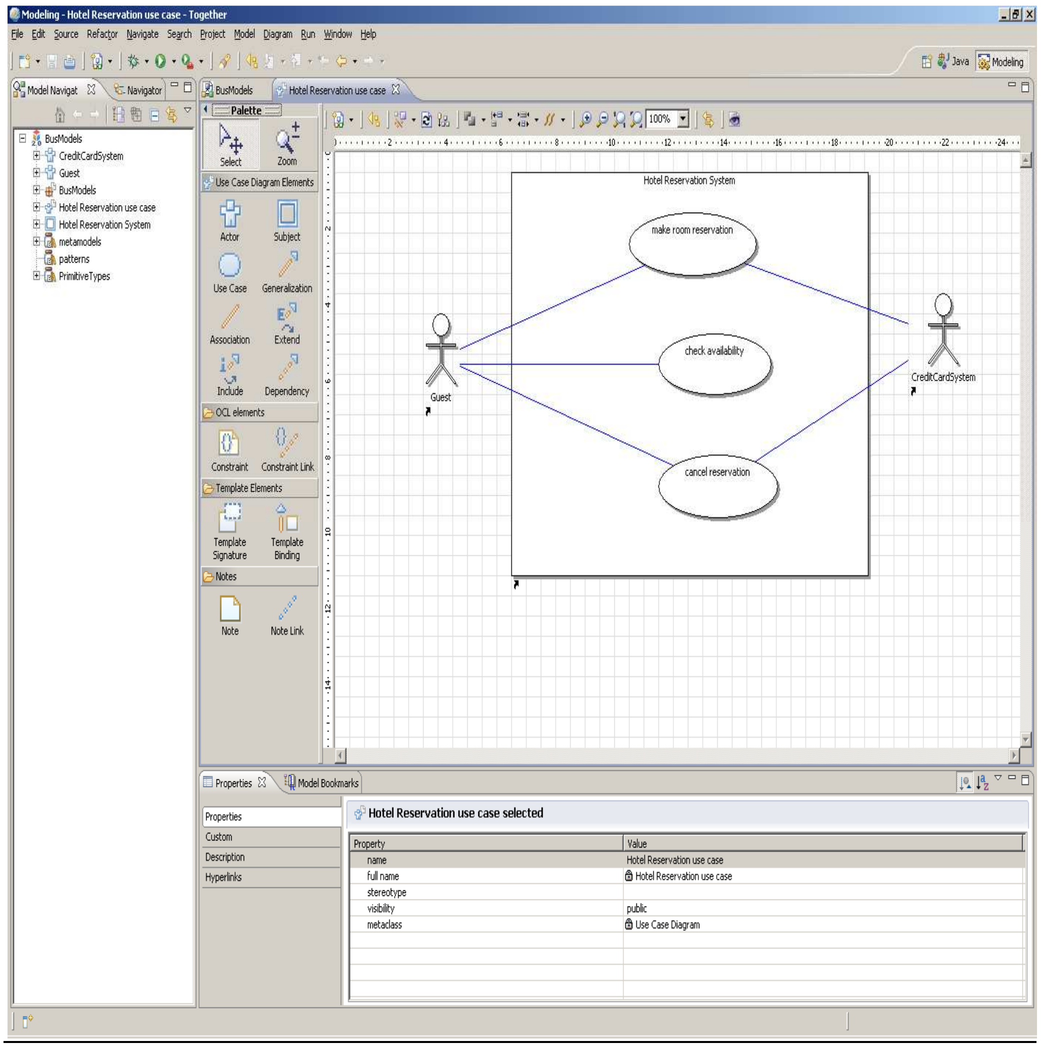
In this section, we provide an overview of the way in which these tools represent use case models. We review the similarities and differences between use case models in these tools, and the extent to which we the tools provide users with facilities to amend or add to use case models within each of the tools.

5.2.1 Use case models

Consider 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.

5.2.1.1 Use case model - Together

An example use case model to show some of these business functions is shown below. The model is built using the Together UML tool:



Hotel Reservation use case selected

Property	Value
name	Hotel Reservation use case
full name	Hotel Reservation use case
stereotype	
visibility	public
metaclass	Use Case Diagram

Figure 11: Use case model in Together 2007

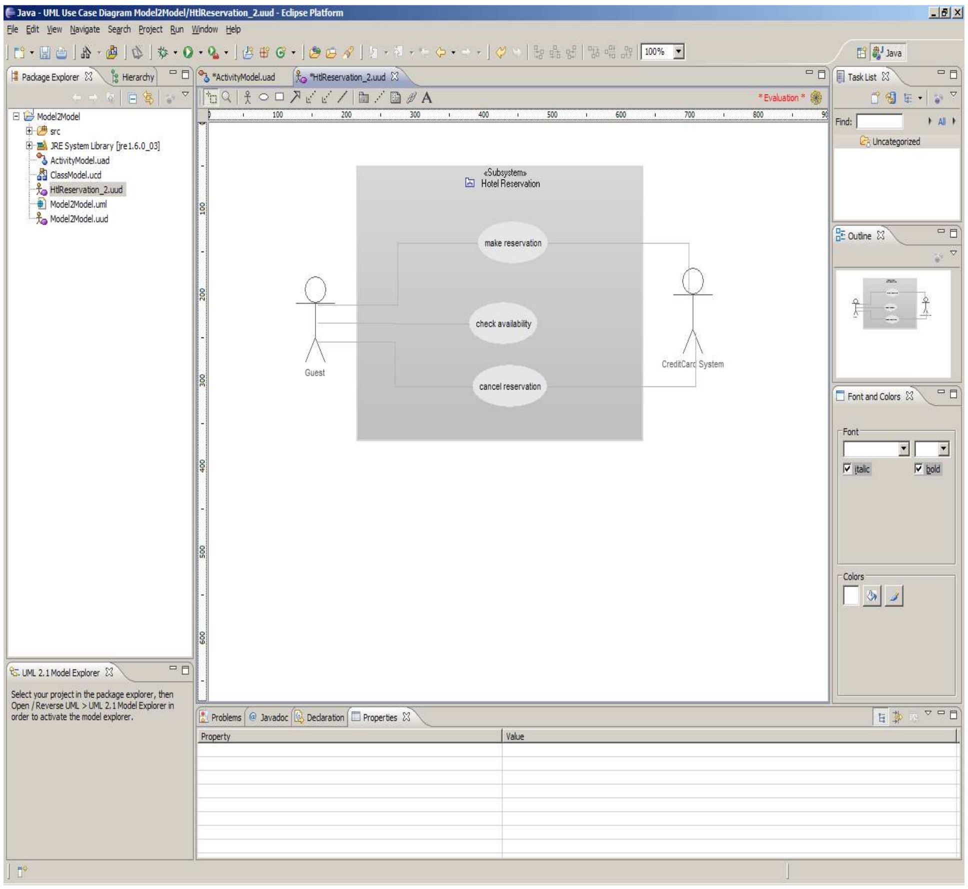
The Together CASE tool requires a user to be familiar with the notion of a project, and indeed, a model such as the use case in Figure 11 does belong to a UML type of project. Other project types include a Java project, Plug-in project, etc. Hence, the project to which the use case model above belongs may comprise of other models, 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. There does not appear to be any visibility issues with these components for the use case model.

A few issues regarding amending or adding to a Together use case model were encountered. 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 the links and linked elements have to be moved to appropriate positions 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 elements in the palette are not quite 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 is Columns as it seemed more visible.

5.2.1.2 Use case model - Eclipse

Unlike in Together, use case modelling components in Eclipse are laid out at the top of the model editor space as shown in Figure 12:



Select your project in the package explorer, then Open / Reverse UML > UML 2.1 Model Explorer in order to activate the model explorer.

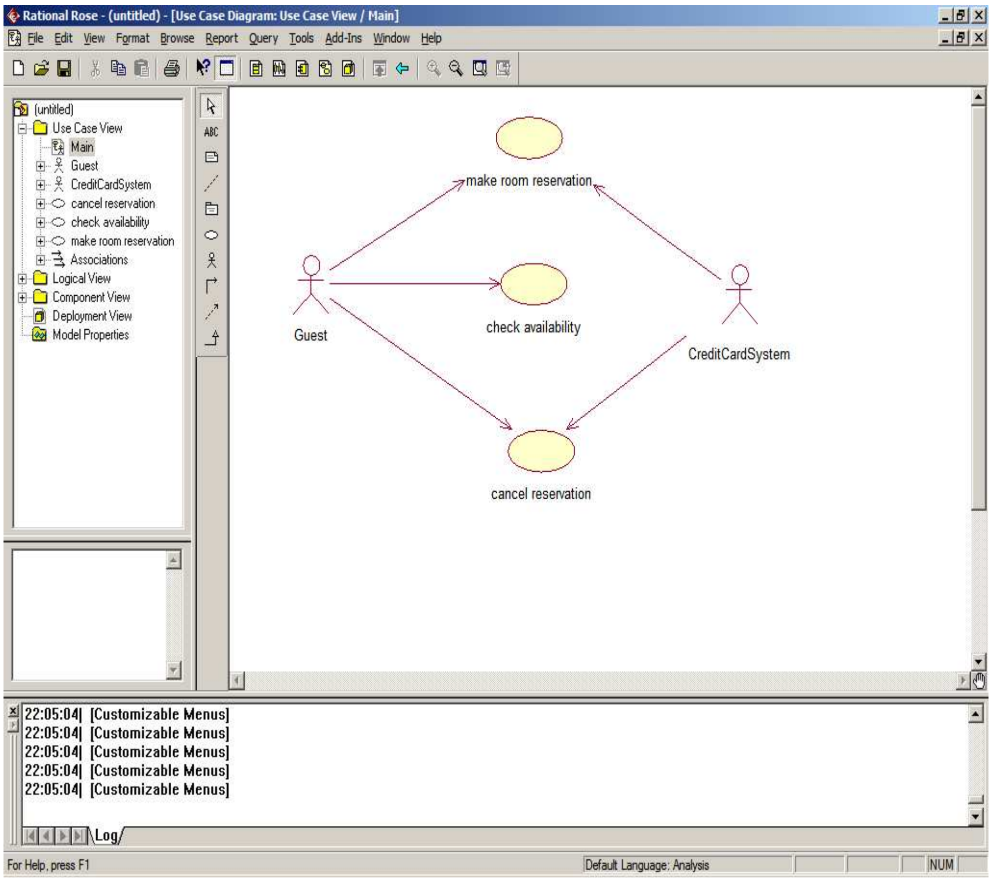
Figure 12: Use case model in Eclipse

An issue with building the use case model in Eclipse is that it is not possible for a user to lay out associations in a style that the user wants. That is, moving an actor or a use case, or indeed adding an extra use case causes the Eclipse modeller to move the 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 much time can 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.

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 system scope.

5.2.1.3 Use case model – Rational Rose

In Rational Rose modelling, 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 (see Figure 13) much like in Together :



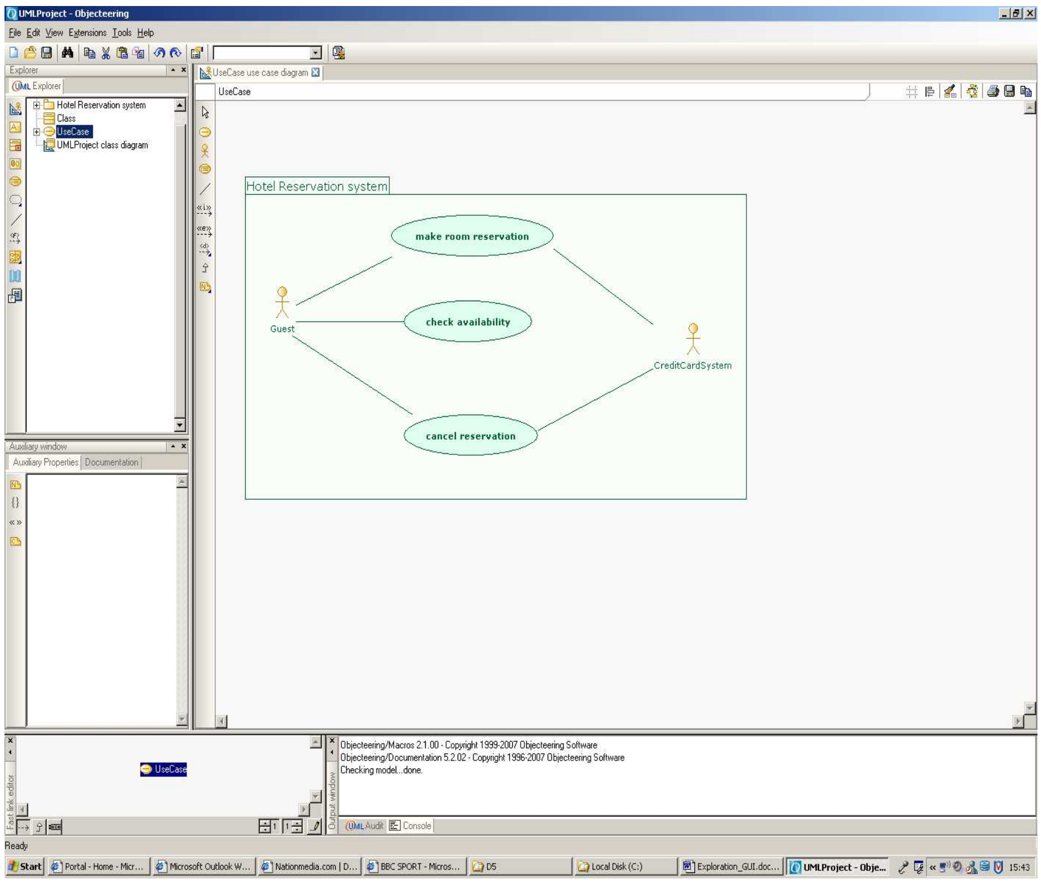
```
22:05:04|  [Customizable Menus]
22:05:04|  [Customizable Menus]
22:05:04|  [Customizable Menus]
22:05:04|  [Customizable Menus]
22:05:04|  [Customizable Menus]
```

Figure 13: Use case model in Rational Rose

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 can be felt 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 needing to move other use cases/actors below or in the neighbourhood. Rational Rose allows the modeller to write a specification for each use case element. Since descriptions aren't part of the standard notation, one can argue that Rational Rose allows modellers the use of a secondary notation to provide more detail about the use case using unstructured text.

5.2.1.4 Use case model - Objecteering

Objecteering requires that a use case model cannot be a standalone model. That is, an Objecteering use case model must belong to either a package or a class. Figure 14 shows an Objecteering use case model:



```
Objecteering/Macros 2.1.00 - Copyright 1999-2007 Objecteering Software
Objecteering/Documentation 5.2.02 - Copyright 1996-2007 Objecteering Software
Checking model...done.
```

Figure 14: Use case model in Objecteering

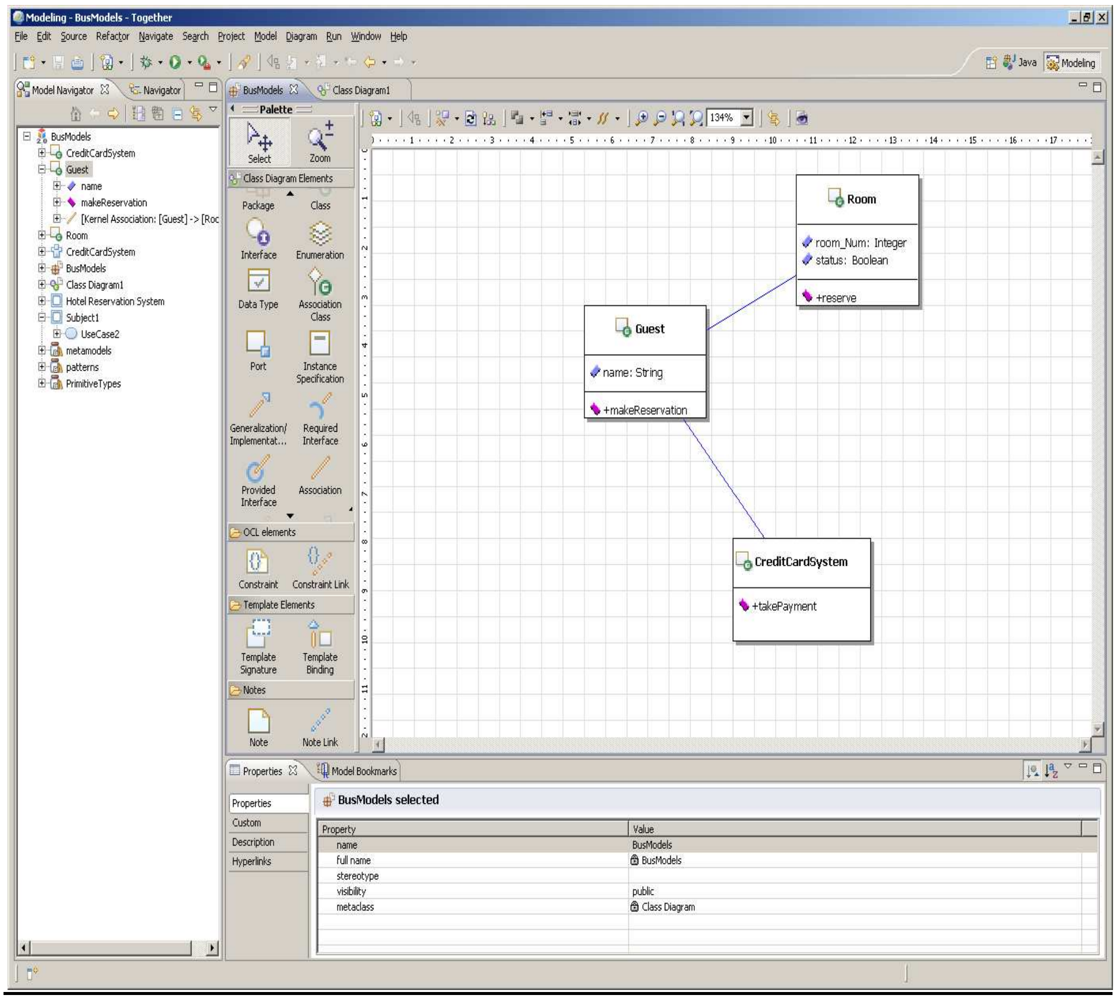
Objecteering (like Together and Rational Rose) allows modellers the flexibility of positioning use case elements on their chosen screen areas. The tool places modelling objects in a vertical toolbar to the left of the model editor window. Like the other tools seen so far, Objecteering models belong to a project that a user must create prior to any model construction. There seems to be an implicit dependency between use case models and containing elements (e.g., package or class). That is, since Objecteering requires a use case to belong to a package or class, there seems to be an implied hidden dependency between such artefacts. It might be that the dependency is simply to be able to group use cases within a package the same way one would group together class models in a package. Where a use case is associated with a class, the dependency might be to imply the use cases being realised by the class.

5.2.2 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.

5.2.2.1 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 11. 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 below was built without such direct derivation:



BusModels selected

Property	Value
name	BusModels
full name	BusModels
stereotype	
visibility	public
metaclass	Class Diagram

Figure 15: Class model in Together

Class modelling components in Together are laid in a vertical toolbar, just like use case modelling components are in the same tool. The class modelling components have clear visibility in the Columns layout. In List layout, however, (even in Details Layout), the visibility of the class components diminishes remarkably. 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. That requires moving lower level classes and associations around to produce a legible model. The class model in Figure 15 is based on the previous use case model (Figure 11).

It is likely that modellers might want to derive some classes from a use case model. However, there is no way for a user to lay 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 was no way to derive the classes (or other class model elements) from the use case model. Additionally, there was no way to display the use case model adjacent to the class editor for reference while building the class model.

5.2.2.2 Class model - Eclipse

Eclipse behaves quite differently from Together when building a class model. For example, immediately 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 cardinality of the roles.

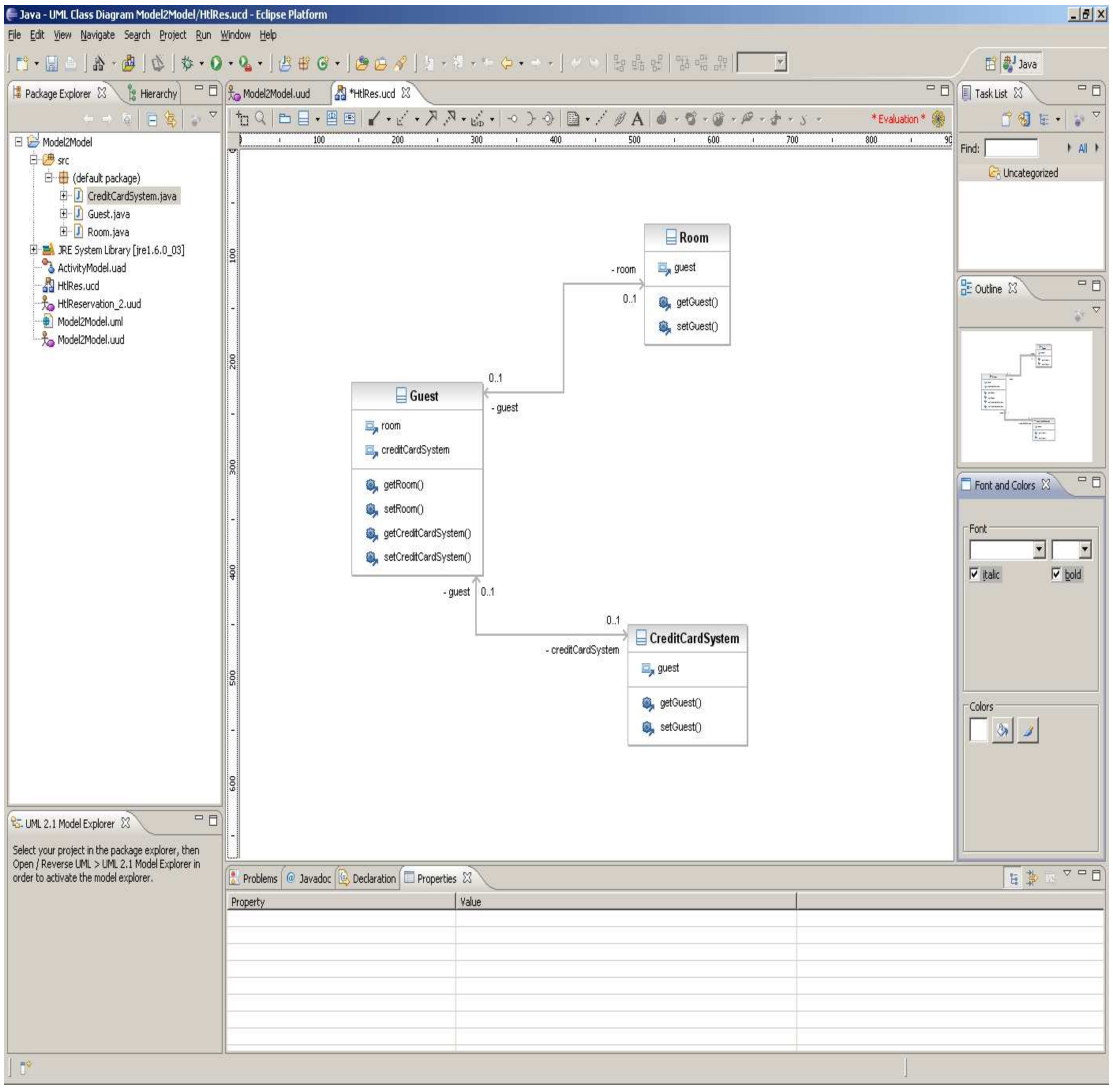
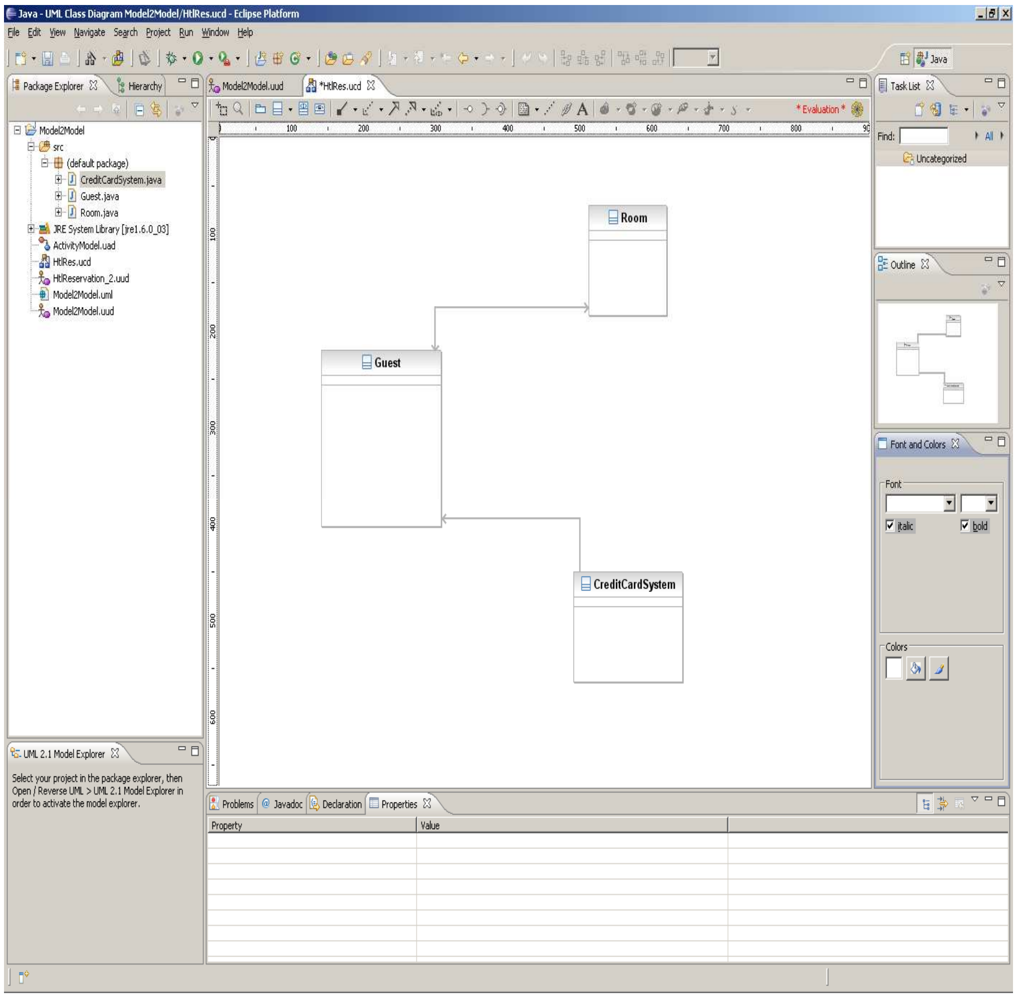

Select your project in the package explorer, then Open / Reverse UML > UML 2.1 Model Explorer in order to activate the model explorer.

Figure 16: Class model in Eclipse

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 (Java) code generation. Eclipse also lays out the classes in the model based on the class size and where on the screen a class is positioned. For example, in Figure 16, moving the Room class will cause Eclipse to redraw the association between Guest and CreditCardSystem differently. This is not intuitive since a modeller might wish to place classes and association lines in screen areas where they feel most legible.

In Eclipse, drawing components for say, the class diagram 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 were built from scratch as there was no facility to move any elements of a use case, or activity diagram into the class model.

A modeller may edit the roles, cardinality or the methods to only depict a general class diagram as in Figure 17:



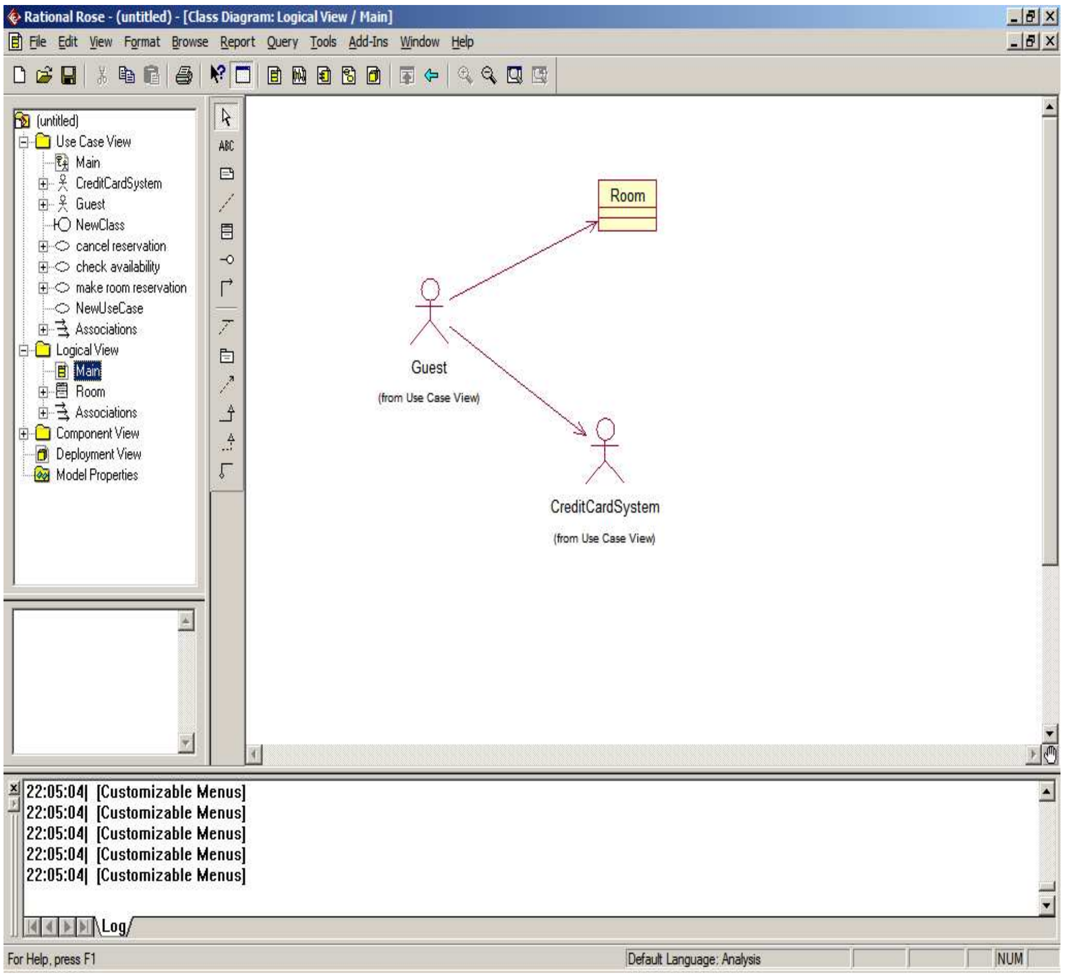
Select your project in the package explorer, then Open / Reverse UML > UML 2.1 Model Explorer in order to activate the model explorer.

Figure 17: Edited class model in Eclipse

Eclipse 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.

5.2.2.3 Class model – Rational Rose

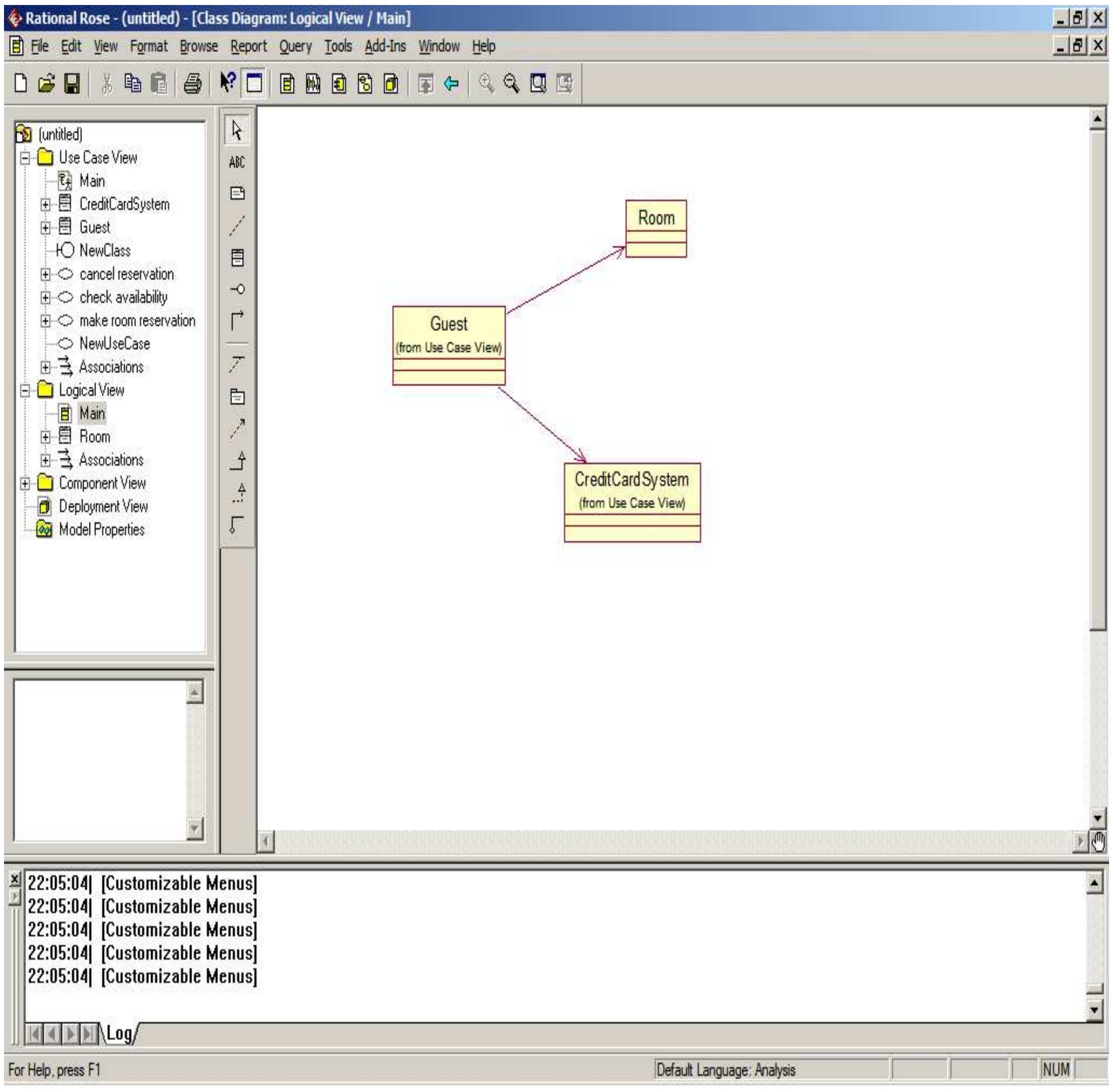
In Rational Rose, building class models seems straightforward, but there is no facility to drag a use case element and use it as a class or class property or method. However, if a modeller builds a class that is given the same name as an existing actor, Rational Rose gives the class an icon that has the look of an actor, with a stereotype indicating the class is from the use case view (see Figure 18).



```
22:05:04|  [Customizable Menus]
22:05:04|  [Customizable Menus]
22:05:04|  [Customizable Menus]
22:05:04|  [Customizable Menus]
22:05:04|  [Customizable Menus]
```

Figure 18: Class model in Rational Rose

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 (see Figure 19).



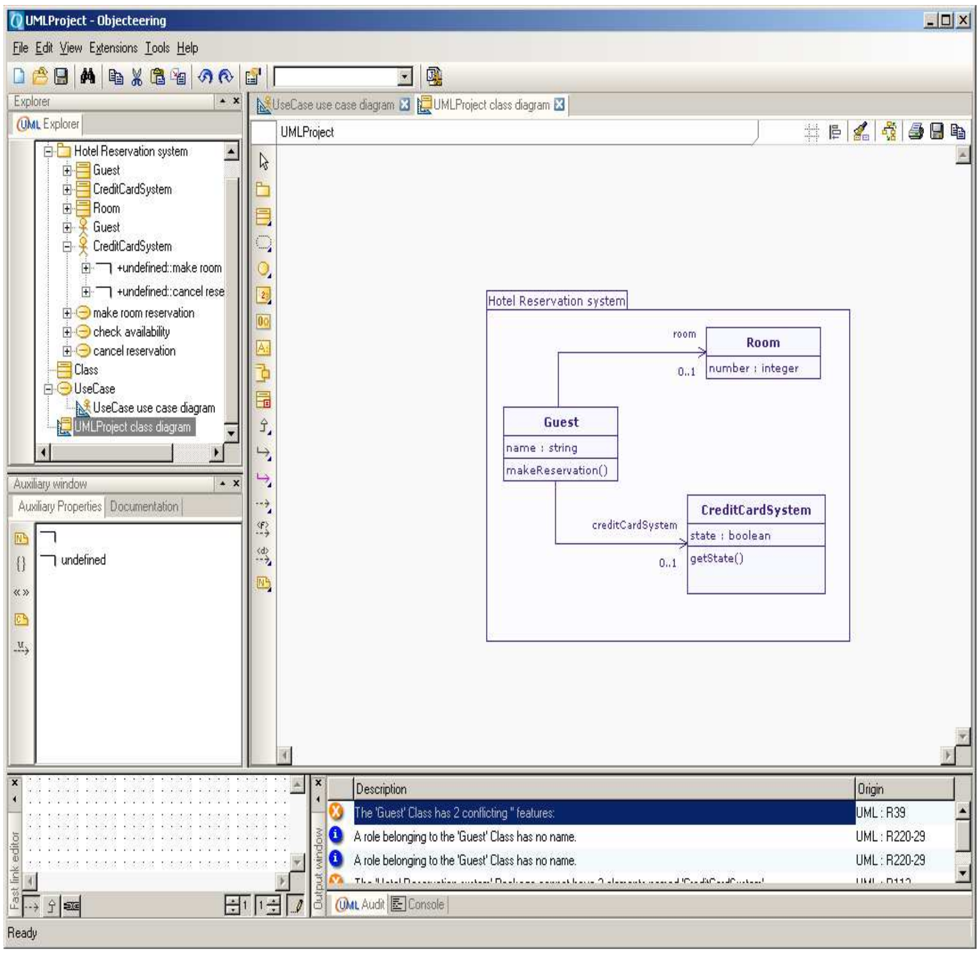
```
22:05:04|  [Customizable Menus]
22:05:04|  [Customizable Menus]
22:05:04|  [Customizable Menus]
22:05:04|  [Customizable Menus]
22:05:04|  [Customizable Menus]
```

Figure 19: Amended class model in Rational Rose

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 some premature commitment to the modeller in deciding that such a relationship between objects in the different models exist. Moreover, there seems to be 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.

5.2.2.4 Class model - Objecteering

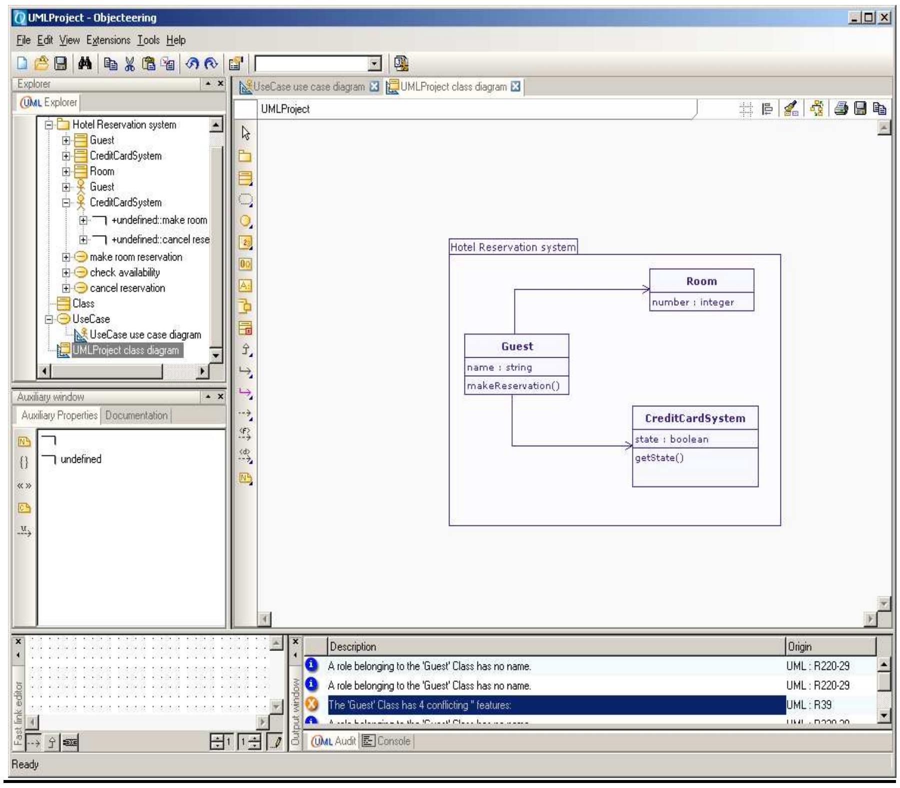
In Objecteering, building a class is relatively easy, and Figure 20 shows how the tool generates roles and multiplicities in the first instance:



	Description	Origin
	The 'Guest' Class has 2 conflicting '' features:	UML : R39
	A role belonging to the 'Guest' Class has no name.	UML : R220-29
	A role belonging to the 'Guest' Class has no name.	UML : R220-29

Figure 20: Class model in Objecteering

Class models in Objecteering belong to a project that must be created prior to progressing with model construction. Class modelling components are laid out in a toolbar to the left of the model editor window. Figure 21 shows how the modeller has an option to edit the first model, to remove generated roles and multiplicities:



	Description	Origin
ℹ	A role belonging to the 'Guest' Class has no name.	UML : R220-29
ℹ	A role belonging to the 'Guest' Class has no name.	UML : R220-29
✗	The 'Guest' Class has 4 conflicting '' features:	UML : R39

Figure 21: Edited class model in Objecteering

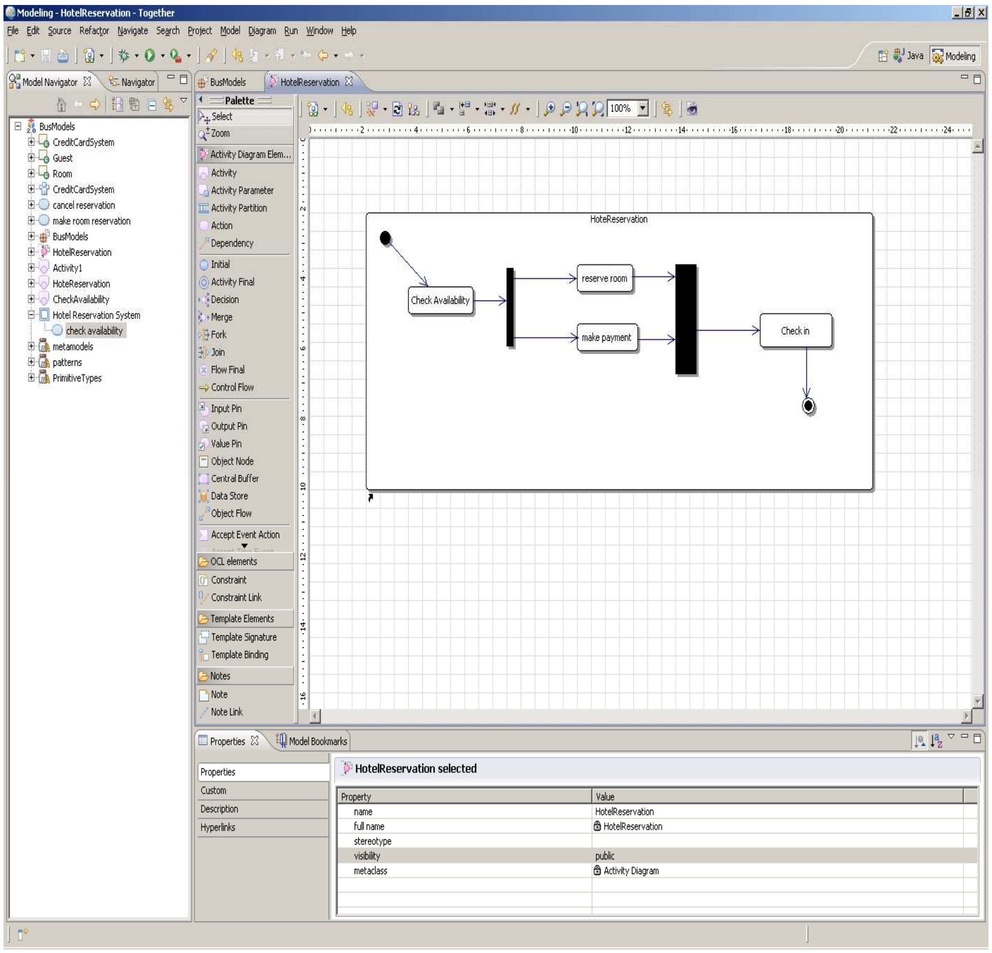
Like the other tools Objecteering does not provide a means to use the elements of a use case model (e.g., actors) to construct classes automatically.

5.2.3 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 reference use case is that already seen in 5.2.1.

5.2.3.1 Activity model - Together

The activity diagram was produced to depict the activities for making a reservation for a hotel room:



HotelReservation selected

Property	Value
name	HotelReservation
full name	HotelReservation
stereotype	
visibility	public
metaclass	Activity Diagram

Figure 22: Activity model in Together – components in list view

The set of components for drawing an activity model in Together occupies a bigger real estate than those for drawing use cases. Hence, the layout for the components (at the left of model editor window) was changed from columns to list, to allow for most of them to be shown. The columns layout of the components is shown in Figure 23.

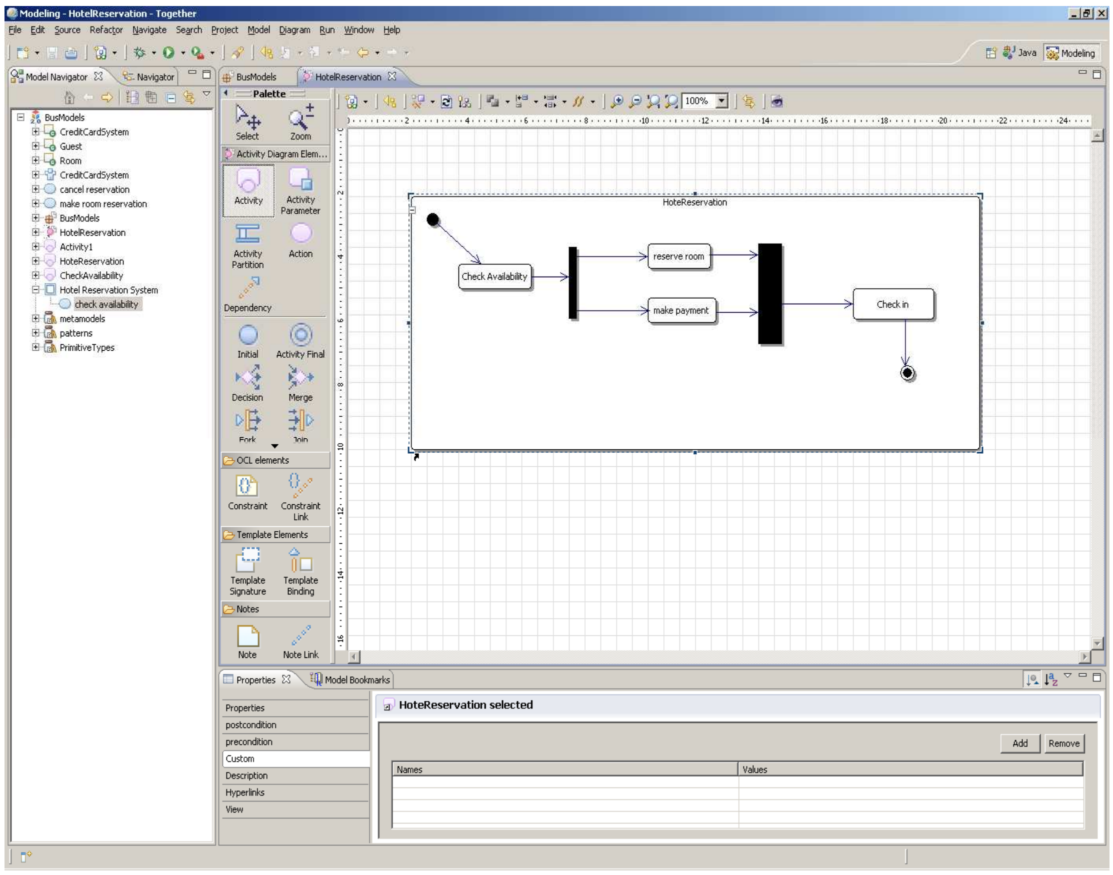


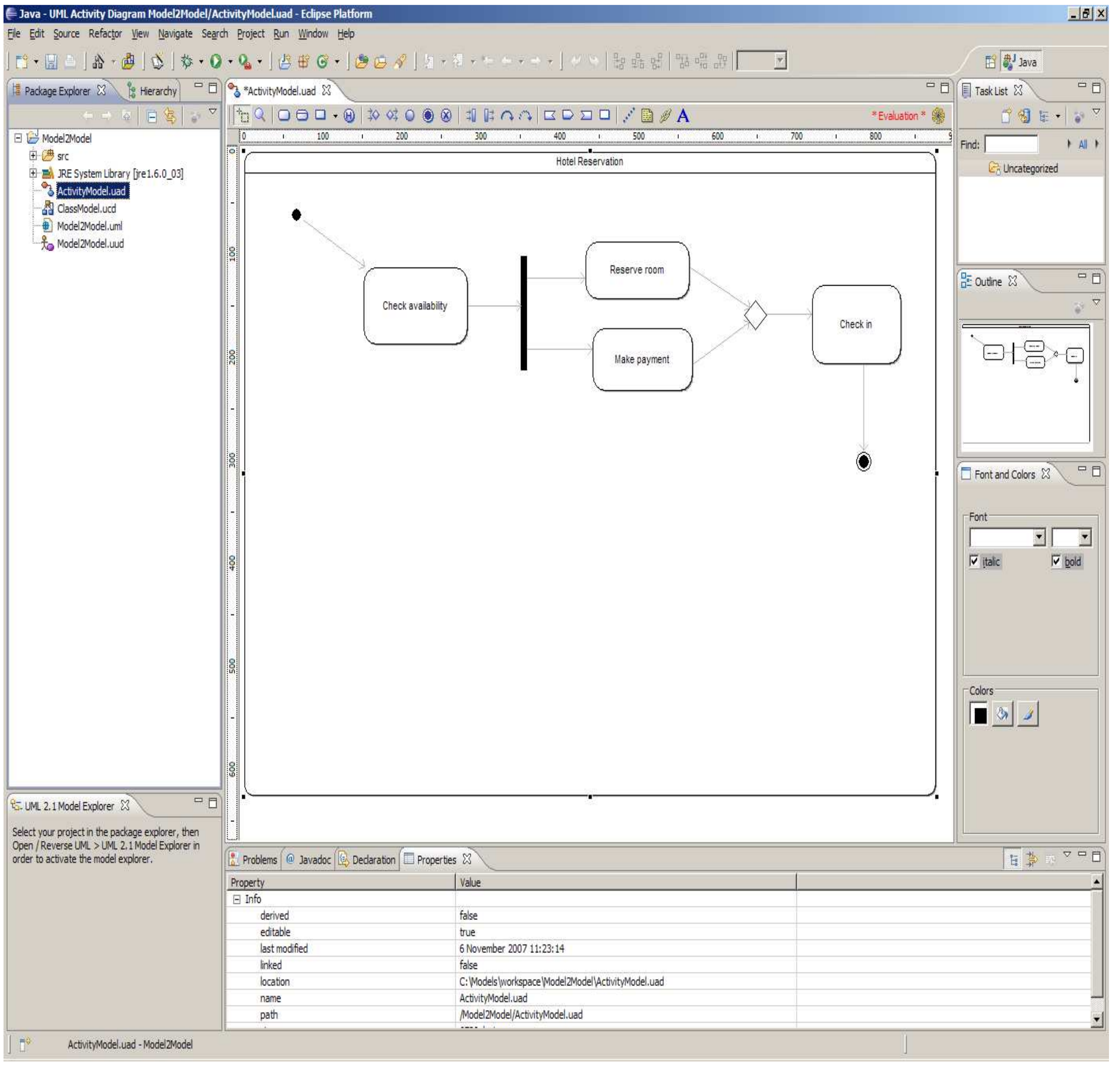
Figure 23: Activity model in Together – components in column view

The icons occupy larger 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. For example, some use cases, such as check availability seemed to be sensible activities for the activity model, but Together does not provided automated support for deriving such activities from the use case model.

5.2.3.2 Activity model - Eclipse

Unlike Together, the drawing components for activities in Eclipse are spread at 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.



Select your project in the package explorer, then Open / Reverse UML > UML 2.1 Model Explorer in order to activate the model explorer.

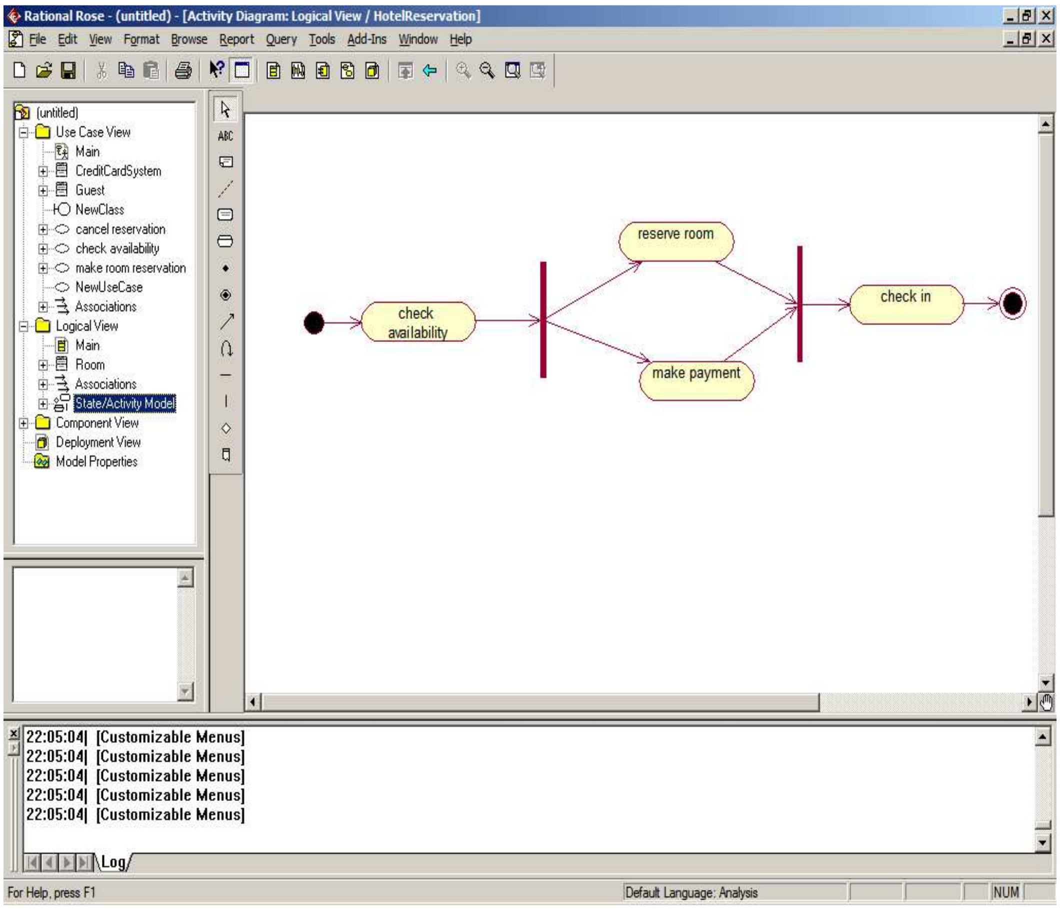
Property	Value
Info	
derived	false
editable	true
last modified	6 November 2007 11:23:14
linked	false
location	C:\Models\workspace\Model2Model\ActivityModel.uad
name	ActivityModel.uad
path	/Model2Model/ActivityModel.uad

Figure 24: Activity model in Eclipse

Eclipse does not provide any other layout for the drawing components. Also, Eclipse does not allow the use of the vertical bar to merge activities.

5.2.3.3 Activity model – Rational Rose

In Rational Rose, as with Eclipse and Together, it was possible to obtain activities from either use case or class models. A Rational Rose activity chart is shown in Figure 25.

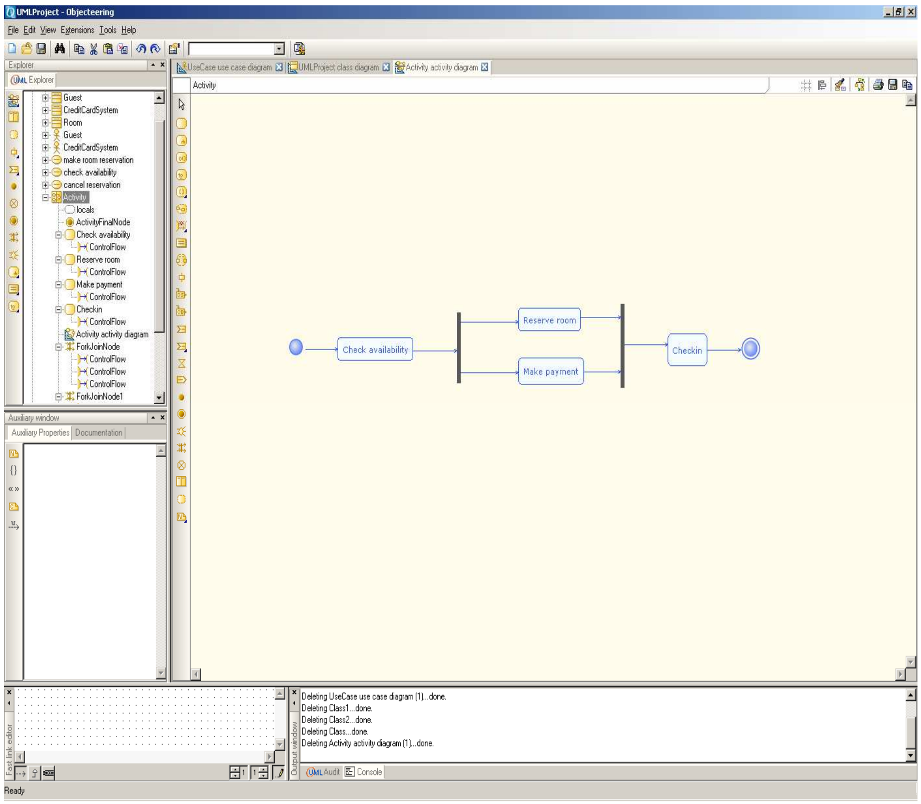


```
22:05:04|  [Customizable Menus]
22:05:04|  [Customizable Menus]
22:05:04|  [Customizable Menus]
22:05:04|  [Customizable Menus]
22:05:04|  [Customizable Menus]
```

Figure 25: Activity model in Rational Rose

5.2.3.4 Activity model – Objecteering

Objecteering allows the use of a vertical bar for splitting and joining activities. In Figure 26 we show an activity model built using Objecteering.



```
Deleting UseCase use case diagram (1)...done.
Deleting Class1...done.
Deleting Class2...done.
Deleting Class...done.
Deleting Activity activity diagram (1)...done.
```

Figure 26: Activity model in Objecteering

Again, there are no means to directly obtain information from use cases or classes for building the activity model. That is, one cannot use actors, use cases, classes or class methods/attributes directly in constructing the activity model. Objecteering however offers a neat way of arranging models together in a tree structure so that a modeller can flip between models for reference purposes.

5.3 Discussion of requirements

Our development of the initial prototype, and the investigation of various UML-based and MDA tools has provided insight into desirable features for the VIDE IDE. The requirements for the VIDE GUI are outlined below.

The VIDE GUI is to provide an editor for constructing business process models. The VIDE IDE will provide RAD business process modelling elements, with a choice for users to deploy the VCLL component for BPMN-based process modelling:

Level PIM	Users: All	**MUST**
GUI REQ ID 61	Support for business process modelling	
Related requirement IDs: 12, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 31, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 61		

Another important feature of the VIDE IDE is to provide a textual editor where users can write descriptions of the domain. Such descriptions may also include pictures, or organisation diagrams:

Level CIM	Users: All	**MUST**
GUI REQ ID 62	Support for construction of descriptions of the domain	
Related requirement IDs: 12, 14, 26, 27, 28, 29, 32, 33, 58, 59, 62, 63, 66, 75, 76		

The VIDE GUI should provide support for users to build analysis models depicting domain entities and their associations. Such entities may be roles, activities, data items, or uncharacterised entities that may later be decomposed or identified as concrete roles, activities, etc.

Level CIM	Users: All	**SHOULD**
GUI REQ ID 63	Provide a means for building an analysis model based on domain descriptions.	
Related requirement IDs: 12, 14, 26, 27, 28, 29, 32, 33, 58, 59, 62, 63, 66, 75, 76		

The VIDE IDE should provide support for further analysis of initially, less understood parts of the domain or entities:

Level All	Users: All	**SHOULD**
GUI REQ ID 64	Provide support for decomposing associations among entities in an analysis model.	
Related requirement IDs: 12, 21, 24, 25, 27, 28, 29, 30, 31, 35, 51, 58, 59, 60, 64, 65, 66		

There should be a means within the VIDE IDE for users to create links among elements of an analysis model:

Level All	Users: All	**SHOULD**
GUI REQ ID 65	Support for creating links/associations among entities, and to label them.	
Related requirement IDs: 12, 21, 24, 25, 27, 28, 29, 30, 31, 35, 51, 58, 59, 60, 64, 65, 66		

The VIDE IDE, like any other development environment, should provide the project construct for holding together models of a given development task:

Level All	Users: All	**SHOULD**
GUI REQ ID 66	Support for creation of projects as container for VIDE models.	
Related requirement IDs: 12, 21, 24, 25, 27, 28, 29, 30, 31, 35, 51, 58, 59, 60, 64, 65, 66		

The VIDE environment is to provide functionality for model storage. This is essential for further development and sharing of models for a given development project:

Level All	Users: All	**MUST**
GUI REQ ID 67	A repository for VIDE models providing persistence of models	
Related requirement IDs: 8, 9, 10, 15, 34a, 49, 50, 51, 58, 67, 72, 73, 74, 79, 80, 81, 83		

An interface is to be provided where users can launch the VCLL modeller within VIDE in order to develop BMPN-based business process models:

Level CIM	Users: All	**MUST**
GUI REQ ID 68	Provide an interface to other components such as the VCLL component.	
Related requirement IDs: 6, 68, 84		

An important aspect of the VIDE IDE is the provision of support for direct of elements of an analysis model in building a formal business process model:

Level PIM	Users: All	**MUST**
GUI REQ ID 69	Support for development of business process models from analysis models	
Related requirement IDs: 1, 2, 3, 16, 17, 18, 19, 20, 21 ,22, 23, 24, 25, 26, 27, 28, 35, 60, 64, 69		

One important aspect of development with the VIDE IDE is the need to be able to build (behaviour) activity models that are derived from a CIM model:

Level PIM	Users: All	**SHOULD**
GUI REQ ID 70	Support for development of behaviour models from CIM models	
Related requirement IDs: 63, 70, 71, 82		

An additional feature of the VIDE IDE is to enable the use of elements of an activity model to build a class design model:

Level PIM	Users: All	**SHOULD**
GUI REQ ID 71	Support for development of class design models from behaviour models.	
Related requirement IDs: 63, 70, 71, 82		

The VIDE IDE is to ensure that elements of a domain model that are derived from the domain description can be traced back to the description:

Level PIM	Users: All	**SHOULD**
GUI REQ ID 72	Provide traceability between analysis models and domain descriptions	
Related requirement IDs: 8, 9, 10, 15, 34a, 49, 50, 51, 58, 67, 72, 73, 74, 79, 80, 81, 83		

The VIDE IDE is to ensure that elements of a business process model that are derived from an analysis model (or domain description) are traceable back to the originating word or element:

Level PIM	Users: All	**SHOULD**
GUI REQ ID 73	Provide traceability between business process models and analysis models	
Related requirement IDs: 8, 9, 10, 15, 34a, 49, 50, 51, 58, 67, 72, 73, 74, 79, 80, 81, 83		

The VIDE IDE is to ensure that any elements of a class design that are derived from an activity model can be traced back to the originating activity model elements:

Level PIM	Users: All	**SHOULD**
GUI REQ ID 74	Provide traceability between class design models and behaviour models	
Related requirement IDs: 8, 9, 10, 15, 34a, 49, 50, 51, 58, 67, 72, 73, 74, 79, 80, 81, 83		

5.4 Summary

There seems to be 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. VIDE modelling will use the “Project” concept in the same way seen in these tools to be able to organise analysis and design models during MDA development with VIDE.
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. VIDE will use a vertical toolbar to hold modelling components for the different modelling views. Some tools (e.g., Eclipse) do not provide a flexibility in laying out models on the model editor. VIDE will allow users to position model elements on the screen area as they wish. Table 22 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	Objecteering	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	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.	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	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 modelling use cases – one has to build use cases as part of a class model or a package	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	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	Seems to force naming of roles and specification of multiplicity	Seems to force association of a use case to another objects, e.g. class or package	Seems to force implication of a class to be based on an actor where a similarly named actor exists.

Table 22: Summary of tools assessment with cognitive dimensions

An important point to reiterate is that the mainstream tools assessed do not provide a means to carry forward information from one model (e.g., use case) to another (e.g., class or activity model). VIDE will provide a means for users to select elements of an analysis model (e.g., entities, roles, activities) for use in developing a design model. Furthermore, VIDE will provide traceability between models to indicate where parts of a given model have been derived from parts of another model (s).

6. Exploratory Prototype

6.1 Introduction

Prototyping is the most frequently used of the modern requirements elicitation methods. Prototypes can be constructed to demonstrate either a portion of a system or the whole system in order to get feedback [130]. Prototyping is particularly useful in requirements gathering as the user interface can be replicated with the appearance of the final system but with limited functionality [131]. It allows users and developers to gain knowledge about the system and the way it will work. There are a number of different types of prototype which Bray categorises in [131] as

- Definitive – forms part of the requirements specification or the definition of the required behaviour
- Structural – to check possible design solutions such as performance, but can be used to ascertain feasibility in the earlier stages of the lifecycle.
- Evolutionary – means that development occurs by refining earlier prototype versions of the system. Sommerville [132]outlines some of the problems with this approach when used for large systems such as management, maintenance and contractual problems.
- Exploratory - that will aid in eliciting or refining requirements and are defined as 'throwaway' by [132]

For the purposes of WP5 two different prototypes will be used the exploratory prototype which will allow for the creation of an initial user interface that will allow further definition of the requirements. From this approach, a full collection will be made of all the requirements prior to the outline of a definitive prototype which will form the requirements specification.

The VIDE GUI is aimed at providing an IDE that supports model driven development (MDD). The specification and (partial) development of the exploratory prototype of the VIDE GUI was undertaken to elicit feedback regarding desirable features of a MDD environment. Various meetings and presentations to the VIDE consortium, provided feedback on several most important aspects of the GUI and needing further consideration. This chapter outlines the requirements that the first prototype set out to address, and discusses the limitations of the initial GUI specification and prototype.

6.2 Additional identified requirements

Additional requirements for the first prototype are outlined below. These were obtained during developer focus group meetings and VIDE meetings throughout the project.

An important requirement is for the VIDE GUI to enable business users to write descriptions regarding the activities they undertake, and their various responsibilities:

Level All	Users: All	**MUST**
GUI REQ ID 75	Support for construction of textual descriptions of the domain.	
Related requirement IDs: 12, 14, 26, 27, 28, 29, 32, 33, 58, 59, 62, 63, 66, 75, 76		

The VIDE GUI should provide an editor with a flexible notation where business users can construct domain entities (e.g., roles, activities) and the associations among those entities without being constrained by rules of any modelling notation:

Level All	Users: All	**SHOULD**
GUI REQ ID 76	Provide an accessible domain modelling editor.	
Related requirement IDs: 12, 14, 26, 27, 28, 29, 32, 33, 58, 59, 62, 63, 66, 75, 76		

A means to construct activity models as behaviour models of a system should be supported by the VIDE GUI. These models will depict interactions among instances of the class design model.

Level PIM	Users: All	**SHOULD**
GUI REQ ID 77	Support for construction of activity models	
Related requirement IDs: 12, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 31, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 61, 77		

The VIDE IDE is required to support the building of a domain model depicting relationships among entities derived from the domain descriptions.

Level PIM	Users: All	**SHOULD**
GUI REQ ID 78	Support for construction of domain models based on the domain descriptions.	
Related requirement IDs: 12,19, 25, 27, 34, 47, 48, 58, 59, 60, 61, 63, 75, 76, 78		

The VIDE GUI is to enable traceability of domain entities to the parts of the description where the entities are derived from:

Level PIM	Users: All	**MUST**
GUI REQ ID 79	Support for traceability of domain models back to domain descriptions.	
Related requirement IDs: 8, 9, 10, 15, 34a, 49, 50, 51, 58, 67, 72, 73, 74, 79, 80, 81, 83		

The VIDE IDE is to provide a means to trace elements of a class model (e.g., attributes, methods, or even classes) back to the domain model element associated with it.

Level PIM	Users: All	SHOULD
GUI REQ ID 80	Support for traceability of class models back to domain models.	
Related requirement IDs: 8, 9, 10, 15, 34a, 49, 50, 51, 58, 67, 72, 73, 74, 79, 80, 81, 83		

Again, the VIDE IDE should provide a means to trace elements of a behaviour model back to the counterpart structural model:

Level PIM	Users: All	**SHOULD**
GUI REQ ID 81	Support for traceability of activity models back to class models.	
Related requirement IDs: 8, 9, 10, 15, 34a, 49, 50, 51, 58, 67, 72, 73, 74, 79, 80, 81, 83		

A means for obtaining class design model elements from an activity model should be provided by the VIDE IDE.

Level PIM	Users: All	**SHOULD**
GUI REQ ID 82	Support for direct derivation of class elements from activity model elements.	
Related requirement IDs: 8, 9, 10, 15, 34a, 49, 50, 51, 58, 67, 72, 73, 74, 79, 80, 81, 83		

The VIDE GUI is to support direct construction of a class model based on information available in the domain description. Such information could be directly used to build a class, or form part of a class by providing attributes or methods:

Level PIM	Users: All	**SHOULD**
GUI REQ ID 83	Support for direct derivation of class design elements from domain descriptions.	
Related requirement IDs: 8, 9, 10, 15, 34a, 49, 50, 51, 58, 67, 72, 73, 74, 79, 80, 81, 83		

There should be an interface to other components such as the VCLL component. This to ensure components separately developed to automate CIM development using the VCLL language can be integrated with VIDE.

Level PIM	Users: All	**SHOULD**
GUI REQ ID 84	Provide interface with VCLL component.	
Related requirement IDs: 6, 68, 84		

6.3 VIDE GUI overview

The VIDE toolset is aimed at providing support for MDA development activities. For example, parts of the IDE will support the construction of CIM models; while other aspects of the VIDE GUI will provide PIM development support. Given that these MDA development activities may often involve elicitation and analysis of domain information, the VIDE GUI also provides supports for the building of descriptions about the problem domain, and the analysis of such information. Hence, a user may consider these various development activities as different views for developing an MDA-type application using the VIDE GUI. Figure 27 shows these views put together in a VIDE IDE:

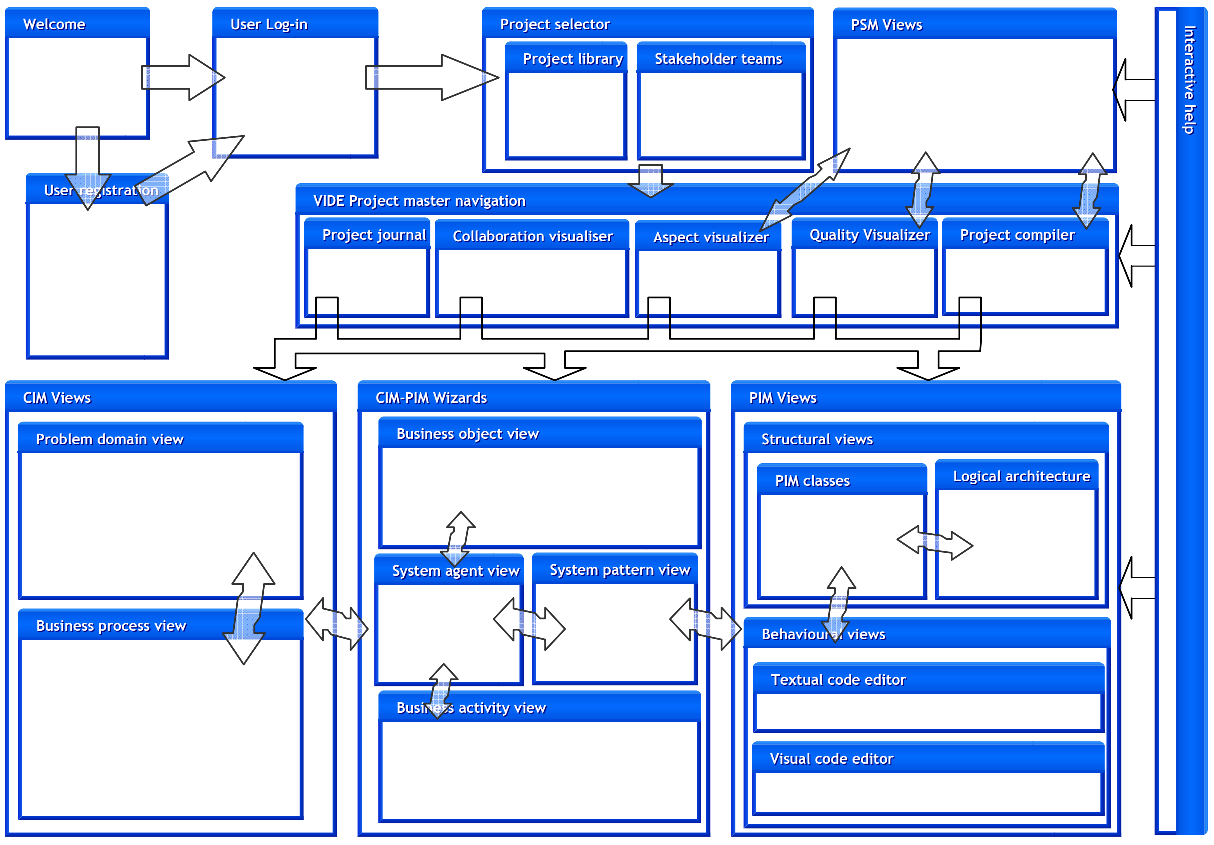


Figure 27: Overview of VIDE GUI

The components of the VIDE GUI are arranged in such as a way as to address a number of significant concerns for a give development project:

- The involvement of non-technical stakeholder in some of the development process
- Elicitation of problem domain information and analysis of such information
- The use of analysis results to build a CIM model
- Development of PIM models from the CIM model

In the following sections, we outline some of the high-level processes that we expect VIDE stakeholders to go through whilst using the VIDE IDE. These act as more detailed specifications of the relevant GUI components presented in Figure 27.

6.4 VIDE GUI Stakeholder interaction processes

6.4.1 Documenting and analysing the problem domain

Software development (whether by following MDA or the traditional) is an activity aimed at producing a software system that addresses a specific issue within a given application domain. A means for eliciting and recording informal facts about the problem domain, and the subsequent analysis of such facts is often an important activity prior to any design. Within MDA, such activities constitute an important step of informing development participants especially on the development of the CIM model. We capture early lifecycle activities in the following RAD model:

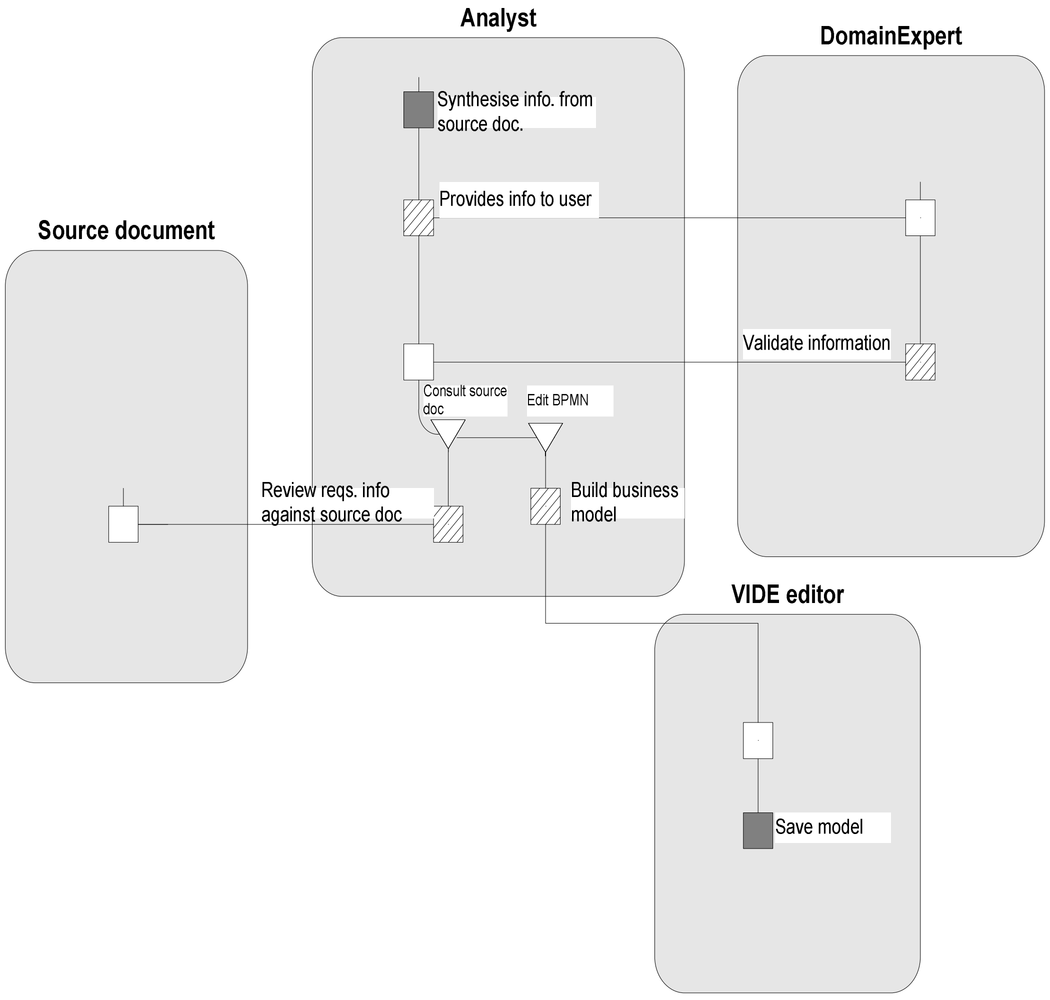


Figure 28: Problem domain construction process

The diagram in Figure 28 indicates that during the elicitation of business related information, one may have a document containing the elicited information. Several documents may be produced depending on the size of the problem. Information from such documents is then synthesised to identify key information that may be relevant the building of a business process model. Such an activity may highlight areas requiring further analysis.

6.4.2 Activities for building CIM models

One of the main MDA development activities is the construction of a formal model of the business process, the CIM model. Some MDA researchers have recommended the use of automatic layout functions within MDA tools to organise CIM model elements for users. We capture the activities for developing a CIM model with the VIDE IDE in the RAD shown in Figure 29:

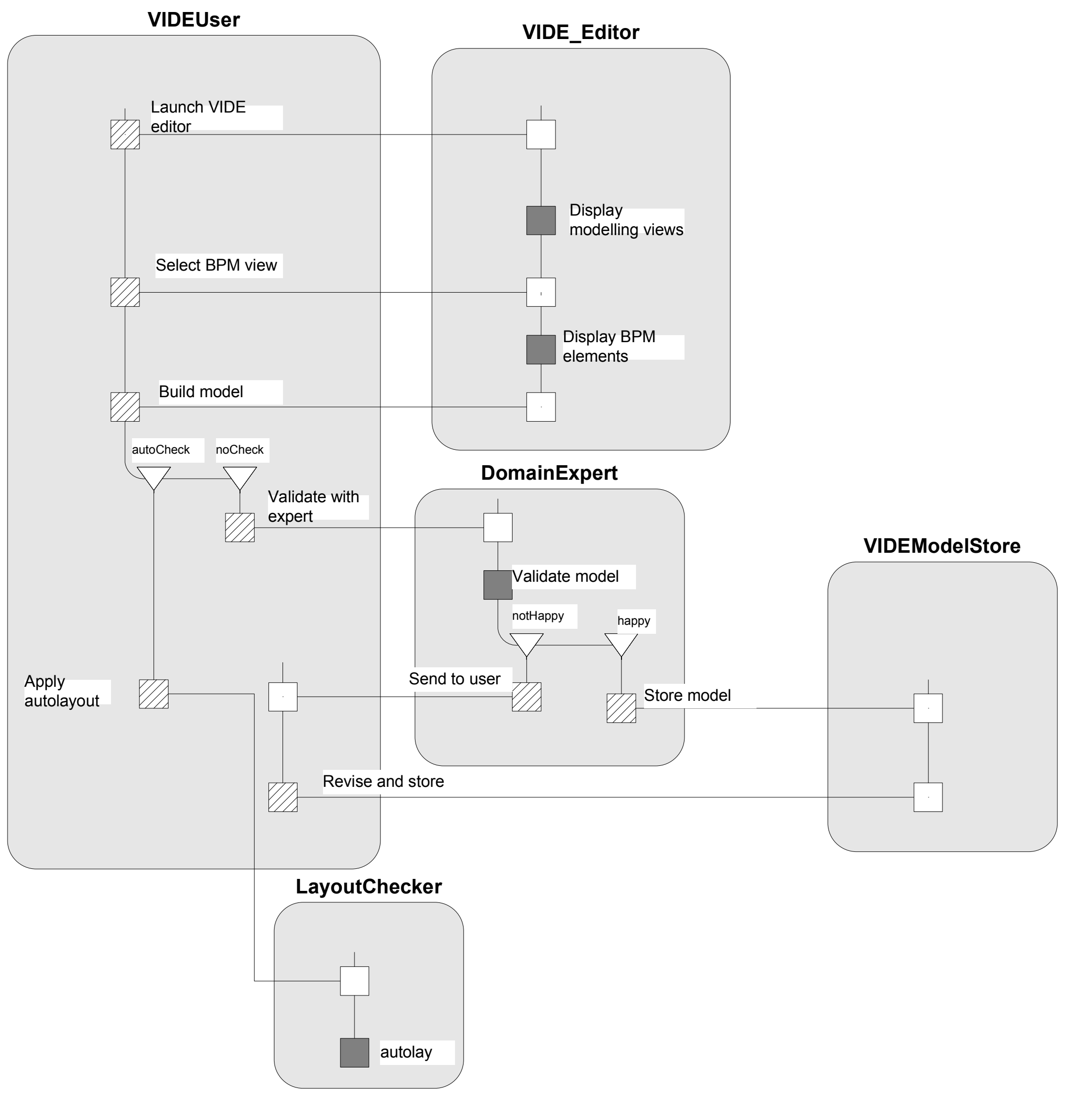


Figure 29: Business process construction process

In Figure 29 a user launches the VIDE editor, and selects the BPM view. This view will display the elements (e.g., VCLL elements, or BPMN elements) for building a business process model. Upon building the model, a user may optionally apply autolayout checking. It is also important models are validated with domain experts.

6.5 The VIDE GUI

6.5.1 Introduction

This section discusses the different views of the VIDE GUI. These views can be categorised into four strands. In the first strand, the views pertaining to user registration, login, and the management of the artefacts of a development project are shown in Section 6.5.2.

The final three strands make up the model refining process. The second strand comprises the various views that show the development of business level models, their display, and how they relate and are described and discussed in Section 6.5.3.The next section describes and discusses moving from a CIM model toward a class design model by using intermediate models of the system behaviour and system services (see Section 6.5.4). Finally the last views that detail the final move to a first cut class model are described and discussed in Section 6.5.5. The model process screens are shown in Figure 30.

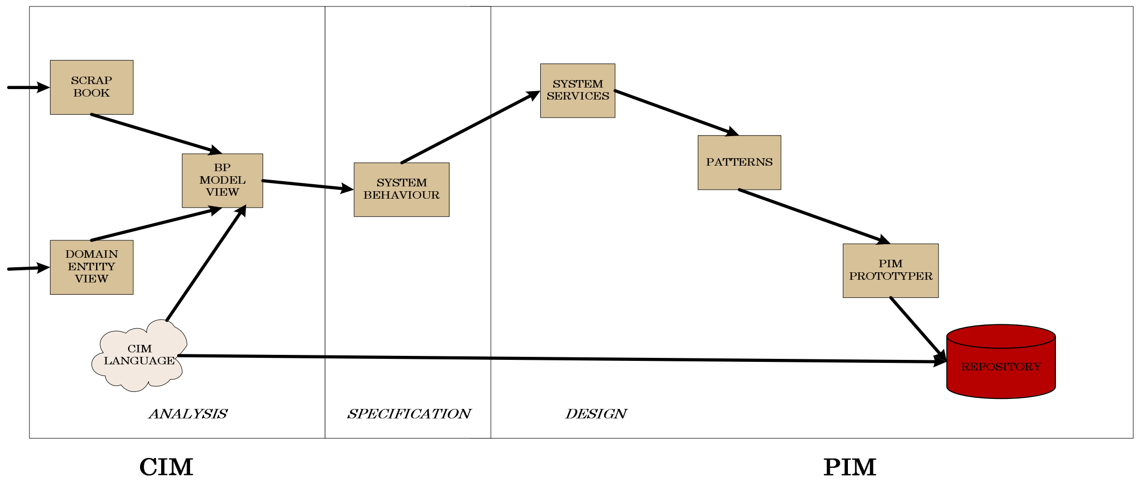


Figure 30: Exploratory prototype process

6.5.2 User login and artefact management

6.5.2.1 VIDE GUI design: Welcome

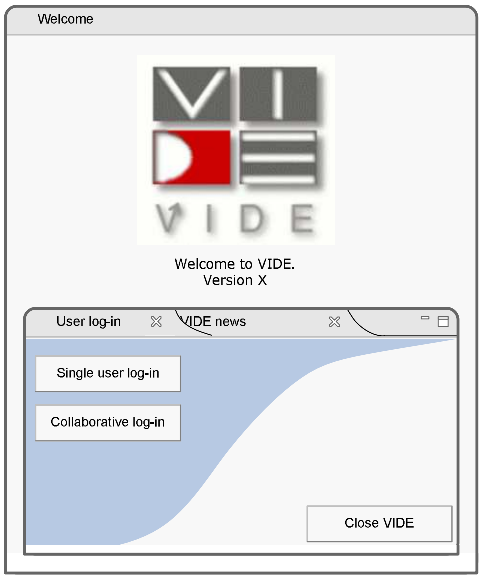
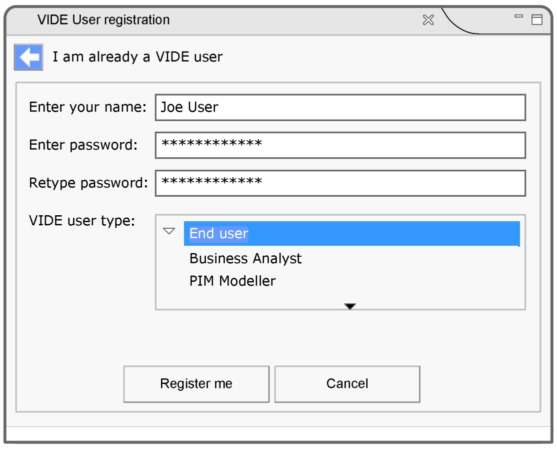


Figure 31: VIDE Welcome dialogue

Upon starting the VIDE environment, a welcome dialogue appears (see Figure 31) that displays appropriate IDE version information and optionally, any news that might be available. The VIDE allows collaboration among multiple development stakeholders. In this prototype specification the model for collaborating users is simple. If more than one user wishes to work on a VIDE project we assume that either a) they have controlled access to the VIDE project artefacts which are associated with their use or b) they are all working at a single development machine and nominate a principal user to lead design or programming activities. In either respect it is important for the purposes of traceability and maintenance to record contributions at any MDA level by users working on a particular project.

6.5.2.2 VIDE GUI design: User registration

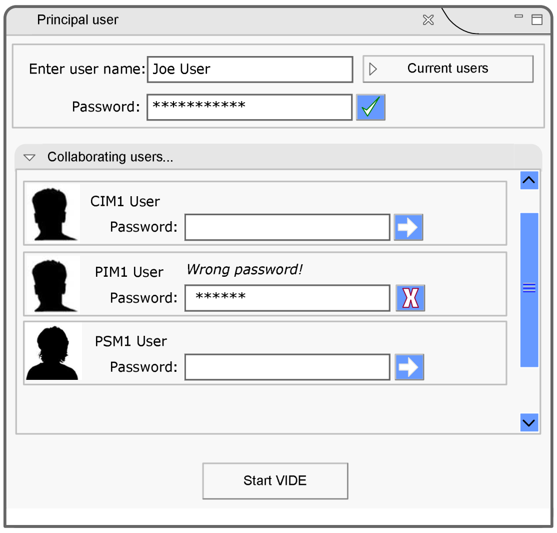


Enter your name: Joe User
Enter password: ************
Retype password: ************
VIDE user type: End user / Business Analyst / PIM Modeller

Figure 32: VIDE User registration dialogue

At the start of a VIDE project, it is reasonable to expect that some stakeholders will not be known to the VIDE IDE. It should therefore be possible to allow these individuals to register themselves with the system, and in doing so, provide additional information that will more clearly specify their contributions (and specific VIDE interaction needs) within a VIDE project. Some of the basic information gathered is depicted in Figure 32.

6.5.2.3 VIDE GUI design: User log-in



Enter user name: Joe User
Password: ***********

CIM1 User
Password:

PIM1 User
Wrong password!
Password: ******

PSM1 User
Password:

Figure 33: VIDE User Log-in dialogue

Given that all users that wish to participate in a VIDE session are registered, it should be possible to specify who has logged in. In Figure 33, one or more users are given the option to log-in; if a single-user log-in was specified at the welcome screen the collaborating users window should be minimised by default. Once all users have successfully entered their VIDE passwords, clicking ‘Start VIDE’ starts the IDE proper.

6.5.2.4 VIDE GUI design: Project selector

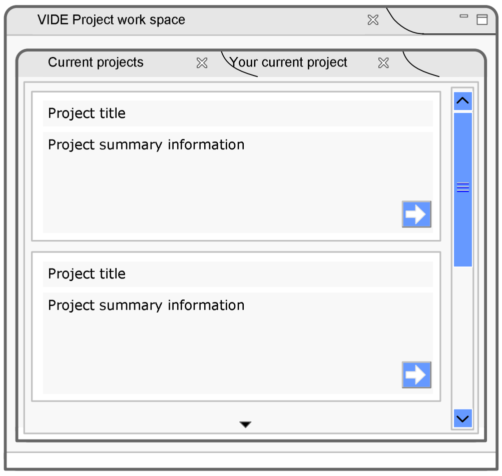


Figure 34: VIDE Project selector

We should expect that VIDE users are likely to work on a number of different projects, for this reason it should be possible to browse through existing projects (possibly contained within an Eclipse workspace). In addition to viewing projects by name, it may also be useful to be given some summary information such as the project instantiation date; number of contributors and so on. For a rapid start, it may also be useful to either allow quick access to the project last opened by the user (see Figure 34 for an example of this arrangement).

6.5.2.5 VIDEGUI design: Project journal

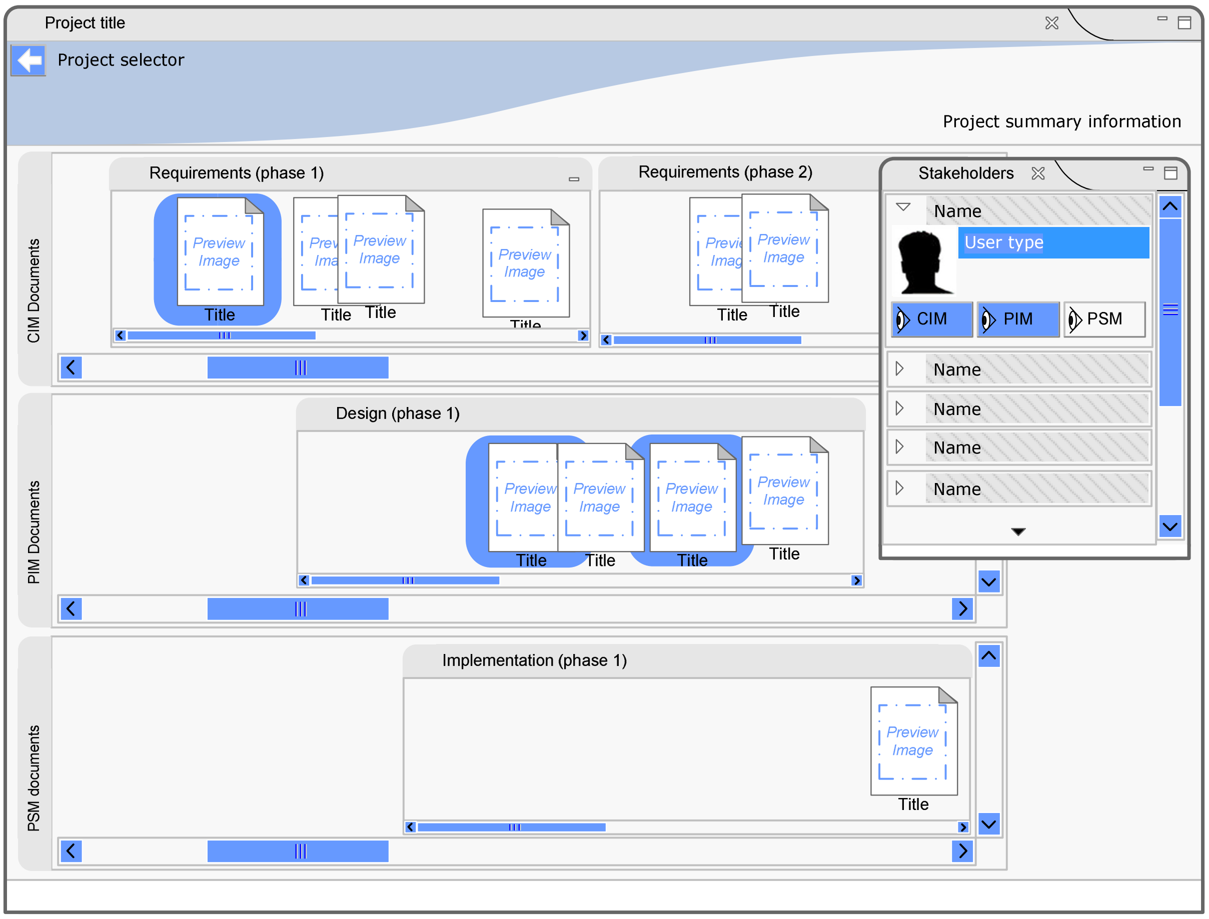


Figure 35: VIDE Project journal

The project journal component (see Figure 35) is an extension to the multi-user support offered in the VIDE IDE. Specifically, the journal is used to represent MDA design and development phases within a project's life time. A principal component of the VIDE environment, the project journal allows users to create development phases within the MDA framework and populate them with CIM, PIM or PSM documents. In addition to this, the project journal should allow one or more users associated with a particular project to be selected and their specific contributions highlighted (this could be further refined with CIM/PIM/PSM filters for each user).

6.5.2.6 VIDE GUI design: Collaboration visualizer

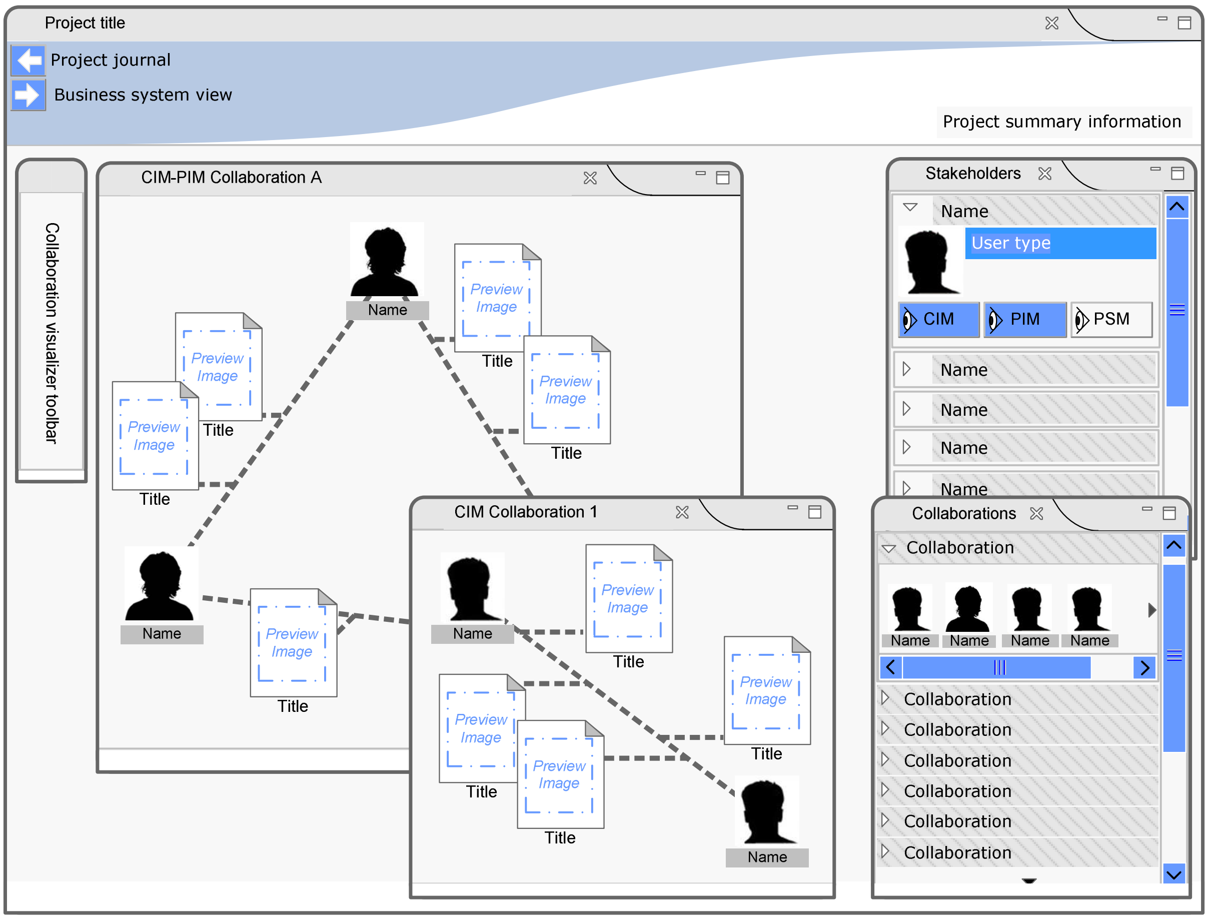


Figure 36: VIDE Collaboration visualizer

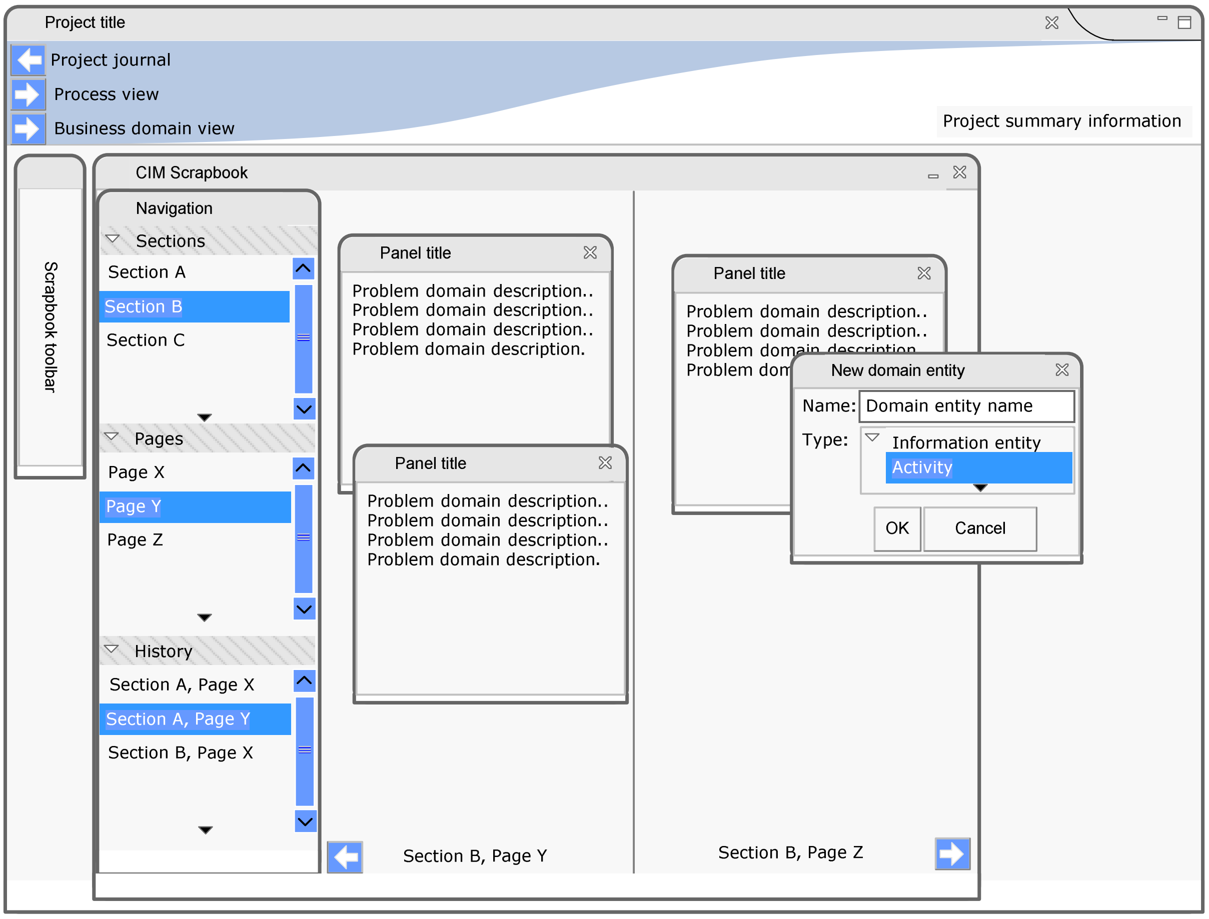
The collaboration visualizer (see Figure 36) provides an orthogonal view on VIDE stakeholders' interactions within a project. Here, particular collaborations are explicitly named and created (within each collaboration, at least two users should be specified, although there could be any number). Shared work between collaborators is represented along dotted arcs, over which documents (from any MDA level) are presented. Again, specialised forms of collaboration can be specified by selecting one or more MDA levels for each user.

6.5.3 VIDE GUI views for business model development

This section discusses model development capabilities of the VIDE GUI. The section presents the process of writing unstructured descriptions about the problem domain, and the construction of informal models to further clarify problem domain concerns.

6.5.3.1 VIDE GUI design: Scrapbook view

The scrapbook metaphor is used to depict the informal and unstructured nature of information at this stage of development. Such information may be used to identify specific business related constructs such as activities, roles, or data items.



Problem domain description..
Problem domain description..
Problem domain description..
Problem domain description.

Name: Domain entity name
Type: Activity

Figure 37: VIDE GUI scrapbook view

In our assessment of many MDA-based software development tools, we found that many did not provide support for the production of domain descriptions or the building of domain analysis models. The VIDE scrapbook view (see Figure 37) provides some basic features for capturing, analysing and 'marking up' parts of an informal problem domain description. Here, users are able to open domain descriptions (either as text, or possibly images) and annotate parts of them for future reference. Each annotation should include a name and a type description based on generic concepts derived from business activity or process models.

6.5.3.2 GUI design: Business domain view

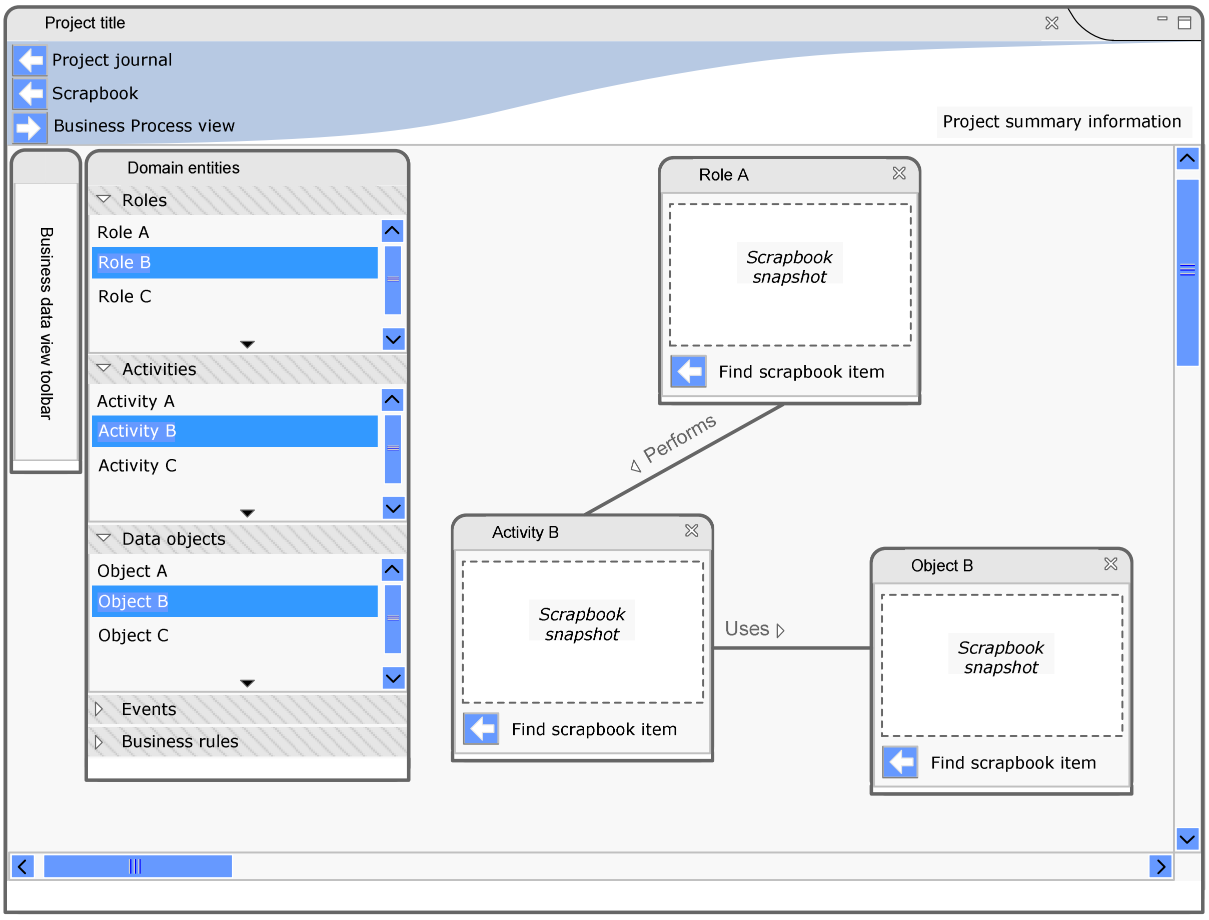


Figure 38: VIDE GUI Business domain view

A further step from the writing of domain descriptions is the identification of various constructs from such descriptions in order to create an analysis model like the one shown in Figure 38. Such models will typically show relationships among roles and associated activities, and any used or produced data items.

This view of business domain artefacts is primarily for managing entities, such that a high-level view of the problem can be generated before a business process model that explicitly refers to these entities, is constructed. Note that it may be possible for VIDE users to move from the scrapbook view to the business process model without referring to this intermediate step.

6.5.3.3 GUI design: Business process view

During business process modelling, a user might build the business process model using activities or roles already identified during the analysis of the information available in the scrapbook. Figure 39 shows the organisation of such information, and the use of formal concepts such as swim lanes (from business process modelling) in constructing a CIM model.

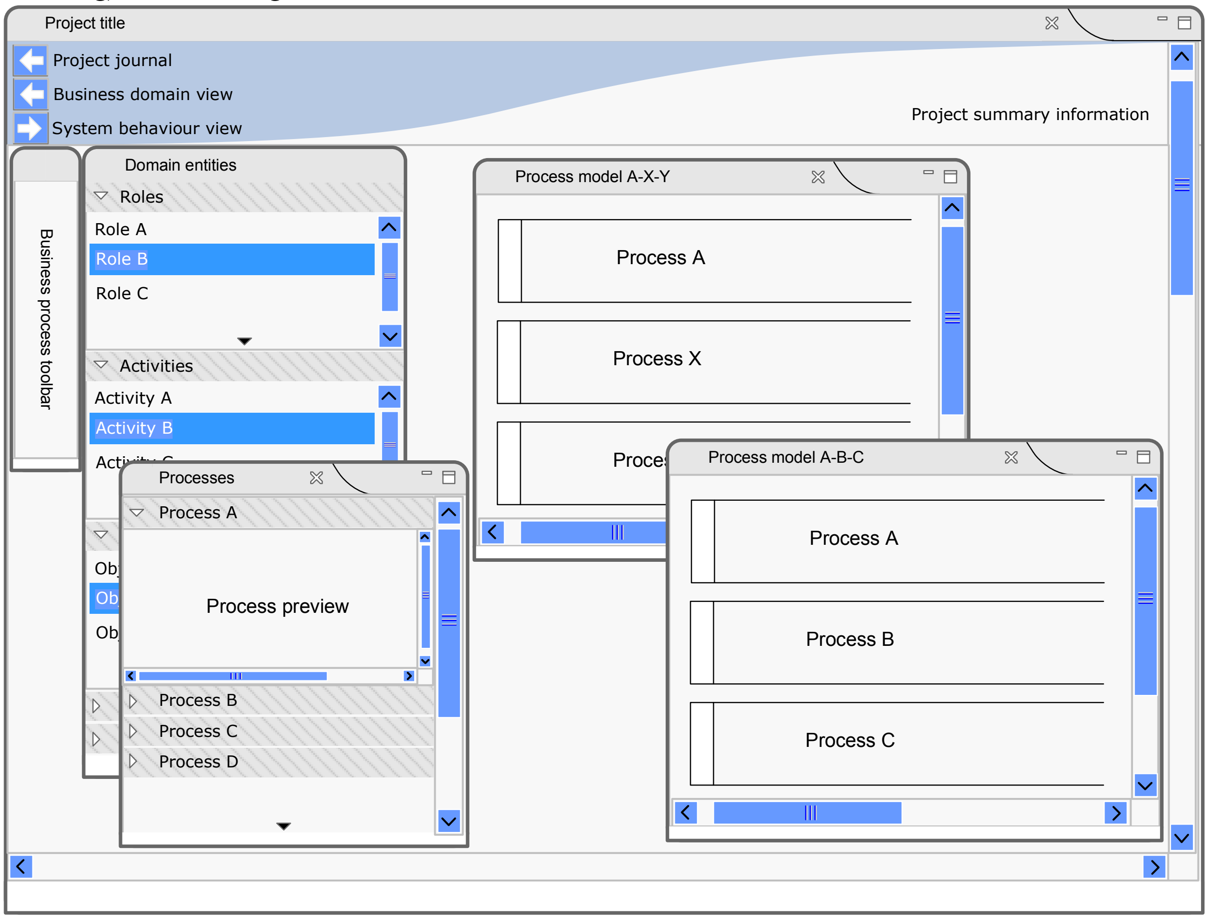


Figure 39: VIDE GUI business process view

One aspect of CIM level support within the VIDE IDE will be the provision of a business process modelling support. The business process view (see Figure 39) allows users to construct business process models within which there may be many named processes – these are additionally accessible via a process list. It is expected that business domain artefacts (such as data objects, activities and events) have been derived from the scrapbook view and should be used as first class inputs for the business process modelling task; however these may be independently generated without a direct reference to a business domain entity.

6.5.4 GUI design: CIM-PIM early transformation views

The movement from CIM to PIM requires some kind of transformation between the CIM models to one or more PIM models. There are recognised challenges of fully automated transformations owing to syntactic mappings and semantic mappings between the views. This section proposes moving from a CIM model to a class design model by using intermediate models of the system behaviour and system services.

6.5.4.1 VIDE GUI design: System behaviour view

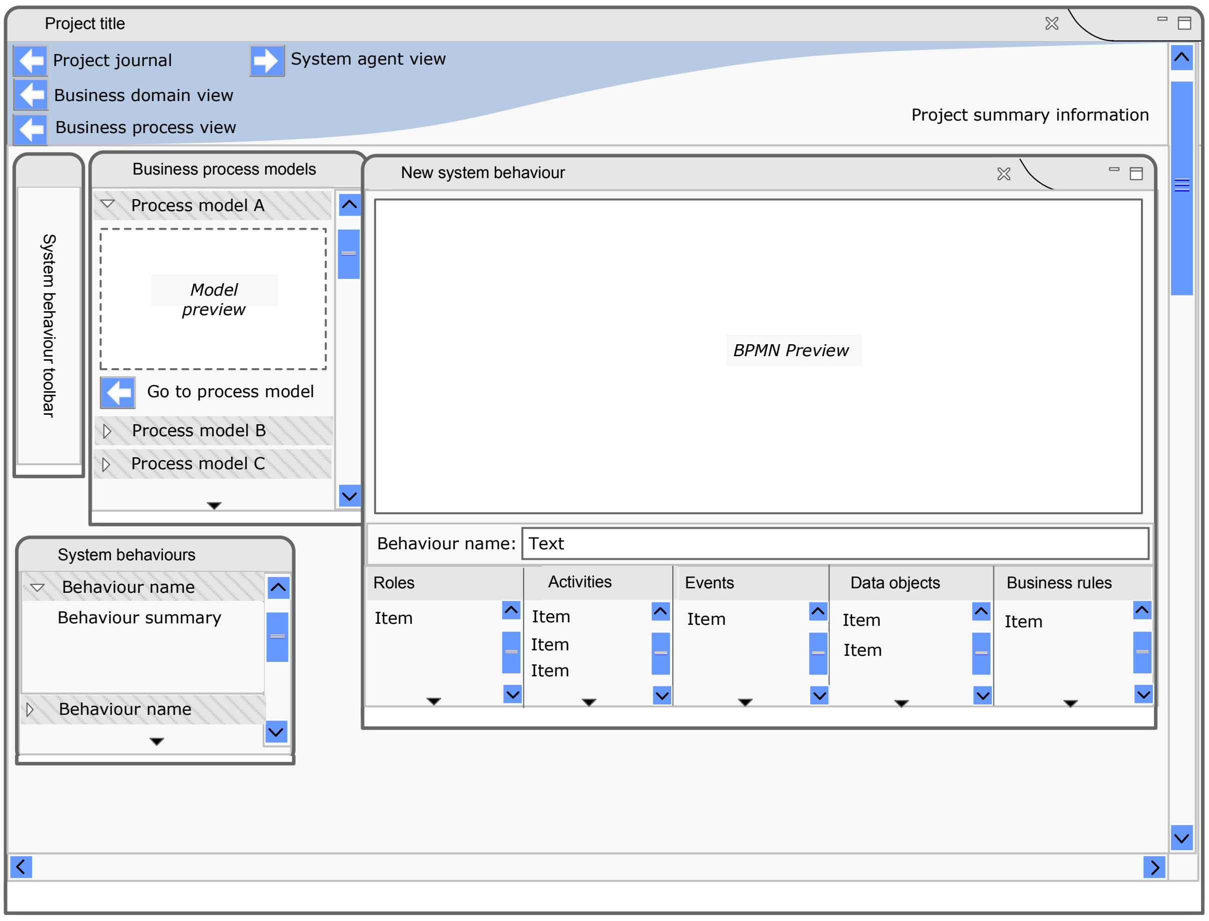


Figure 40: VIDE GUI System behaviour view

The first step toward the transformation between CIM and PIM level model is to identify entities from one or more business process models that are likely to be performed by the target system. In other words, appropriate stakeholders from the VIDE project team would be able to select entities such as roles, activities, events, data objects and business rules that are likely to be associated with some functional aspect of the system. At this stage, the user is invited to navigate through existing process models so that they are able to select elements that can be associated with a new (or existing) system behaviour definition. This is achieved by first creating a new system behaviour, and then dragging and dropping elements from the business process preview window into the behavioural overview. Once finished, it should then be possible to assemble these behaviours into more abstract collections of system functionality - system agents.

6.5.4.2 GUI design: System agent view

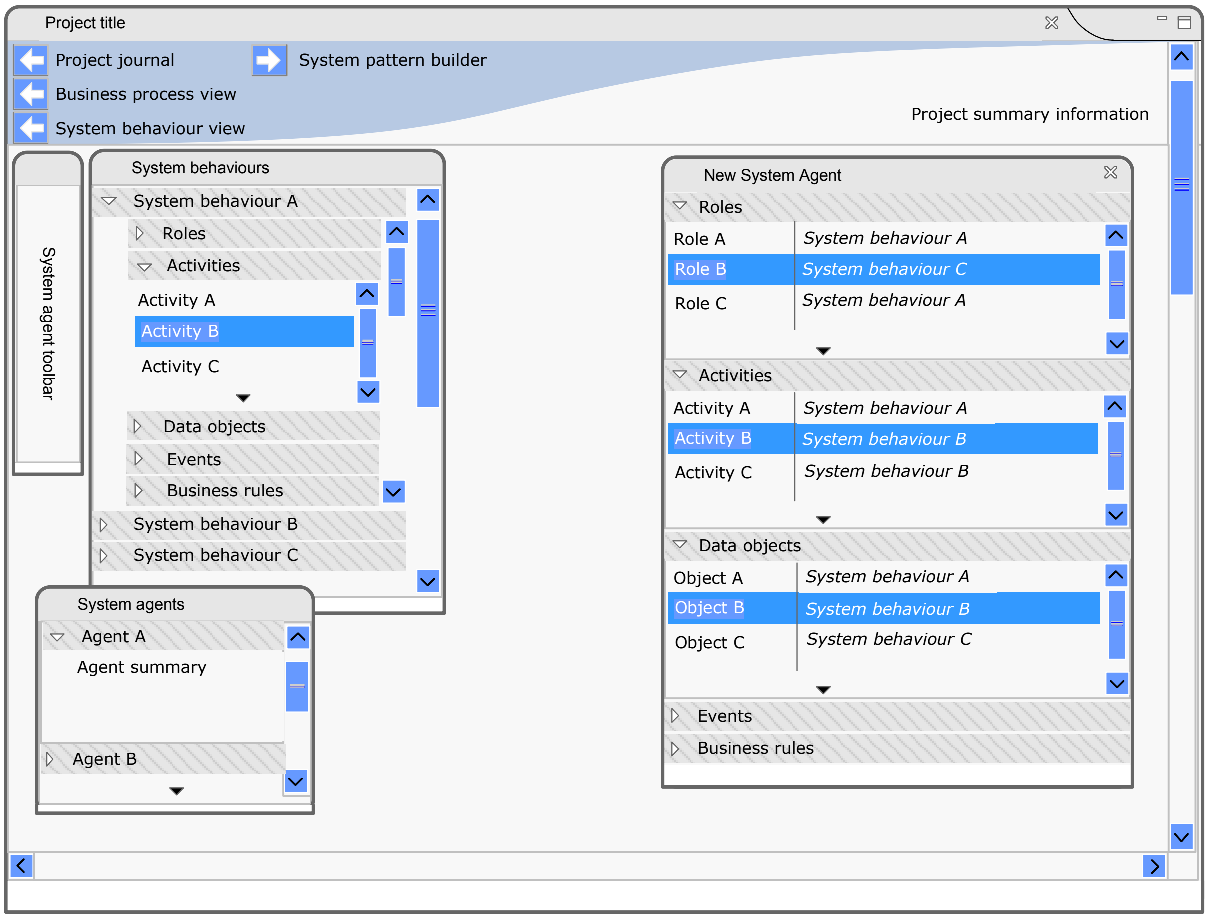


Figure 41: VIDE GUI System agent view

The system agent view, as shown in Figure 41, progresses the process of CIM to PIM transformation. A system agent should be considered as a software-based entity that, given a specific context, knows about certain types of information and can act in particular ways (this is expressed in terms of their composite system behaviours .To do this, the user is expected to select aspects from system behaviour descriptions and drag them onto new instances of system agents. Aspects of the system behaviour may include roles, activities, data objects, events and business rules. It is therefore possible (although not necessarily desirable) to mix a variety of previously defined system behaviours with one or more system agents. Agents are subsequently used in VIDE's final transformation stages in the CIM to PIM transformation process.

6.5.5 GUI design: CIM-PIM final transformation views

In this section, we discuss the final steps leading to the production of a first-cut class model of the system.

6.5.5.1 GUI design: System pattern builder

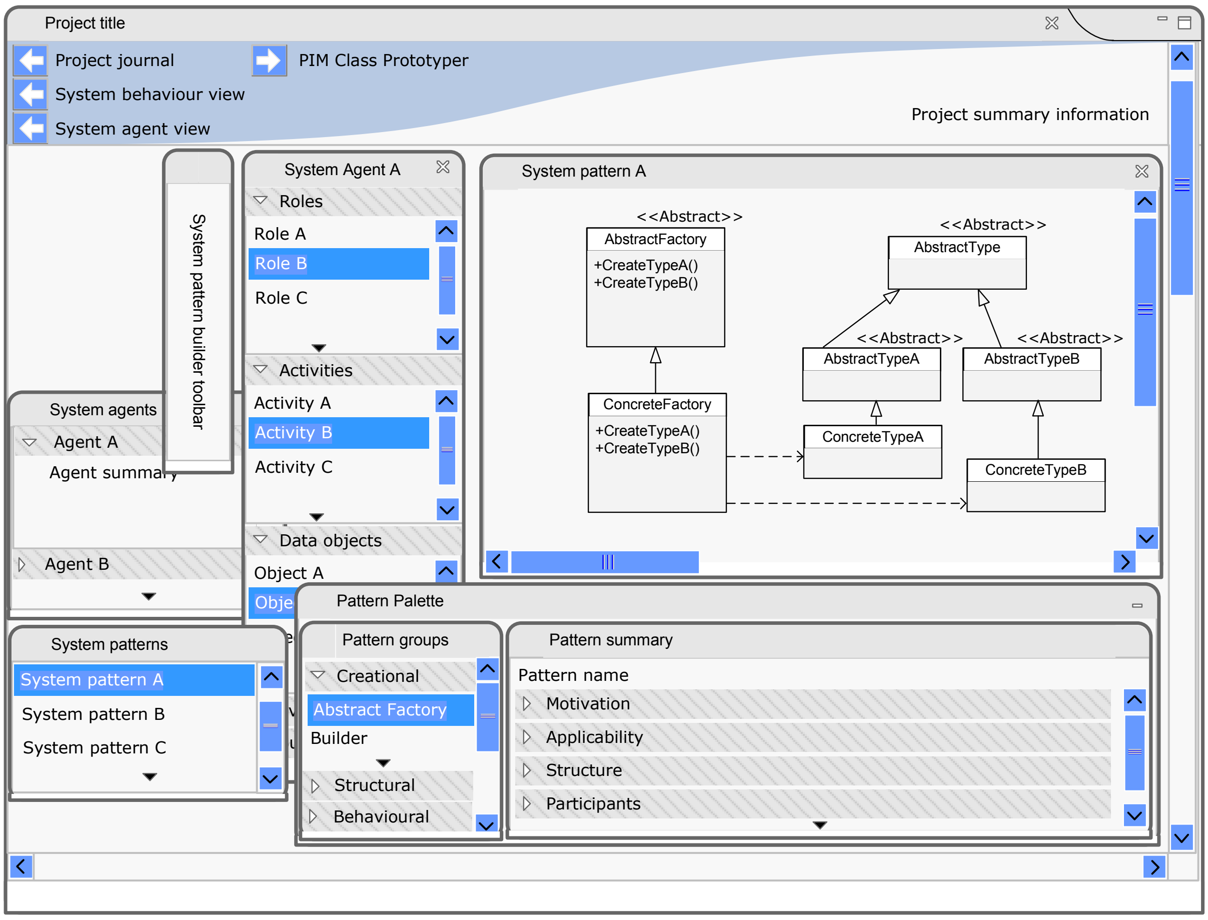
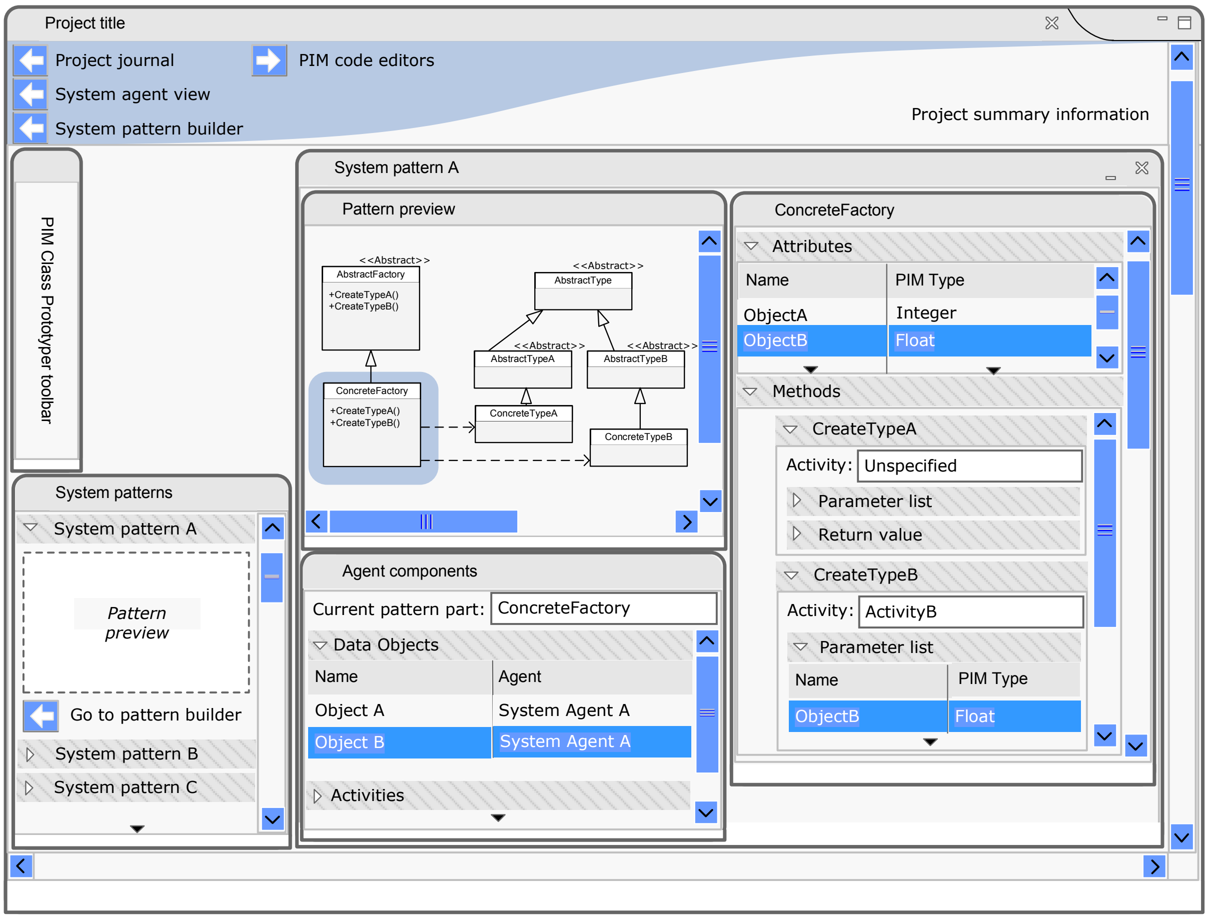


Figure 42: VIDE GUI System pattern builder

The system pattern builder represents a significant step for the transition between CIM level concepts and the PIM level class model. A mapping between a high level specification of the target system's agency, encapsulated through system agent objects, is started by identifying an agent to be implemented by a software system. The next step is to select from a collection of software patterns made available by the VIDE toolset (these may include user-defined patterns). A standardised description of any pattern is made available to the user; once a new pattern is selected for generation, a template PIM class equivalent is presented to the user. At this stage, users should select aspects of one or more agents that are to be associated with elements of the software pattern (here, these are primarily classes). As with system behaviour specifications, complex and multiple associations may be allowed here. Further refinement of these associations should be made using the PIM class prototyper.

6.5.5.2 GUI design: PIM class prototyper



Current pattern part: ConcreteFactory

Name	Agent
Object A	System Agent A
Object B	System Agent A

Name	PIM Type
ObjectA	Integer
ObjectB	Float

Activity: Unspecified

Activity: ActivityB

Name	PIM Type
ObjectB	Float

Figure 43: VIDE GUI PIM class prototyper

The PIM class prototyper is the final stage of the CIM to PIM transition process. Here, software patterns previously defined in the system pattern builder are further refined. Specifically, data objects carried over from the CIM agent model (displayed in the 'Agent components' window) can be associated with PIM class attributes (and their types); in a similar fashion, activities or events carried over from the CIM agent model can be associated with pattern methods. It is unlikely that all CIM agent data objects, activities and events will easily find a home in the prescribed parts of the chosen software pattern template. For this reason, it may be important to provide some additional, basic prototyping capabilities for the PIM class templates that allow the user to add additional attributes and methods of their own devising such that appropriate CIM parts can be mapped.

Having completed the PIM class templates, it should be possible for the VIDE user to navigate to the PIM code editors. Here, the actual implementation of the methods specified in the PIM class pattern (based on their associated activity and data object models) should be written in VIDE code.

6.6 Summary

This section outlined the final requirements that were used in addition to those collected earlier. It added that Role Activity Diagrams of the process as we understood it of the end-user is using the VIDE environment to complete a task. We then detailed the exploratory prototype in four separate sections. The log-in and artefact management section were explored, and finally the three model building sections from scrapbook to class model we described.

7. Evaluation of exploratory prototype and requirements summary

7.1 Evaluation

The exploratory prototype, as presented in Section 6.5, was also presented at a series of focus group meetings and the feedback was collected. Feedback from presenting the system at two workshops to consortium members was also collected. These issues are outlined below together with our developer evaluation using the Cognitive Dimensions Framework. In addition, pertinent requirements obtained from the research carried out over the year are discussed. The feedback, comments and research allowed some paper prototypes to be created. Paper prototypes add value because they are 'hands on' and enable critical feedback [133]. From this collected information the system will be re-modelled and a definitive prototype will be created synthesising the lessons learned from these various forms of evaluation and .research.

7.2 Issues

7.2.1 High level summary

The exploratory prototype was very useful in identifying issues. It is possible for system designers to get too close to the designed system and be unable to see such issues. The addition of feedback from the focus groups and presentations allows the developer to take a step back and to view the system from others' points of view.

Overall, the system was well received, but it became apparent, when issues were identified, that it did not appear to be sufficiently accessible to business users and was generally inflexible in usage. By following the screens through, it imposed a particular process which was a little quirky. There was no traceability, from domain information to PIM model and many of the models suffered from fragmentation and loss of information. Finally the system did not take into account behavioural and structural information.

7.2.2 Issues with the initial prototype

- The scrapbook idea was good, but it was too structured and did not allow sufficient free flow of ideas. It did not make use of existing documents and appeared complicated.
- The domain models screen should aid analysis but it was not possible to create informal conceptual ideas and there was no explicit relationship to the scrapbook (the previous development step). It forced the end user/analyst to become too detailed too early in the process (for example by defining roles and objects) and using terminology that could confuse business people and giving no support. It was also not possible to define the links between different items and did not support a process of gradual refinement. Finally the domain model does not scale well (to allow for decomposition).
- Construction of domain models from domain descriptions was inflexible since it only permitted consideration of pairs of entities and produced or consumed data objects.
- A lack of consideration for persistence of models. Given that VIDE is an MDD environment that will interface with other MDD-type tools, persistence issues are important.
- Traceability between domain models and respective descriptions was inadequate since the built entities (e.g., roles, activities) did not bear any correlation with selected (partial) description of the domain.
- The initial prototype had considered transition between process model and design model based on system services and behaviours. Whereas this is a possible route to take, one might not want two modelling phases to achieve what other MDD tools suggest in one step.
- The exploratory prototype based the construction of PIM models on suitably identified (or custom built) patterns. MDD proposes derivation designs from business models without such patterns, hence this may impede the uptake of VIDE.
- There was no clear distinction among elements of a domain model. That is, all elements (roles, activities, and data objects) looked the same, and no stereotype was adopted to qualify distinct model entities. This can be confusing for a large model or an unfamiliar domain.
- There was no way to group together multiple models pertaining to a given development task.
- The business process model and system behaviours screens should allow the creation, editing, import and export of business models but the interfaces were not intuitive. No external interfaces were defined. It was also apparent that there was a loss of information when defining system behaviours; the process allowed the

retention of some structural information and lost most behavioural information which contributed to the fragmentation of models. In addition, as far as the specification was concerned, the system did not deal with how scoping should be handled, nor support the requirements and specification documentation. Once again, traceability of the models, back to the previous document's screens and models, was absent.

- The concept of services and patterns was good, but information that was lost earlier in the process was never replaced and, therefore, a single process model would result in fragments of a PIM model. It became very difficult to define services. To be able to create patterns for the designs needed, extensive pattern knowledge was required along with access to an extensive pattern library. It would also require specialist knowledge to create class diagrams from patterns. There was once more, no traceability. Finally, there was no behavioural model carried through the process.

7.2.3 Evaluation using Cognitive Dimensions

7.2.3.1 The Cognitive Dimensions used for the evaluation

As described in section 5.1 only six cognitive dimensions will be used for this evaluation. They are the most relevant for the type of activities pertinent to software modelling. These six dimensions are used in a similar way by Blackwell in [91]. We briefly repeat the description of the dimensions here for completeness:

- Viscosity –the ease with which users can make changes to models.

- Visibility – we assess whether the prototype provides users with 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 – we assess whether the tool provides 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 Business Process model. An important side to this dimension is whether or not one can derive parts of a given model from another directly, including traceability of information between models.
- Hidden dependencies – we assess whether there are any hidden dependencies between components.
- Premature commitment – we assess the extent to which the prototype requires premature commitment during modelling on the part of the users.
- Secondary notation – we assess whether the prototype provides secondary notation to provide extra information about models.

7.2.3.2 The issues:

As indicated above, the interesting point of the scrapbook concept is to afford non-technical users flexibility and accessibility for creating models based on their understanding of the problem domain. This flexibility was curtailed in the first prototype because entities derived from the scrapbook could only be associated in three ways. That is, one could only obtain and classify elements from the scrap book as roles, activities or data objects. Furthermore, associations among these elements did not allow for roles to be associated with more than one activity, or for activities or roles to be associated with more than one data object. This constraint is akin to the cognitive dimension known as **premature commitment**, since a modeller is "forced" to prematurely decide on any of the three elements to depict in a given model fragment. The exploratory prototype provided a means to create and label associations among roles, activities and data objects, but deletion of a link, or an activity in order to add a different activity or link caused the modeller to have to often rebuild the entire model, or significantly move other model fragments that were within close real estate space of the one altered. This, again is akin to the cognitive dimension of knock-on **viscosity** because amendments of given parts of a model "forced" significant changes to associated models. An important aspect of the early stages of software development (e.g., domain analysis, requirements determination) is the need to provide a means to describe a rich picture of the problem being addressed. For example, associations between roles and activities in a domain model may subsume further information about the problem. It may be that such associations need further description, or modelling with a different notation to enrich the parent model. The exploratory prototype did not provide any secondary notation for augmenting standard models of the prototype.

The design of the exploratory prototype had a key requirement of maintaining traceability among different models. For example, where activities in a CIM model are derived from activities in the domain model, such corresponding activities are to be highlighted in both models. However, to be able to move from a domain model to a CIM model, one needs to display the domain model side by side with the CIM model, or its editor window. This issue is akin to the cognitive dimension of **juxtaposition**, and the initial prototype could not juxtapose different models.

An issue with design models is the extent to which several intermediate models were involved in deriving a first cut design model. For example, the concept of system behaviours, system services and design patterns meant that appearance of various class design sections restricted the actual class model to a very small screen estate, hence reducing the **visibility** of the design model. It was rather confusing as to whether the design model was dependent mainly on the identification of a suitable design pattern, or the ability to define appropriate system services and behaviours, yet again, faring inappropriately against the cognitive dimension of **hidden dependencies** among models or model components.

7.3 Requirements summary

This section will draw together the major requirements from earlier work and the exploratory prototype that will be carried forward. Some of the ideas were carried forward using paper prototyping which is useful for exploring ideas informally.

7.3.1 The scrapbook

The exploratory prototype showed that there were a number of new requirements that the system needed to take into account and these began with the scrapbook. It was felt that the idea of the Scrapbook was good, but the execution of the concept needed some attention. It needed to be more flexible and to allow the free flow of ideas. The screen looked complicated at the moment and should be simpler.

Requirements from the description of work, carried forward in Section 1.2, highlighted the fact that the toolset "should be used by IT specialists and individuals with little or no IT experience". Further requirements from Work Package 1 and outlined in Table 1 REQ- NonFunc1 discuss the requirement for accessibility at the CIM level. Non-technical users working at the CIM level should be able to input, retrieve and understand their business domain descriptions in a notation that is non-technical and accessible.

There are a number of requirements that are useful here from the software visualisation research, such as Section 4.2.1.8, that suggest that the components should be structured and have features to aid navigation and, in Section 4.2.1.11, that the system should be intuitive with regard to navigation and control. Perhaps the most important requirement from this area for the scrapbook is that the user should be able to return to the original information source and link to other views of the same information, as discussed in Section 4.2.1.12. Since users at this level may well be novices, a requirement, from our Visual Programming research in Section 4.3.1, that is important here is that all graphical representations should be relevant because novices will try to make sense of all explicit connections even if they are not relevant. Finally the Natural and Codeless programming research highlight the fact that components of a project should be easy to find and identify.

Additional flexibility can be gained by using a tree structure which facilitates the adding nesting and organising of entries. Entries should allow cross referencing and notes for the user to annotate various entries. The system should allow linking to both existing and external documents and provide a simpler and cleaner, less cluttered interface. The paper prototype of our ideas in the area is shown in Figure 44

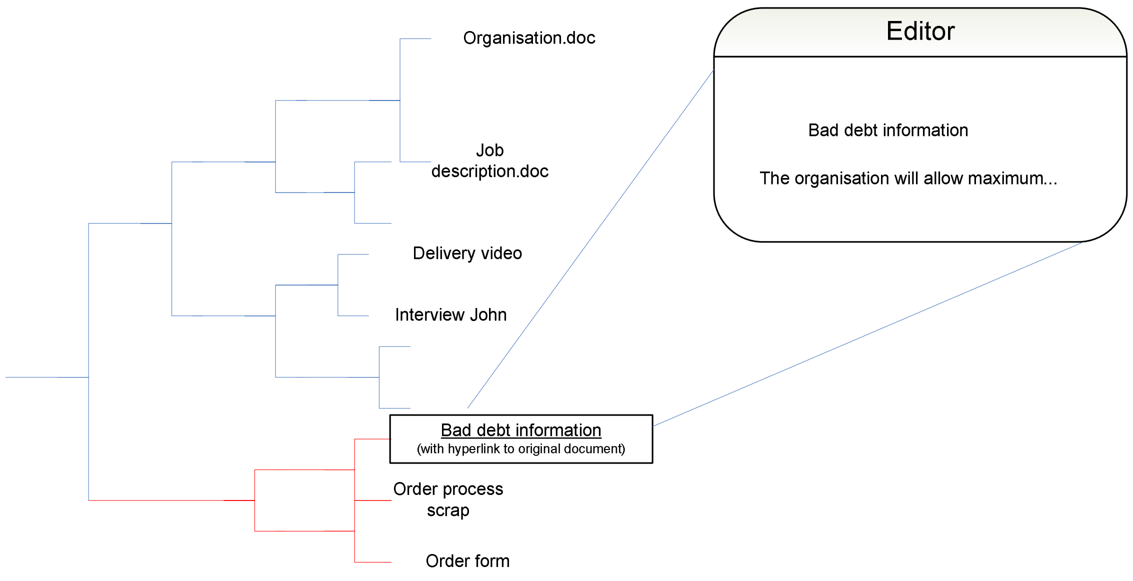


Figure 44: Paper prototype of the scrapbook

7.3.2 Analysis palette

Taking ideas forward from the feedback from the domain model screen of the exploratory prototype it became apparent that there was nothing that would aid the user in handling vague (essentially informal) conceptual ideas and was quite a complex and confusing screen. The Description of Work described in Section 1.2 states that all stages of application development should be more accessible to non-IT professionals. This screen is likely to be used by non-IT professionals, end users and business analysts and should thus answer the requirements in Section 1.2.2 REQ – NonFunc1 the users should be able to understand their domain descriptions in a notation that is non-technical and accessible. NonFunc3 states that the system should provide context sensitive help for users working at the CIM level and this should be explained in non-technical terms. NonFunc4 requires that we use clear and unambiguous notation. NonFunc 5 requires that model views should be user oriented – the scope and content of views should be controllable by the user.

Our research from the area of Software Visualisation is particularly relevant here. Firstly, the domain user is likely to have conceptual ideas that will need to be represented, and over time, these may be developed into some entity that is understood. Different visual components should be distinctive (Section 4.2.1.2), present as much information about the component (Section 4.2.1.3), and should be as simple as possible (Section 4.2.1.4). Perhaps most importantly the user should be able to link to other views of the same piece of information, as defined in Section 4.2.1.13. The user at the Analysis level is also likely to have low levels of IT experience and will have similar requirements to the scrapbook user, in terms of ensuring that graphical representations and explicit connections are relevant. The diagramming research (in Section 4.4.1.) highlights the requirement that modellers should be able to arrange their model themselves and that models should be stored persistently.

Additional requirements from the exploratory prototype feedback suggest a need for provision to help users define concepts, although context sensitive help is mentioned earlier, user concepts are much more specific. It should be possible to define links between items and most importantly the traceability from the scrapbook items and scraps should be maintained. The layout and screen components should support a process of gradual refinement, and whilst the requirement is that users should be able to arrange their own models, the system should allow them to decompose or create a hierarchy view of the model. Ideally the system should not impose a process on the user, rather they should be able to enter or exit the system at any point. Finally, the system should be semantically ‘loose’ to allow it to be used in a more flexible and business friendly way. A paper prototype of the concepts that would respond to these requirements was developed as shown in Figure 45

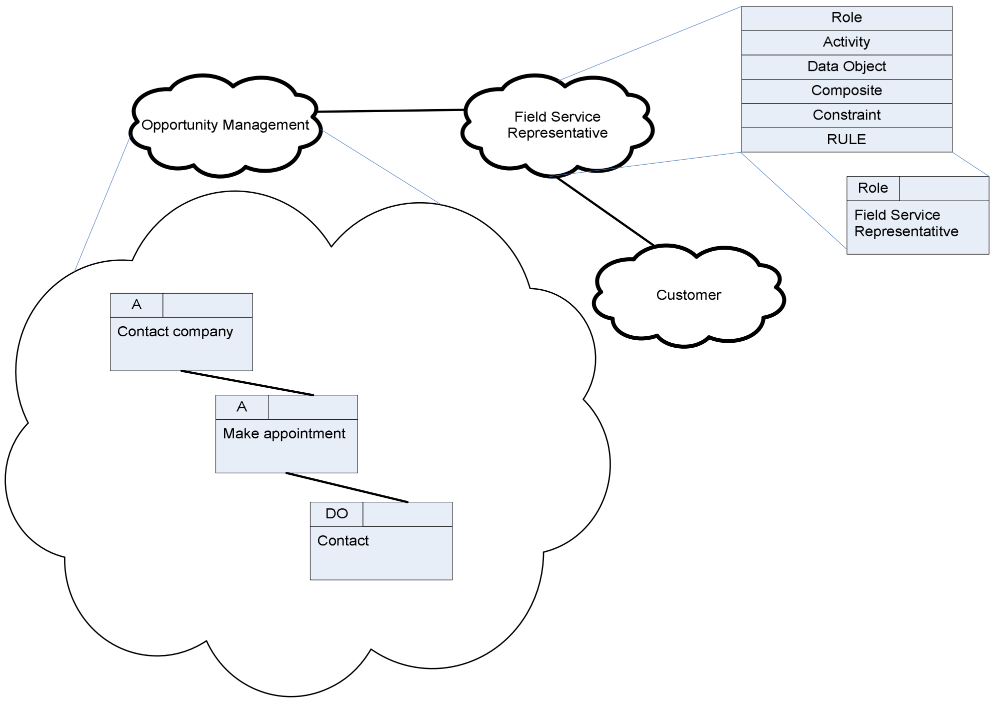


Figure 45: Paper prototype of the analysis palette

7.3.3 CIM Palette

This area was defined in two screens in the exploratory prototype, the business process model and system behaviours screens. This palette will have to be capable of dealing with the BPMN notation, the VIDE CIM level Language (VCLL) and, perhaps, other notations such as Role Activity Diagrams. Requirements from the Description of Work in Section 1.2.1 highlight the fact that VIDE should be an open and interoperable platform and requirements from Work Package 1 in section 1.2.2 REQ - User1 also support this need. However the UML standard should be used where possible as users are sensitive to standards. REQ –User2. To enable these concepts a plug-in mechanism should be considered which is a requirement at REQ -Tool 12.

Feedback from the exploratory prototype has highlighted the requirement for a single view for both the process and system model with an explicit definition of the boundary in a component that supports a process of gradual refinement. It should detail two forms of model in the same tool the business process model and the system process model. Wizards should support the process of requirements and specification integrated with the system model and there should be full traceability where it is available. A component structure will allow for different model types therefore interfaces should be defined for import and export.

A paper prototype of the requirements was developed and is shown in Figure 46

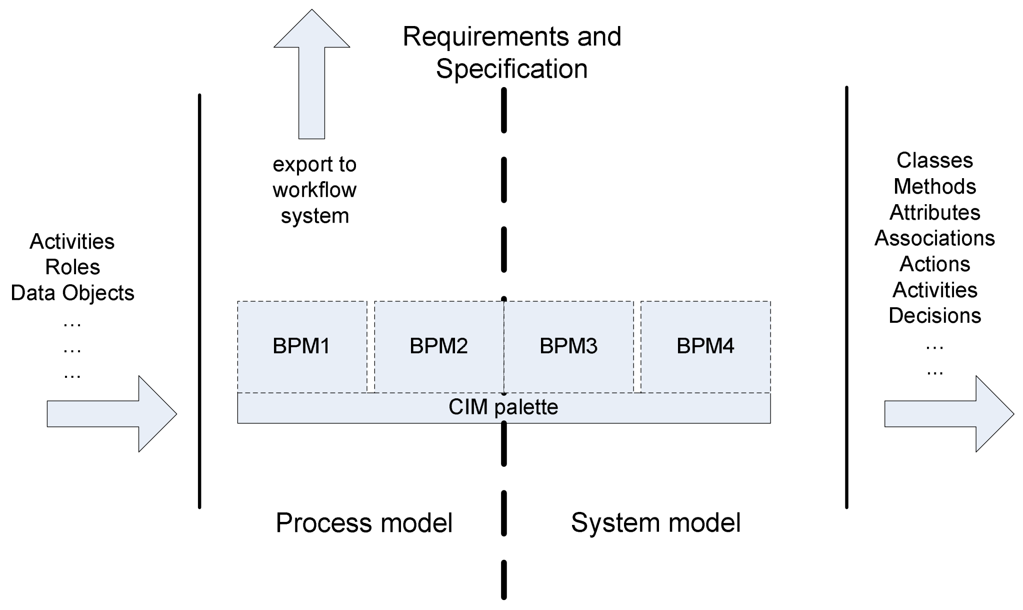


Figure 46: Paper prototype of the CIM palette

7.3.4 Design Palette

The Systems Services screen and the Pattern Editor screen, whilst interesting concepts, were flawed in a number of areas. These were, specifically, the loss of information that was never replaced, and the fragmentation of the process model. This screen needs to take input from the previous component and support the creation of 'first cut' PIM models.

A number of Requirements from Work Package1 outlined in Section 1.2.2 are relevant here. REQ-NonFunc 4 states that the notation should have clear, comprehensible and unambiguous semantics and REQ-User 2 requires the use of the UML standard and in addition REQ-Tool 14 requires VIDE to follow a strictly model driven approach. Research in Software Visualisation highlights the requirements that the amount of information that is visible will increase the complexity discussed in Section 4.2.1.5 and to allow minor changes which will not cause major changes to the system as discussed in Section 4.2.1.9. These are particularly relevant here because there is a considerable amount of information available that needs to be displayed or made available, and in addition the changes that are made should not cause major changes to the environment. The Visual Programming research prompted the requirement for the use of secondary notation where possible to improve comprehension (see Section 4.3). Secondary notation is important when considering a large amount of information, the layout and order of that information will assist in a major way in reducing the complexity. The diagramming research has been carried out in Section 4.4 backs this up by highlighting a number of particularly relevant requirements. From class diagram layout research it should be possible for the user to stipulate either automatic layout or to arrange the models themselves, and with automatic layout compositional elements should be kept close together, and an inherited classpath should be easy to follow. Other requirements relate to allowing modellers to interactively navigate through the model history, and allowing the storage of different model versions. This will increase the comprehension of the models and hence reduce complexity.

7.4 Requirements summary

The requirements from D1 and all the additional requirements have been collected in a table which is shown at

Requirement	Name	Priority	Cross Reference
REQ – NonFunc1	Accessibility at the CIM level - the VIDE environment should provide non-technical, business domain descriptions. Non-technical users working at the CIM level should be able to input, retrieve and understand their business domain descriptions in a notation that is non-technical and accessible.	SHOULD	GUI REQ ID 12, 14, 26, 27, 28, 29, 32, 33, 58, 59, 62, 63, 66, 75, 76
REQ – NonFunc 2	CIM level collaboration - the VIDE environment MAY offer collaboration mechanisms. It may be possible for CIM or PIM users to collaboratively work on a shared CIM view through a communication mechanism (such as shared notes or links to shared views between stakeholders).	MAY	GUI REQ ID 10,67
REQ – NonFunc 3	On-line support for CIM/PIM users - Users working at the CIM/PIM level should have immediate access to online/in-system, context sensitive help that describes how transformations between CIM, PIM and PSM levels are specified and used in the modelling activities supported by VIDE. Help should be expressed in non-technical terms wherever possible	SHOULD	GUI REQ ID 12,19, 25, 27, 34, 47, 48, 58, 59, 60, 61, 63, 75, 76, 78
REQ – NonFunc 4	Clear and unambiguous notation - the VIDE environment should use notation that has clear, comprehensible and unambiguous semantics suited for the user working at the CIM, PIM or PSM level.	SHOULD	GUI REQ ID 12, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 31, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 61, 77
REQ – NonFunc 5	Model view saliency - VIDE models views must be user-oriented. Views on CIM, PIM and PSM must be controllable depending on specific user interactions with the VIDE environment. It should be possible for users to dynamically control the scope and technical content of these views depending on their specification/comprehension needs.	SHOULD	GUI REQ ID 12, 21, 24, 25, 27, 28, 29, 30, 31, 35, 51, 58, 59, 60, 64, 65, 66
REQ – NonFunc 6	Appropriate textual/graphical fidelity - VIDE must provide appropriate textual and graphical modalities for its users. They should be able to work with textual or graphical notations that offer the most effective expressiveness for CIM, PIM and PSM concerns.	SHOULD	GUI REQ ID 1, 2, 3, 16, 17, 18, 19, 20, 21 ,22, 23, 24, 25, 26, 27, 28, 35, 60, 64, 69
REQ – NonFunc 7	Timely feedback and constraints - the VIDE environment should provide feedback on user actions at all modelling levels. Multiple users working on the same VIDE project should receive rapid feedback on their attempted actions within the VIDE environment. Such feedback should indicate their success or failure to complete an action or task; its impact on their local modelling level; its potential impact on other modelling levels; and any constraints that may impact on the success of their intended action.	SHOULD	GUI REQ ID 12, 12a, 25

REQ – NonFunc 8	Runnable and testable VIDE prototypes - the VIDE environment should allow execution of runnable models. VIDE users should be able to validate at any time (where possible) the models that can be automatically transformed into an executable form.	SHOULD	GUI REQ ID 7, 52, 53, 54, 55, 56
REQ – User 1	Flexibility and interoperability of VIDE language and tools - the VIDE language and tools should have flexibility and be interoperable with existing tools	SHOULD	GUI REQ ID 11, 57, 68
REQ – User 2	Reuse of UML Standard - the VIDE tools for certain user groups SHOULD be informed by existing tools for the user groups. End users are very sensitive to using standards.	SHOULD	GUI REQ ID 4, 5, 11
REQ – Tool 7	Meta-modelling Framework - VIDE SHOULD use GMF as it's graphical modelling framework	SHOULD	GUI REQ ID 11
REQ – Tool 9	CIM modelling standards - VIDE may support CIM level modelling with BPMN; where there is inadequate or no support for BMPN, VIDE may provide CIM modelling capability with UML activity diagrams	MAY	GUI REQ ID 61,68
REQ – Tool 12	VIDE extensibility - the VIDE tools should be extensible via a plug-in mechanism.	SHOULD	GUI REQ ID 6, 68, 84
REQ – Tool 14	Model driven approach - the VIDE tool must strictly follow a model driven approach.	MUST	GUI REQ ID 63, 70, 71, 82
REQ – TRAC1	VIDE must allow the traceability and persistence of models	MUST	GUI REC ID 8, 9, 10, 15, 34a, 49, 50, 51, 58, 67, 72, 73, 74, 79, 80, 81, 83
REQ – DOC1	VIDE should allow for document generation at all levels	SHOULD	GUI REC ID 14,28

Table 23: : Requirements from Work Package 1 cross referenced with WP5 low level requirements

7.5 Summary

This chapter has explained the feedback received from focus groups and the evaluation that was carried out on the Exploratory prototype. This work highlighted several possible areas for improvement over the initial design and has provided valuable lessons which have resulted in a much clearer set of requirements to be taken forward into the design of the Definitive prototype. Some of the requirements have been highlighted by endeavouring to draw together the work from Work package 1 and the high level requirements and how they and the research in the areas of Software Visualisation, Visual programming and Diagramming have influenced the work that has been done.

8. Definitive Prototype

8.1 Introduction

Following the specification of the Exploratory prototype as outlined in Chapter 6 and discussion in Chapter 7 which described how the requirements have been drawn together, this Chapter gives the specification of the new definitive prototype of the CIM level interface designs. It follows a top-down approach by commencing with detailed screen designs at the pre-CIM level. The pre-CIM level has been identified as the area where the scrapbook will sit. It is the area that is 'upstream' of the CIM giving business users an informal modelling tool so that they are not constrained by their lack of knowledge of any formal modelling paradigm.

8.2 Overview

The architecture of the pre-CIM to PIM level is illustrated in Figure 47.

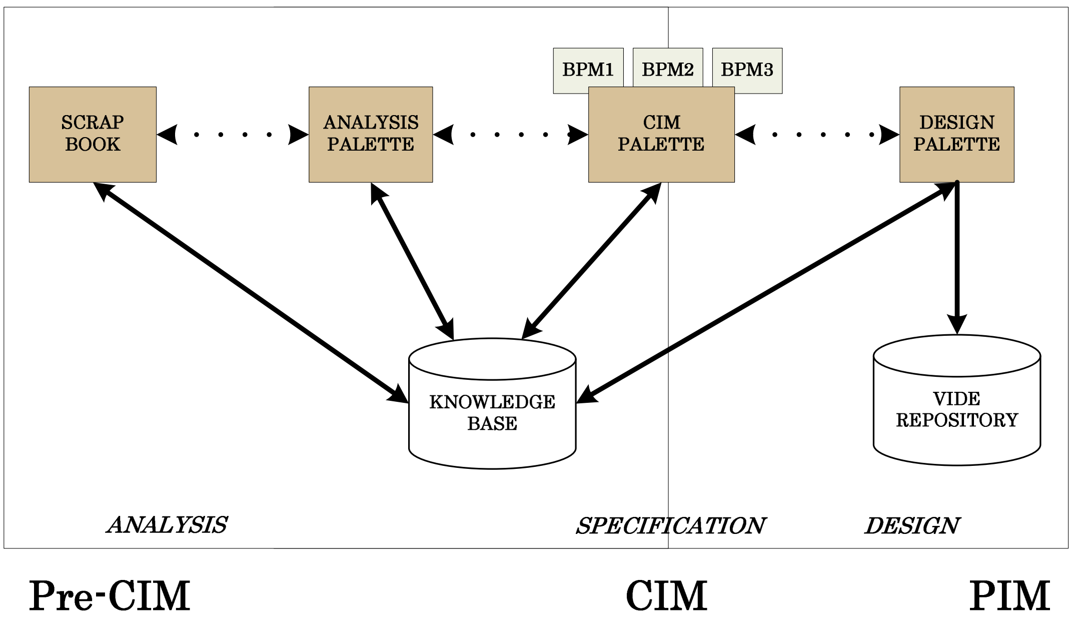


Figure 47: Architecture of the pre-CIM to PIM process for the Definitive prototype

8.2.1 Project Management

The screen below is a placeholder for the welcome screen of the VIDE IDE, it will be more colourful and engaging when built and is shown in Figure 48 for completeness.

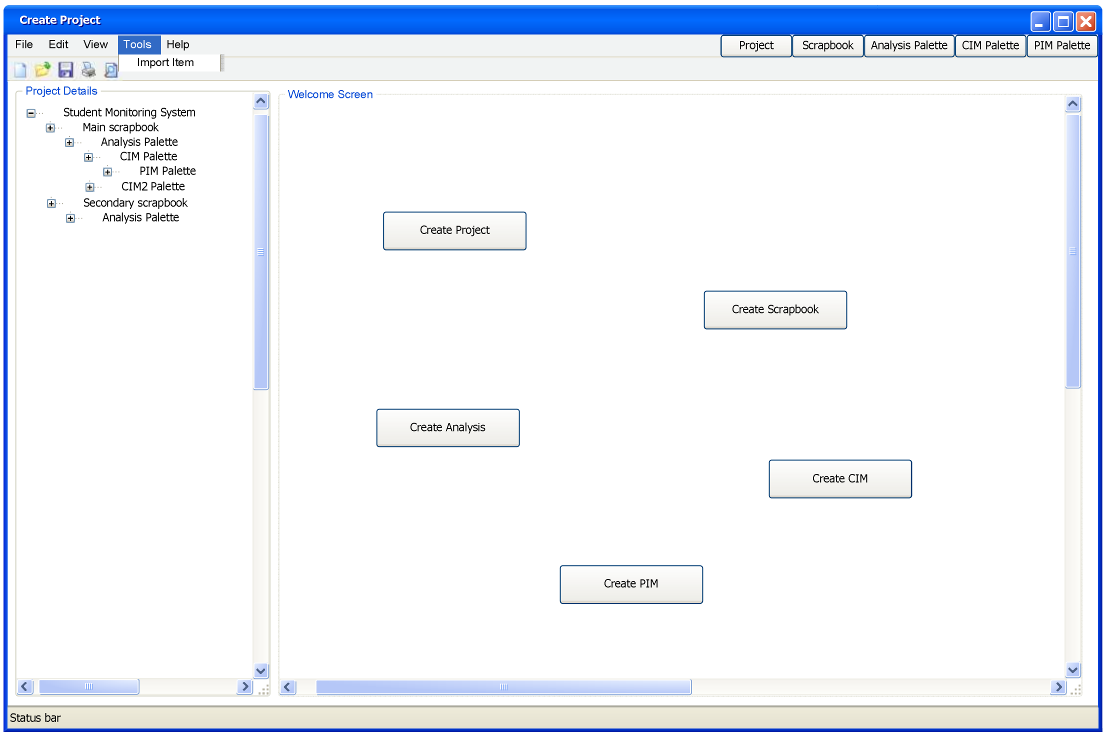


Figure 48: Creating a project using the VIDE IDE

The VIDE IDE provides a development environment where development artefacts (e.g., analysis models, design models) are organised into projects. Figure 49 shows the project creation environment with options to build various artefacts that may form a project.

Users can create multiple projects; each project can contain multiple scrapbooks, analyses, CIMs and PIMs which can be organised as required. A user does not have to create each of these elements, for example they may go straight to the CIM level model, bypassing the scrapbook and analysis.

All the models that are created or imported (users can also import existing models) will be stored in the model repository, and multiple versions of models can also be created and stored. The use of the repository in the definitive prototype allows for model persistence, which was lacking in the exploratory prototype.

The Scrapbook

The concept of the scrapbook has been retained from the exploratory prototype, however it has been restructured and simplified to present a much less cluttered, more easily navigated model. The tree structure, with additional links available, has been used for the scrapbook following the findings of the exploratory prototype evaluation and the concepts and models have also been simplified to enable novice (non-technical) users to use the tool.

A scrapbook can contain any documents the user sees as being useful for the project. These can be existing files or specially created for the project. Once a project has been created, the user may follow the File menu to create a New Scrapbook or to Open an existing scrapbook, etc: as shown in Figure 49.

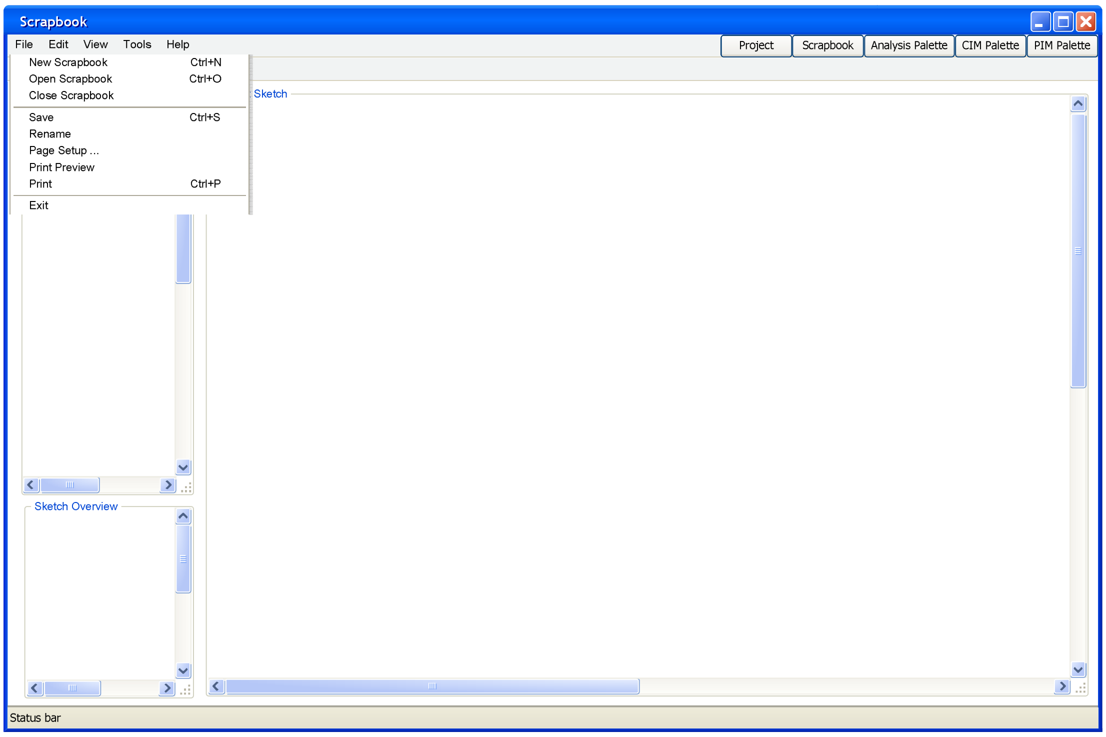


Figure 49: Scrapbook operations

Using the Opportunity Management scenario, an example of using the VIDE GUI to create a new scrapbook is shown in Figure 50.

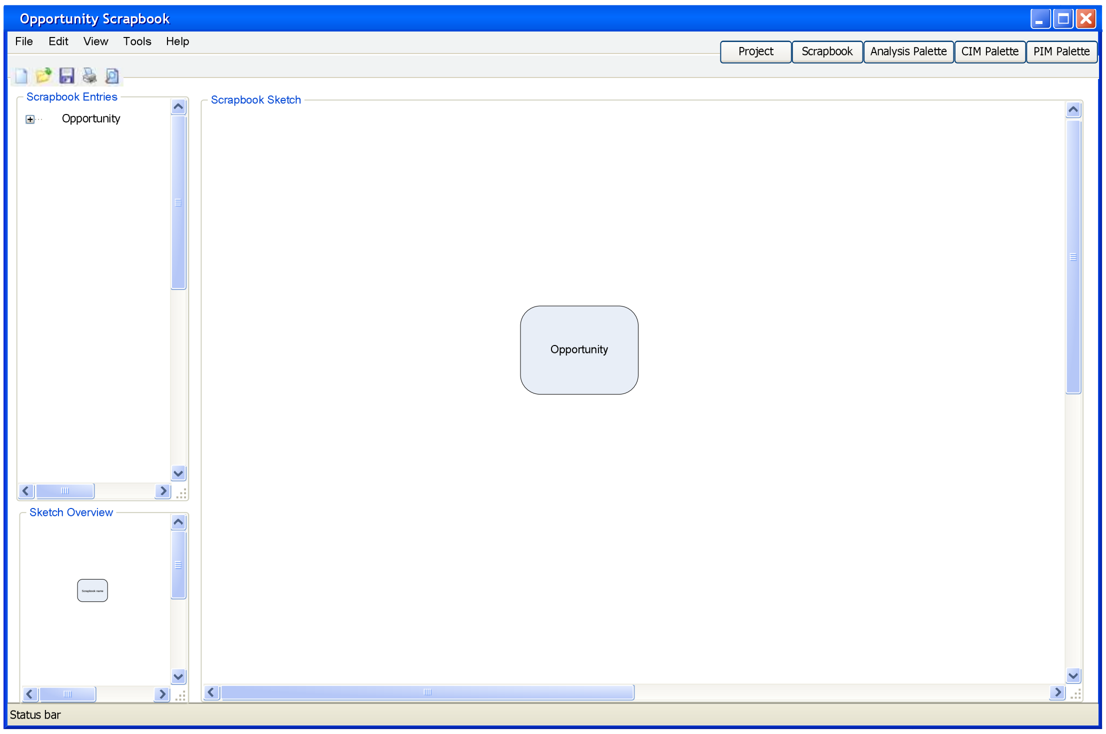


Figure 50: Opportunity scrapbook example

Once a scrapbook (e.g., the opportunity scrapbook) has been created, a user might want to add folders, documents or scraps to the scrapbook. A drop-down list of menu choices is provided for such operations (see Figure 51).

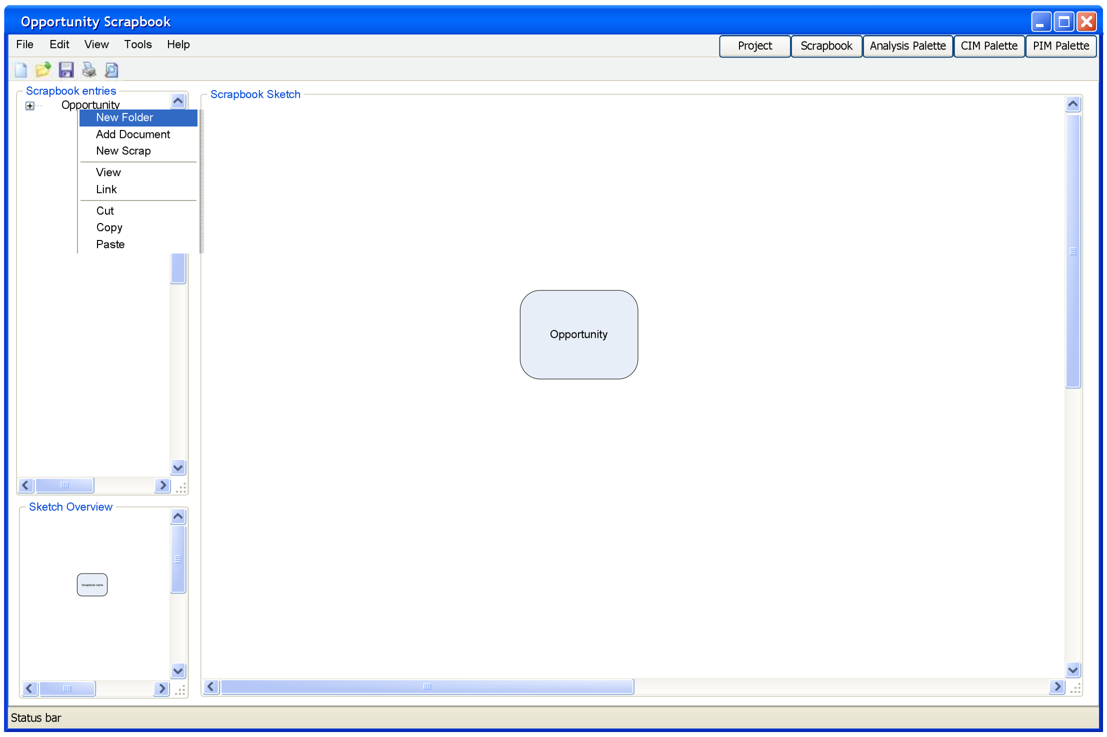
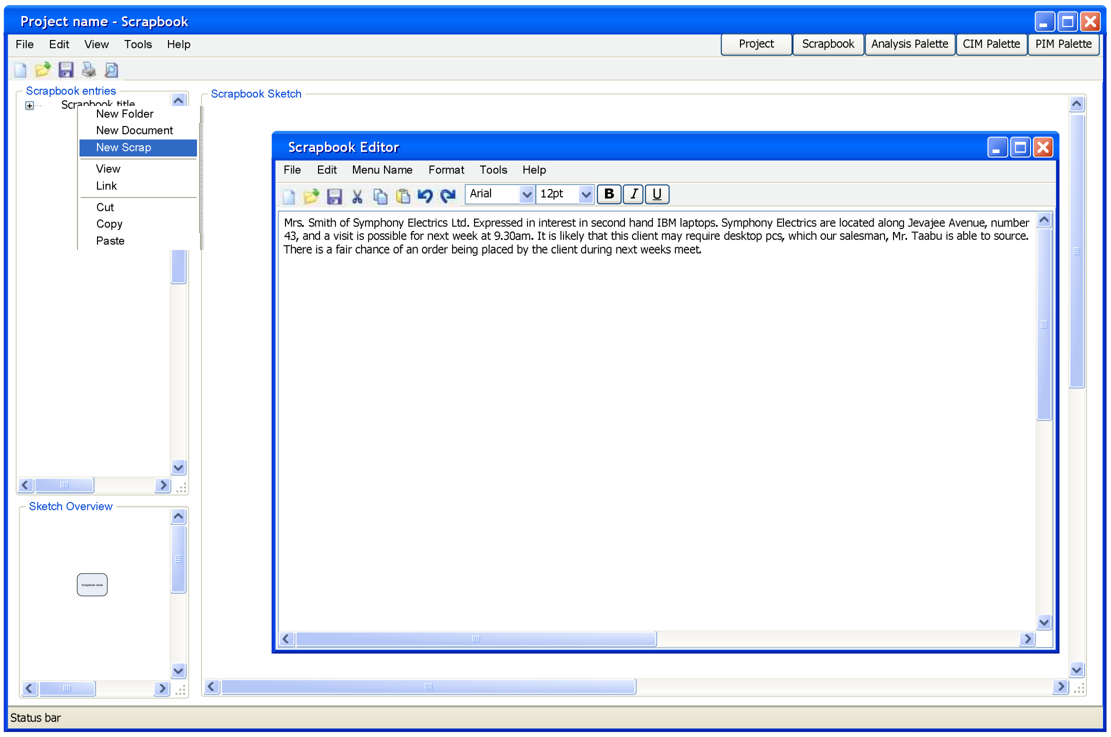


Figure 51: Operations for adding elements to a scrapbook

If there are existing folders containing information about products or contacts, that concern the business opportunity, or specific documents that comprise information regarding existing orders or issues to be addressed, these folders and documents can be browsed and items can be selected for inclusion into the scrapbook, using the standard Windows dialog as shown in Figure 52.

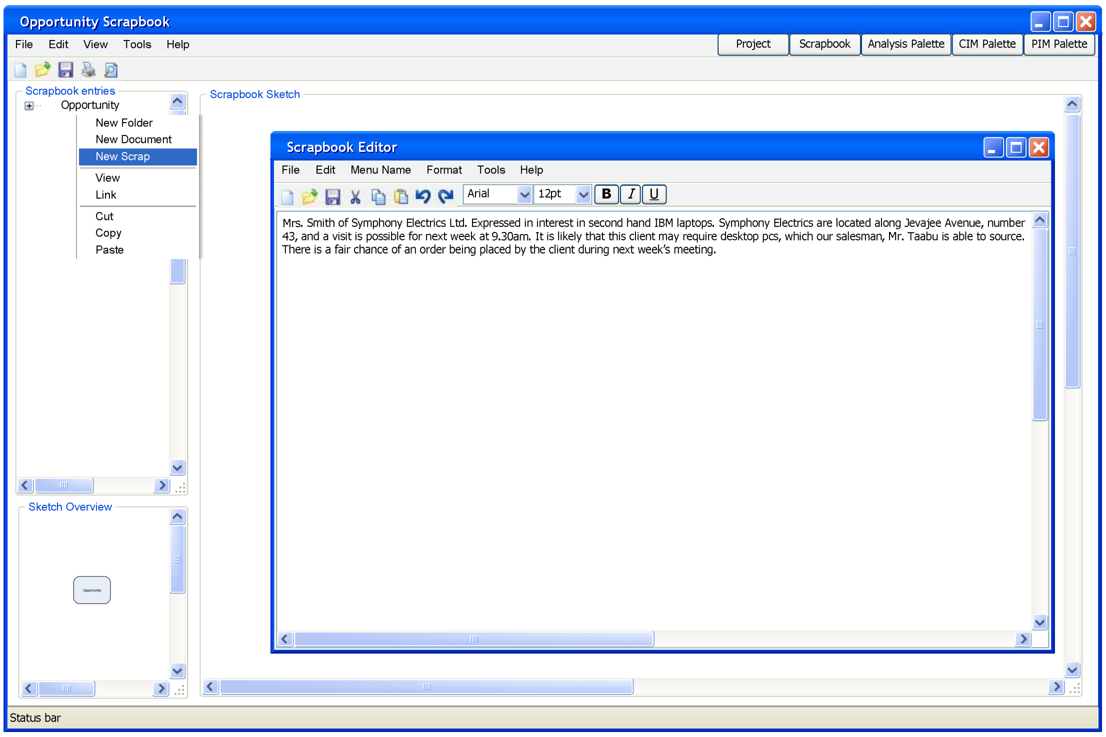


Mrs. Smith of Symphony Electrics Ltd. Expressed in interest in second hand IBM laptops. Symphony Electrics are located along Jevajee Avenue, number 43, and a visit is possible for next week at 9.30am. It is likely that this client may require desktop pcs, which our salesman, Mr. Taabu is able to source. There is a fair chance of an order being placed by the client during next weeks meet.

Figure 52: Selecting documents or folders

The VIDE tool can import documents into the scrapbook which can then be used as the basis of further analysis.

A scrap is a fragment of information (either imported or created) that will be required for the modelling later. Adding a scrap is crucial in capturing and recording information about the problem domain. For example, a scrap about an opportunity might be as shown in Figure 53.

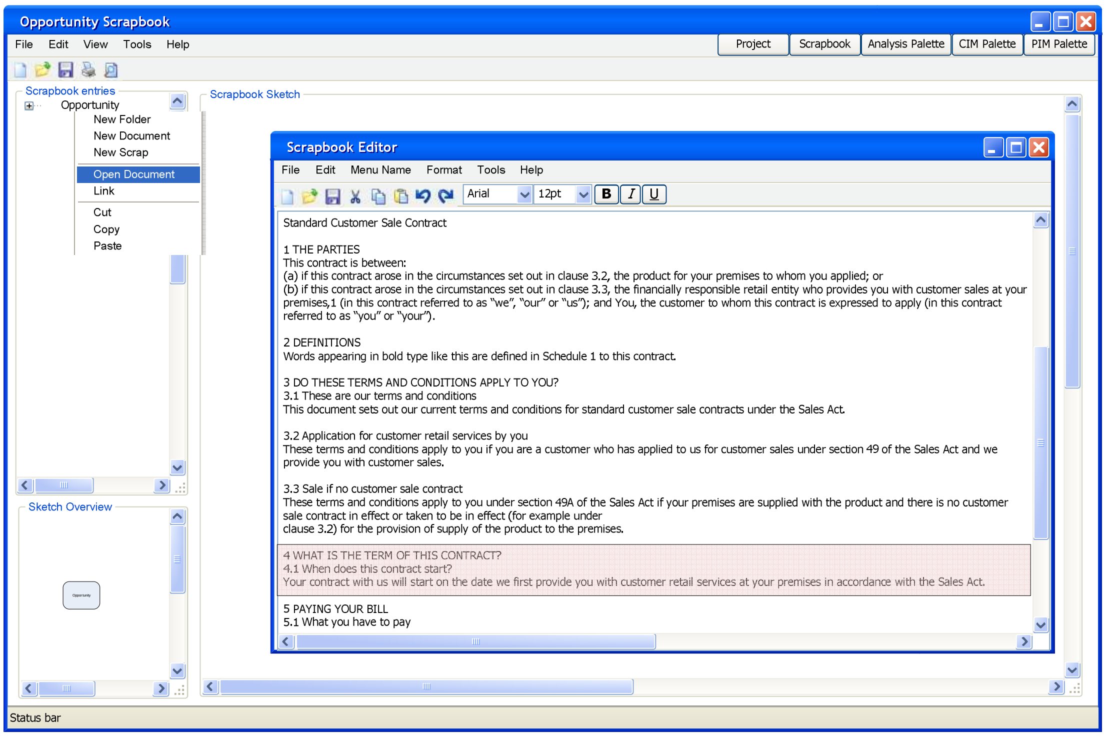


Mrs. Smith of Symphony Electrics Ltd. Expressed in interest in second hand IBM laptops. Symphony Electrics are located along Jevajee Avenue, number 43, and a visit is possible for next week at 9.30am. It is likely that this client may require desktop pcs, which our salesman, Mr. Taabu is able to source. There is a fair chance of an order being placed by the client during next week's meeting.

Figure 53: Creating a new scrap to add to the Scrapbook

A small, simple editor for the creation and modification of both documents and scraps will be included in the tool.

Often, a scrap may be populated with content from an existing document. Figure 54 demonstrates support for such functionality within the VIDE IDE.



Standard Customer Sale Contract

1 THE PARTIES
This contract is between:
(a) if this contract arose in the circumstances set out in clause 3.2, the product for your premises to whom you applied; or
(b) if this contract arose in the circumstances set out in clause 3.3, the financially responsible retail entity who provides you with customer sales at your premises,1 (in this contract referred to as "we", "our" or "us"); and You, the customer to whom this contract is expressed to apply (in this contract referred to as "you" or "your").

2 DEFINITIONS
Words appearing in bold type like this are defined in Schedule 1 to this contract.

3 DO THESE TERMS AND CONDITIONS APPLY TO YOU?
3.1 These are our terms and conditions
This document sets out our current terms and conditions for standard customer sale contracts under the Sales Act.

3.2 Application for customer retail services by you
These terms and conditions apply to you if you are a customer who has applied to us for customer sales under section 49 of the Sales Act and we provide you with customer sales.

3.3 Sale if no customer sale contract
These terms and conditions apply to you under section 49A of the Sales Act if your premises are supplied with the product and there is no customer sale contract in effect or taken to be in effect (for example under clause 3.2) for the provision of supply of the product to the premises.

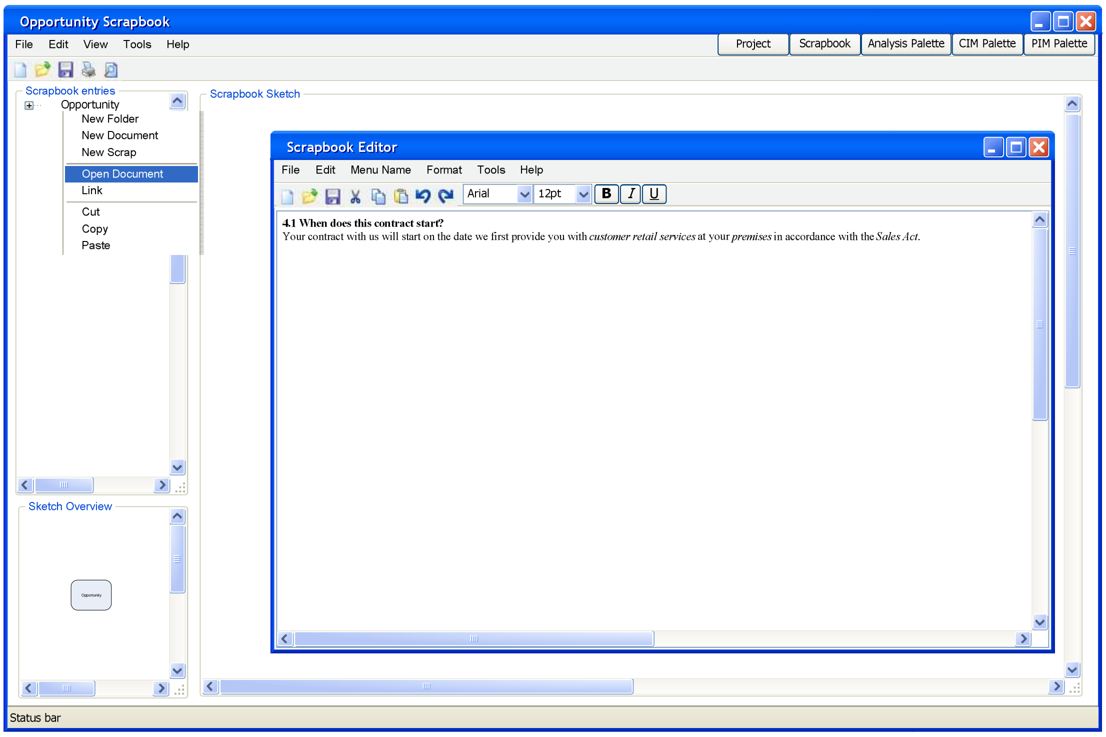
4 WHAT IS THE TERM OF THIS CONTRACT?
4.1 When does this contract start?
Your contract with us will start on the date we first provide you with customer retail services at your premises in accordance with the Sales Act.

5 PAYING YOUR BILL
5.1 What you have to pay

Figure 54: Selecting text from an existing document to populate a scrap

Thus the required text can be highlighted and a scrap can be created by selecting an option in the tool menu. Multiple scraps can be created from one document and these scraps must be traceable back to the original document. Traceability is an important addition to the definitive prototype as it allows the users to move backwards and forwards through the information, one of the issue found in the evaluation of the exploratory prototype. This concept is supported in all the models used in the VIDE IDE.

A user may want to open an existing scrap to review or modify its contents and this is shown in Figure 55.



4.1 When does this contract start?
Your contract with us will start on the date we first provide you with *customer retail services* at your *premises* in accordance with the *Sales Act*.

Figure 55: Opening an existing scrap in the Scrapbook Editor

As the user adds documents and scraps a 'Scrapbook Sketch' is populated with these documents and scraps, so an informal model or 'sketch' of the information is built up. This helps the user maintain a visual representation of the information they are collating. There is also an overview of the sketch to help the user navigate what could potentially become a very large model. The user can navigate around the potentially large model either by using the overview window or by use of the scroll bars. The user can rearrange and move any items they wish by dragging and dropping elements.

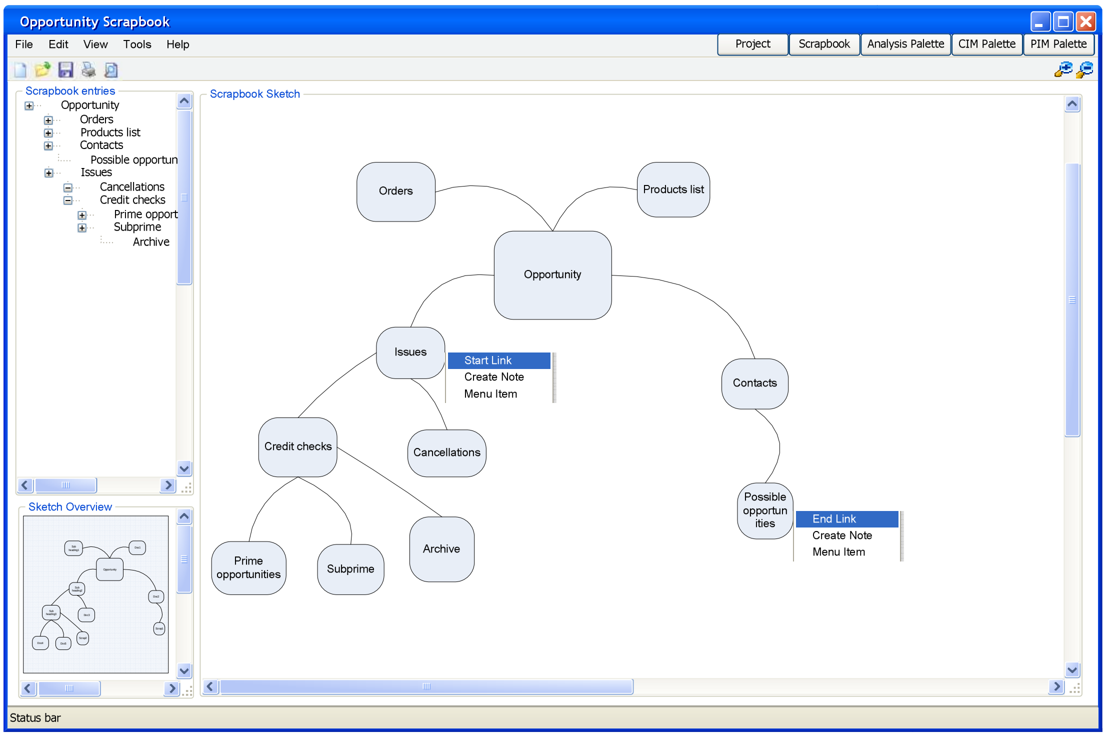


Figure 56: A populated scrapbook and adding further links among scrapbook items

The tree structure will be reflected in the visual model; however it is also possible to create further relationships or links between elements. Creating these links between documents, their source folders and associated scraps is important in making sense of the information elicited from problem domain experts. These links are demonstrated in Figure 57.

It is important to be able to annotate links to add further meaning about the associations between items. This feature was developed after the evaluation of the exploratory prototype, which found that it did not provide any secondary notation for augmenting models. Such annotations can be created as shown in Figure 57.

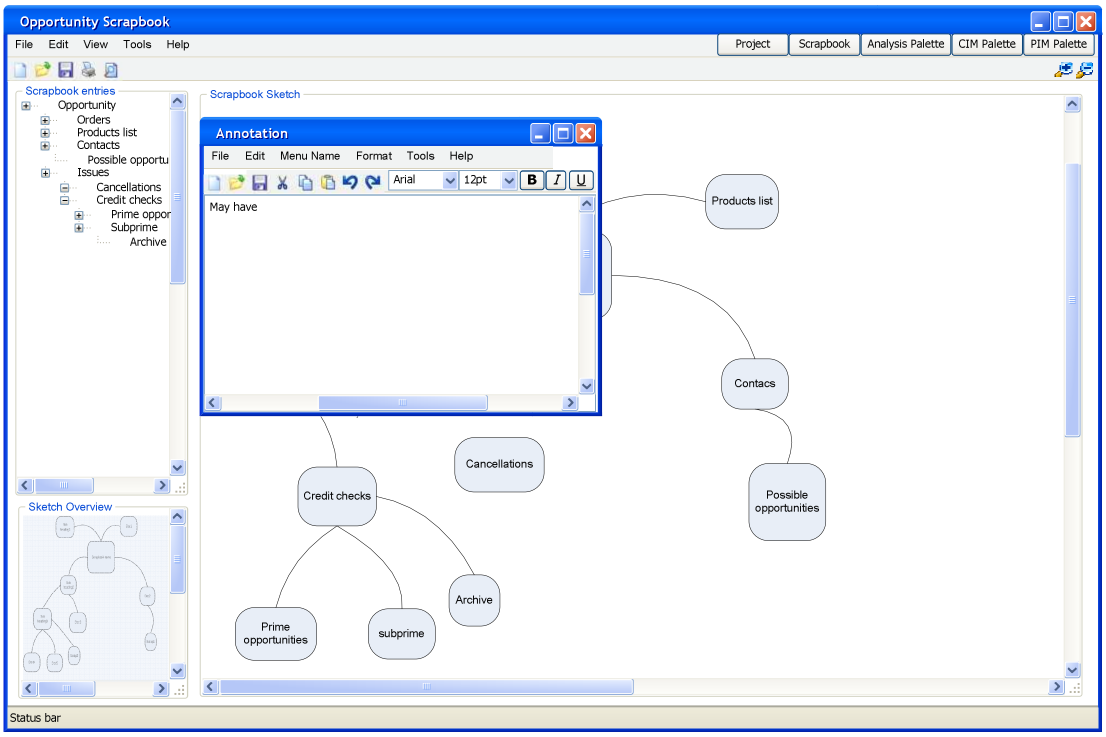


Figure 57: Creating a link and annotation between scrapbook elements

Suppose a user wants to indicate that an association between *issues* and *possible opportunities* exists, they may add the link and an annotation. The example shown in Figure 57 demonstrates how the annotation is created. Figure 58 shows the results of the addition.

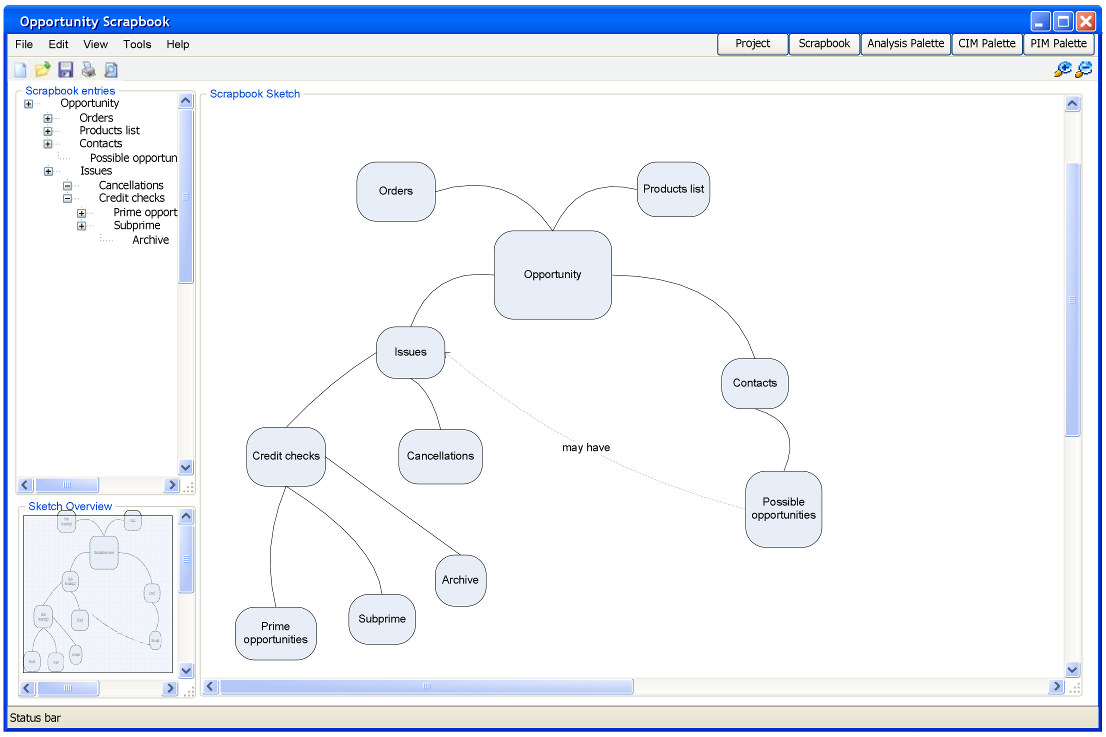


Figure 58: Example of the link between items in the Scrapbook

Once the user has finished building the scrapbook, or is ready to move into the analysis phase, elements from the scrapbook can be identified for taking into the Analysis Palette (AP). The elements required for the analysis model can be obtained from the Scrapbook by dragging and dropping items into the analysis palette ‘shopping cart'. The 'shopping cart' is a screen to the right of the Scrapbook Sketch, labelled Analysis Palette, which will collect items that the user wishes to carry forward into the Analysis Palette. Alternatively, a user can double click a scrapbook item, and it will be lodged in the Analysis Palette 'shopping cart' as shown in Figure 59. This 'shopping cart' containing the list of items has been added to over come the juxtaposition drawback in the exploratory prototype, so now earlier model elements can be seen side by side with the next level of model. That is the scrap book elements can be seen next to the analysis model and the analysis diagram elements can be seen next to the CIM model.

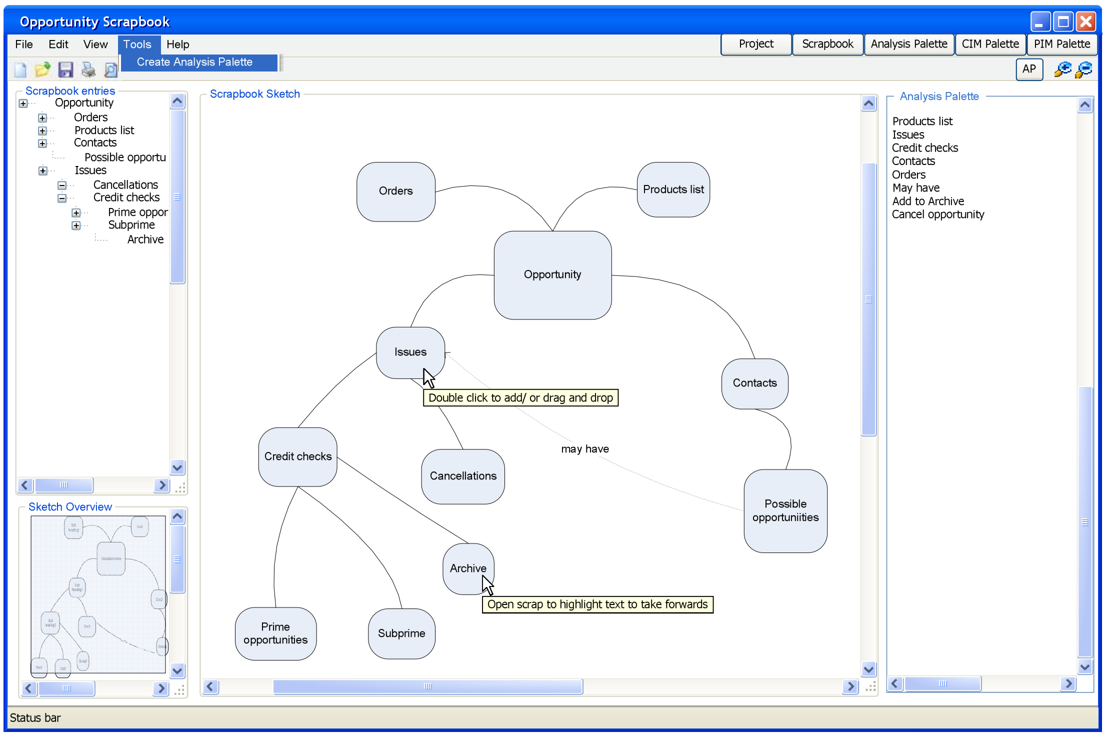


Figure 59: Selecting elements for the analysis model

The Analysis Palette

The Analysis Palette has been added to provide informal modelling that a novice user can use to begin to model the problem domain. This is in response to the evaluation of the exploratory prototype "Non-technical users working at the CIM level should be able to input, retrieve and understand their business domain descriptions in a notation that is non-technical and accessible". Also resulting from the evaluation, navigation should be intuitive so that users can to return to the original information source in the scrapbook and link to other views of the same information.

In order to create an analysis model, the Analysis Palette (AP) provides the user with a set of constructs. The bloop (cloud shape) is used to represent an item from the scrapbook that is not well understood. Bloops can be later decomposed into roles, activities or data items. This much more informal model element has been added to overcome the identified '*premature commitment* 'constraint in the exploratory prototype. This allows the user to develop a quick model and then use a process of gradual refinement, as identified in the exploratory prototype evaluation, to decide what each bloop actual is, or even to leave the bloops definition until the CIM level.

The AP also provides the user with a rectangle shape which may be used to represent a role, activity or data. An annotation with **A** at the top left corner of the rectangle would indicate an activity, whereas **R** would be used to indicate a role, and **D** would be used to signify data. These annotations should aid the users in distinguishing between the various elements, so attempting to solve one of the problems found in the exploratory prototype.

An example AP model is shown in Figure 60:

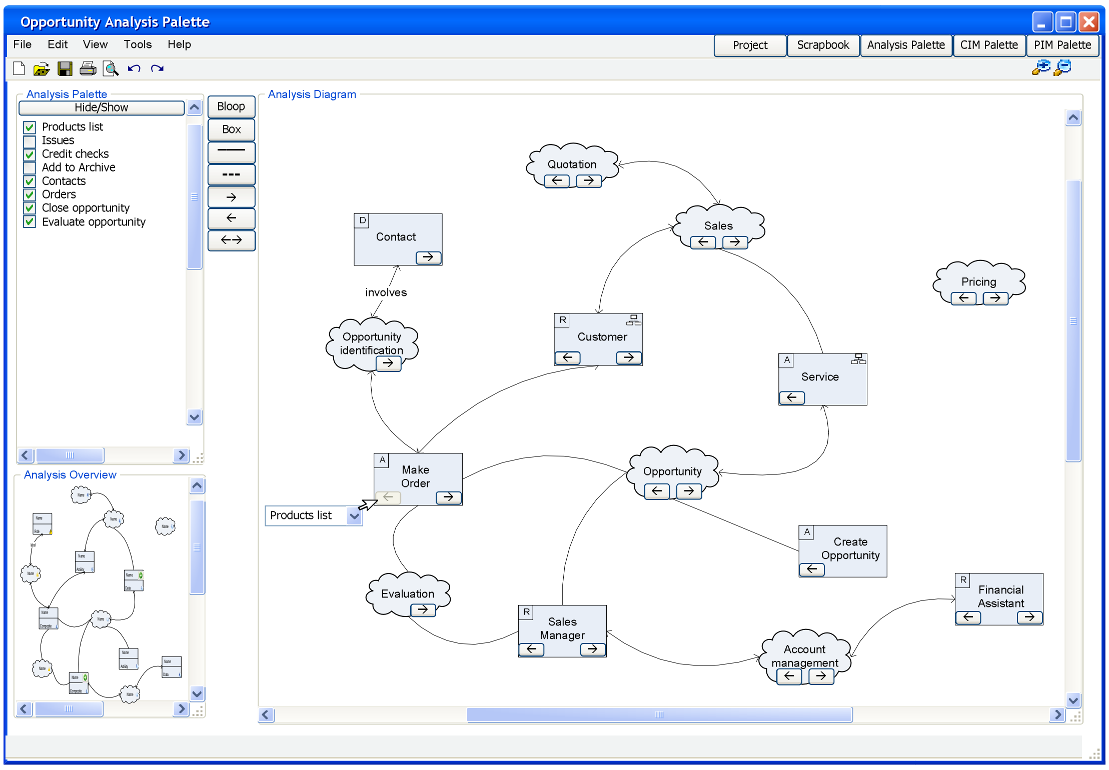


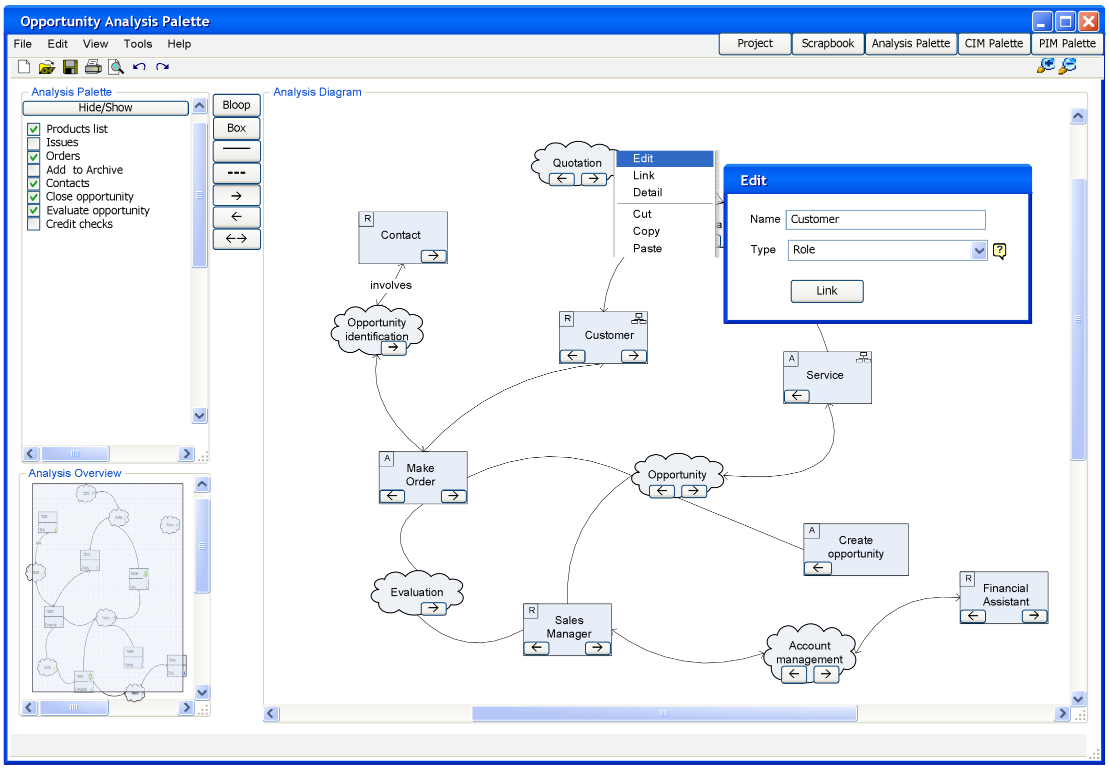
Figure 60: Analysis model with bloops, activities, roles and data items

The user would need to build up this analysis model based on the elements bought forward from the scrapbook. Traceability is supported by left and right arrows on each diagram element. A left pointing arrow would indicate that the element links backwards into more information in the scrapbook. A right pointing arrow would indicate linkages into a CIM model (the next model to be developed). An absence of an arrow means there is no traceability in that direction.

The activity, role or data may also be composed of more elements, that is there levelling of the diagram. If an element has been decomposed and further detail is available a small hierarchy icon appears in the top right of a rectangle. Double clicking on this icon will show the lower level of detail, this is shown in Figure 63.

Users can add any diagram element they wish using the diagramming tools in the vertical tool bar to the left of the screen.

In order to specify the type of a given analysis model element (e.g., whether an activity, role, etc), a user is provided with a dialogue box from which appropriate types can be assigned to model elements. For example, to assign the type *Role* to a Customer, the following dialog would be used:



Edit
Name: Customer
Type: Role
Link

Figure 61: Assigning types to Analysis Palette elements

Part of this dialogue of assigning types to elements will be a help sheet that will explain the terminology, thus novice users can be guided through the process of refining a bloop into an activity, role or data.

If elements are added at this stage, there will be no traceability back to the scrapbook model even if something does exist that the new element could be linked backwards to. So the VIDE IDE provides a means for users to link specific elements back to a desired scrapbook element. For example, suppose that a modeller adds a *Make order* activity into the analysis palette and wants to link it to the *Order* element back in the scrapbook the following dialog could be used: See Figure 62

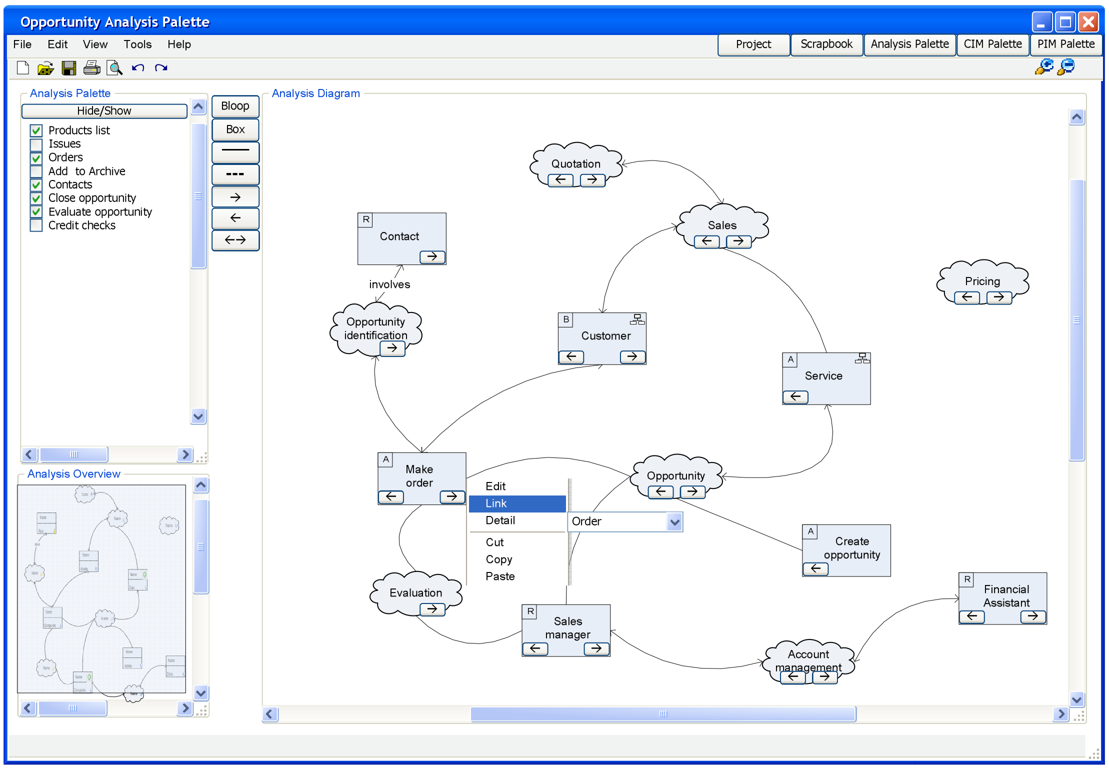


Figure 62: Linking an AP element to a scrapbook element

An important aspect of the analysis models is to be able to expand a given element to view its sub-elements, or to collapse sub-elements into their parent. An analysis model element that is expandable is signified by an organisational chart icon to the top right corner of the element. In Figure 62, such an element is the Service activity. We may suppose that the Service activity within the Opportunity management scenario comprises of related activities such as Service request and Service confirmation. An expansion of the Service activity is shown in Figure 63:

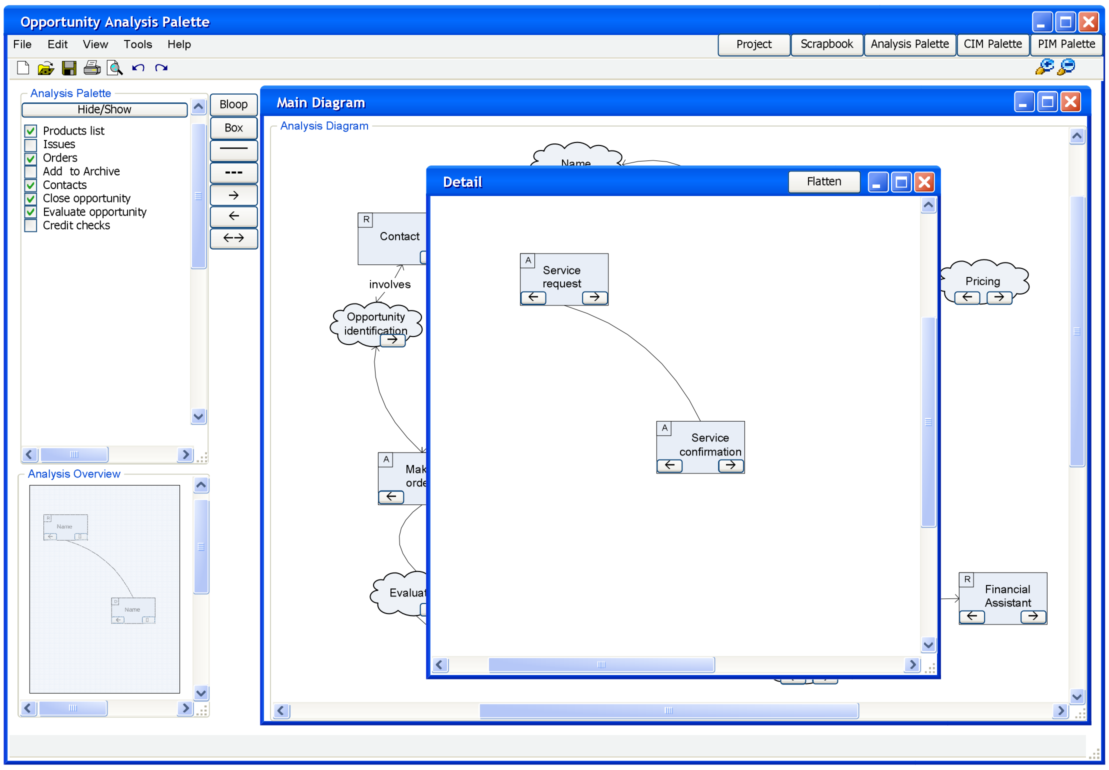
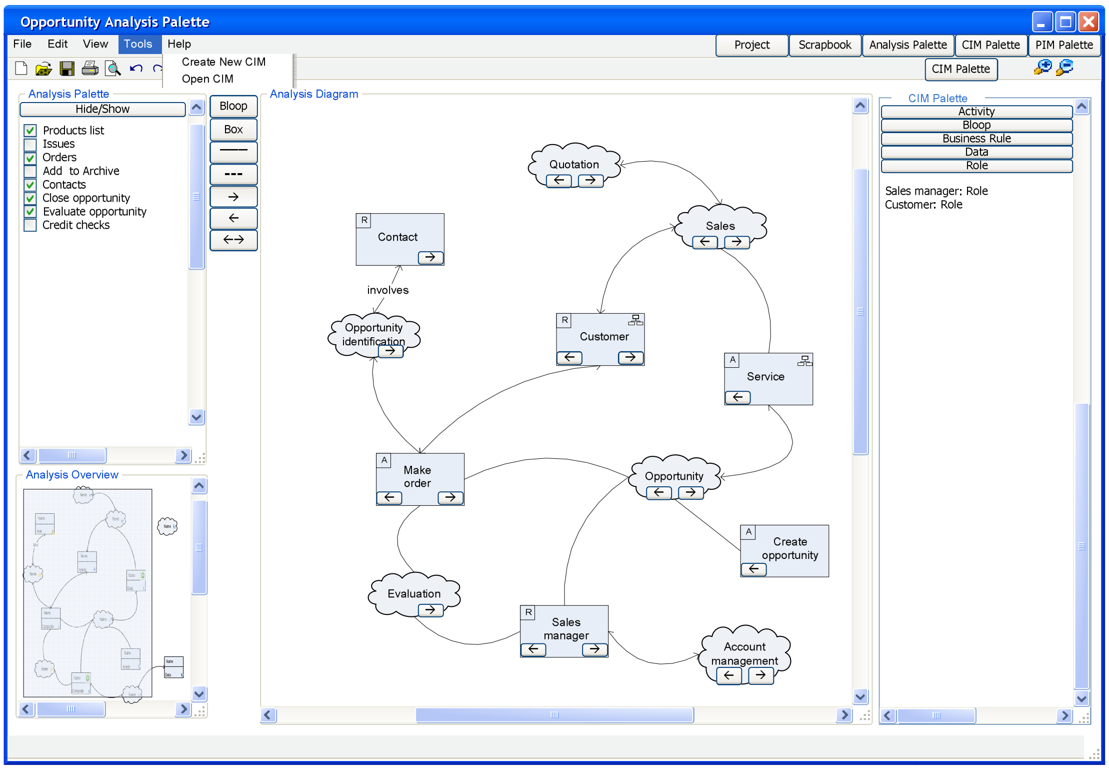


Figure 63: Viewing more detail of a model element in the Analysis Palette

Once an analysis model has been created, and types assigned to various elements, a user may now start considering the elements of the analysis model that they want to be carried forward for the building of a CIM model. For example, a user may immediately identify roles such as Sales Manager and Customer (from the analysis model), which would also feature within the CIM model as roles and thus be placed in the 'shopping cart' to be carried forwards as shown in Figure 64.



Sales manager: Role
Customer: Role

Figure 64: Identifying CIM elements from the analysis model

8.2.2 CIM Modelling

The CIM Palette shows the elements selected from the Analysis Palette and brought forward in the pane on the left (keeping the consistent placing and functionality to match the relationship between the scrapbook and the Analysis Palette).

The CIM uses the VCLL (VIDE CIM Language developed by IWi) language. A typical VCLL CIM model comprises of activities, data, roles, and associations among the roles and respective activities, and produced or used data. Hence, since an analysis model consists of elements (e.g., activities, roles, data, etc) that may be mapped to those in the CIM model, it is expected that many CIM elements would be derived from the analysis model and be thus automatically transformed. The CIM model below shows a list of activities obtained from the analysis model and appearing in the CIM model:

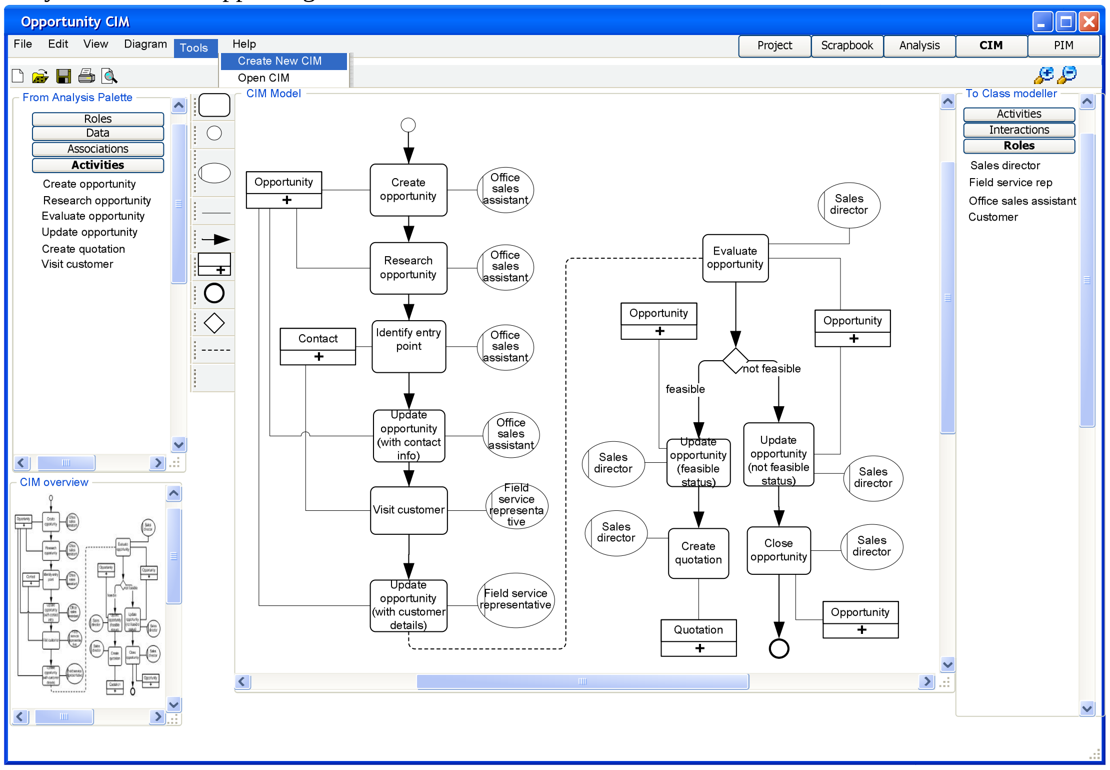
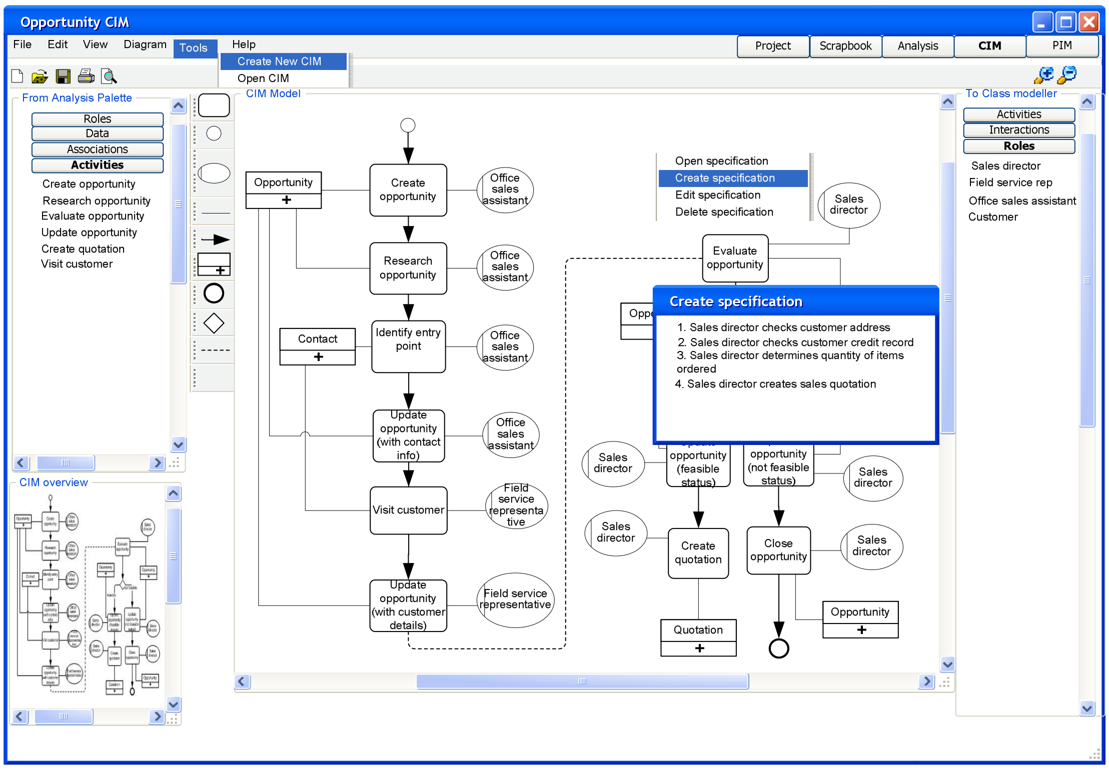


Figure 65: CIM model with elements derived from analysis model

One of the crucial development activities following business process modelling is the production of a specification for parts of the business model that may be developed into a software system. For example, associations among roles and activities may require specification to describe the behaviour of a system that would support performance of such activities. The VIDE IDE provides an editor for writing use case descriptions as a specification of such parts of the system. Suppose a modeller would like to build a use case description providing further detail about the Sales director’s activity of ‘Evaluate opportunity. The following dialog could be used:



Create specification

1. Sales director checks customer address
2. Sales director checks customer credit record
3. Sales director determines quantity of items ordered
4. Sales director creates sales quotation

Figure 66: Creating a specification

A complementary model to a VCLL CIM model is the Entity Relation (ER) model. Like a VCLL CIM model, an ER model is built by obtaining many of the entities from the analysis model, and making associations among the entities based on a user's understanding of the problem domain (see Figure 67).

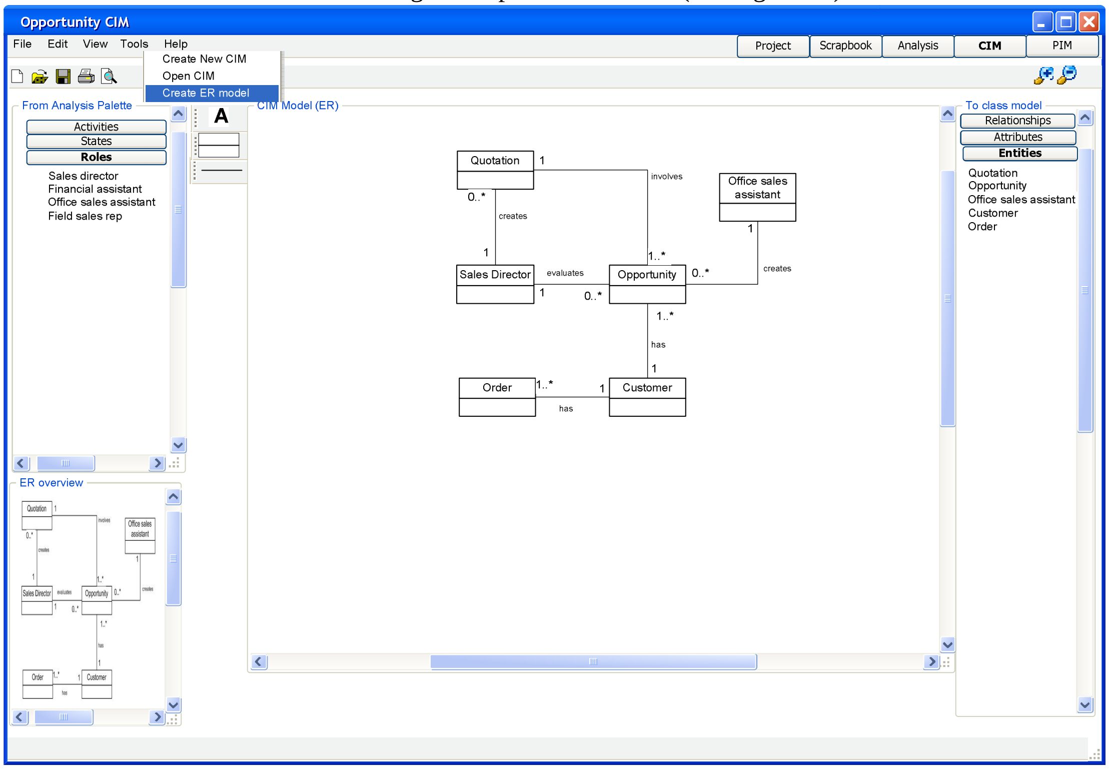


Figure 67: Partial ER model of opportunity management scenario

8.2.2.1 First-cut design model

Creating a first-cut class model would be a useful input into PIM level design. Given that the VCLL and ER models both comprise of elements that may be used directly to build a class model, the VIDE IDE provides a wizard whereby the modeller can select elements such as Roles from a VCLL model to form part of the class model. The modeller can also select specific entities from the ER model which can be directly placed in the class editor as classes.

Suppose a modeller intends to obtain some classes from the CIM model shown in Figure 68. Double-clicking a given category of items (e.g., Roles) in the “To Class Modeller” window displays a dialog from which any of the roles may be selected for automatic rendering as classes in the class editor window:

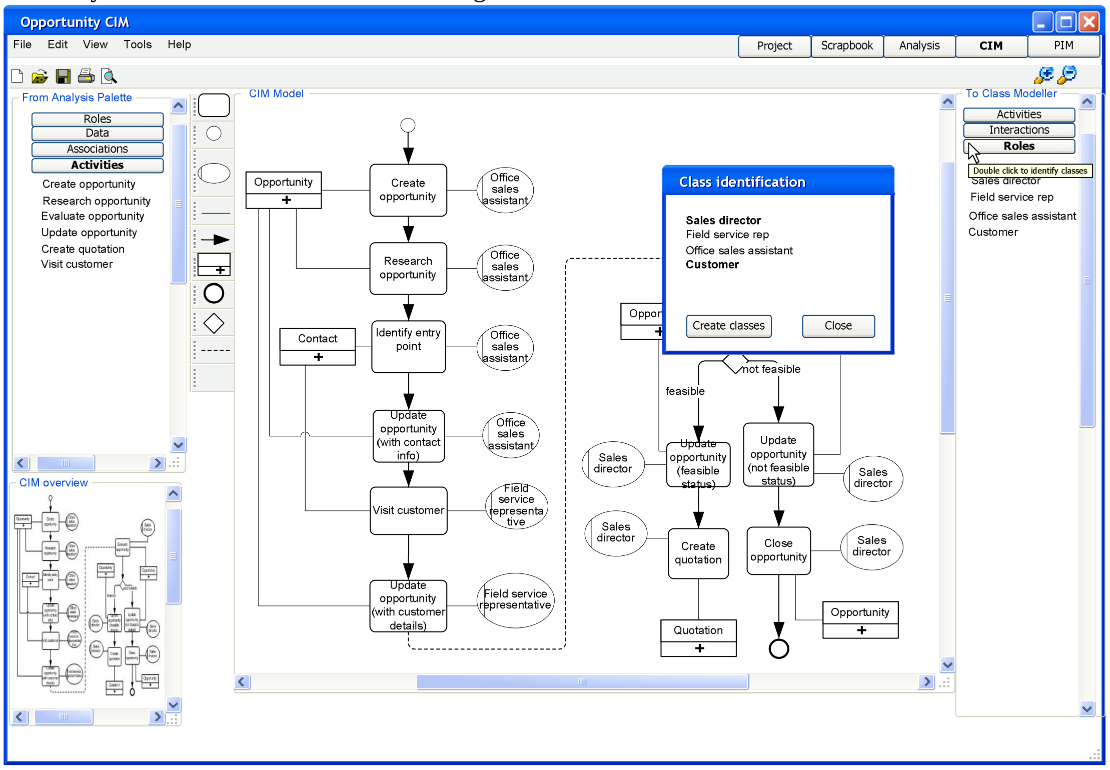


Figure 68: Class identification from roles

In Figure 68, a user double-clicks the *Roles* button, and a dialog with the list of available roles is provided. From the list in Figure 68, only two roles have been selected for creating respective classes, namely, Sales director and Customer. The selected candidate classes are in bold print.

In the class model editor, the list of roles from which a user made a selection is provided, with the already chosen candidate classes indicated in bold. The other roles are listed in case a user wants to add them to the class model as classes, or even as attributes. The class modeller allows additional classes to be added even where such classes are not derived from any CIM element. A first cut class model with further classes (rather than just the sales director and customer classes) is shown in Figure 69:

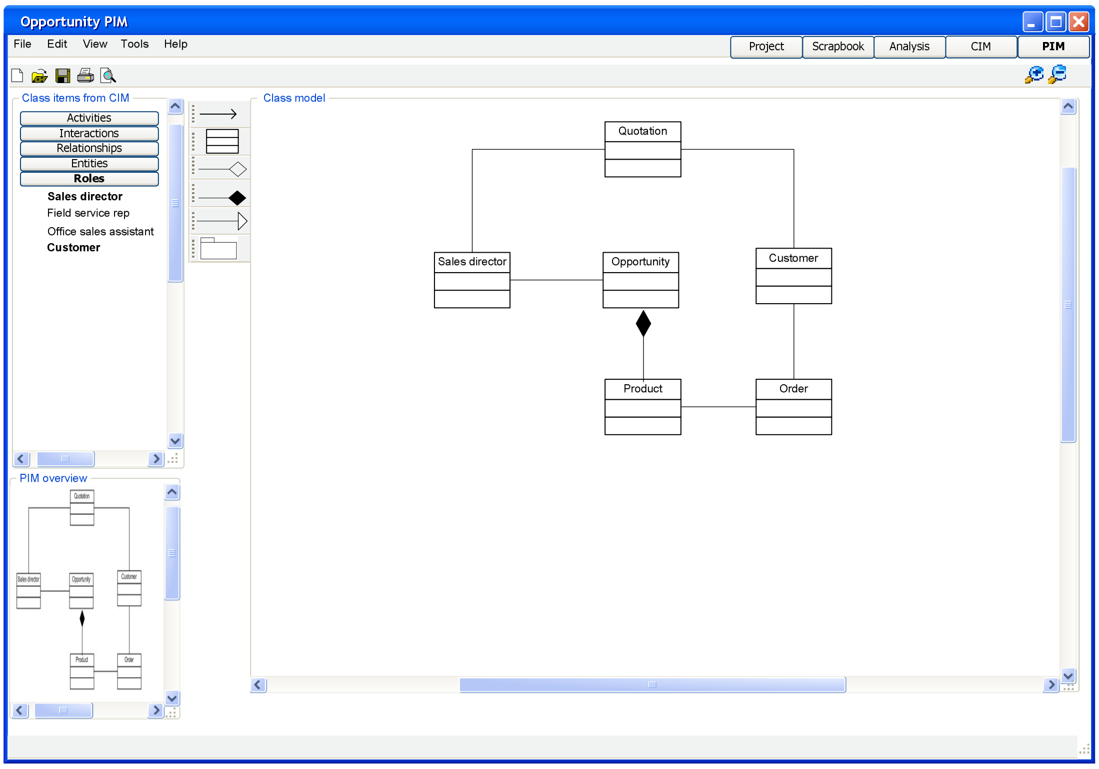


Figure 69: First cut class model

One way to add attributes or methods to the classes is to use a wizard similar to the one used in class identification. For example, activities in the CIM model may often be possible methods of some of the classes. Hence, double-clicking the activities button will activate a wizard where a user is able to select candidate methods. Since the VCLL model is such that activities have direct associations with Roles, selected activities for use as methods are added to respective classes. For example, selecting the “evaluate opportunity” activity to use as a method in the class model would place the corresponding method in the Sales director class. Methods or attributes that have no correspondence to a CIM element can be added as desired using the pop-up dialog in Figure 70.

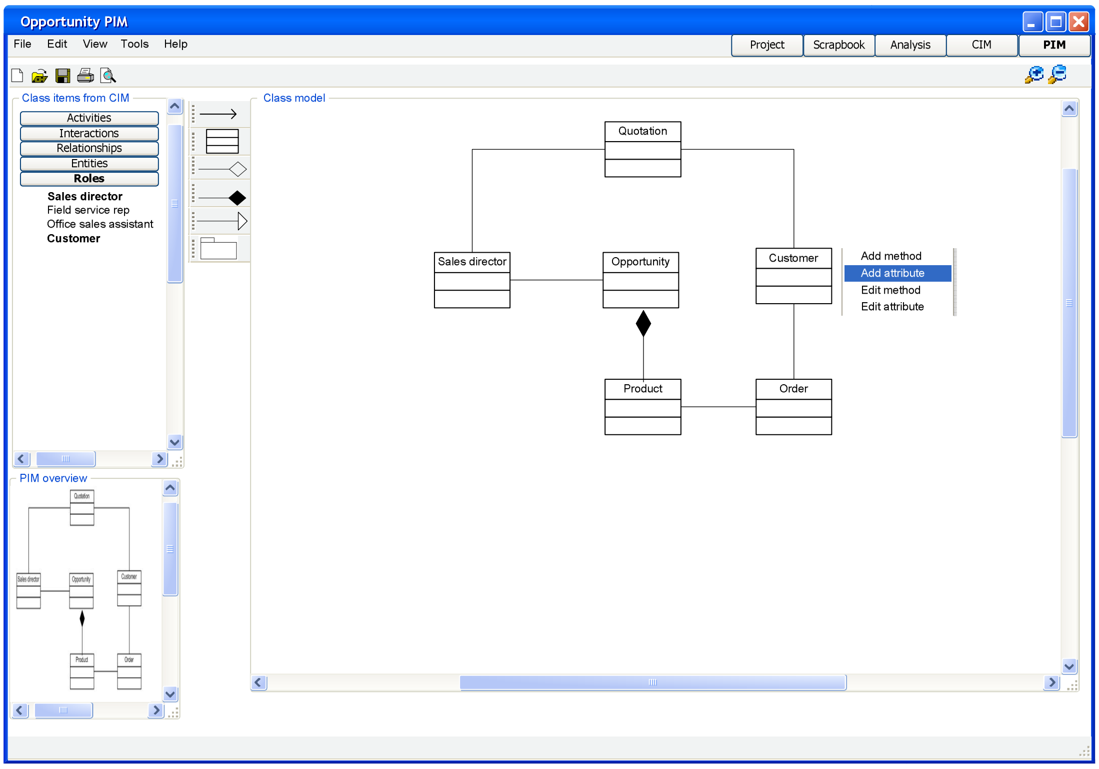


Figure 70: Adding new methods or attributes

An important part of elaborating a class/structural model is providing a behaviour model that depicts the flow of system activities when the system executes. UML activity diagrams are commonly used for this purpose. For example, an activity model may be created to show the flow of activities from opportunity identification to the making of a quotation:

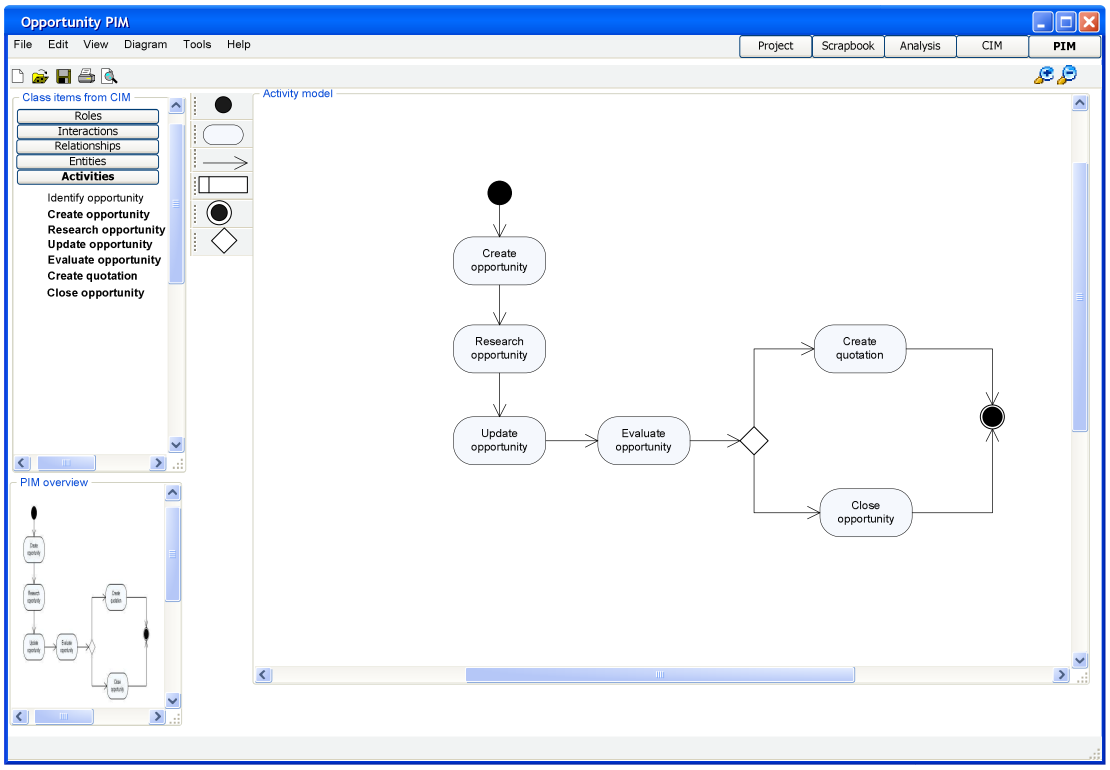
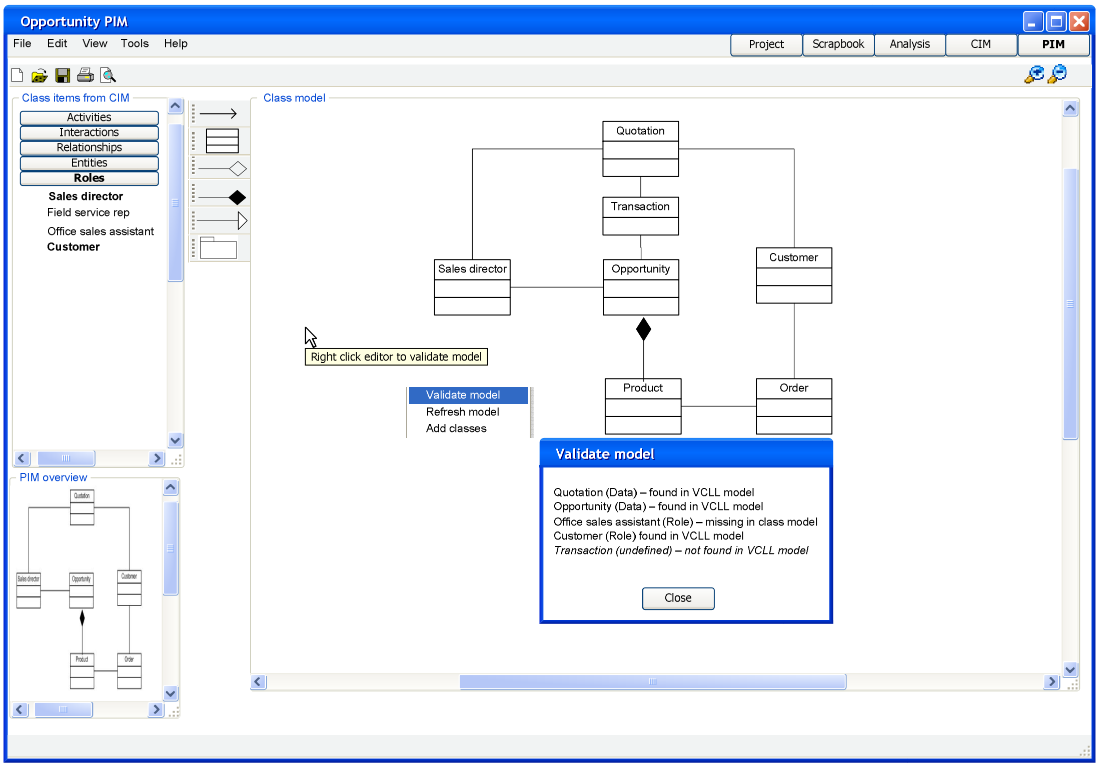


Figure 71: Behaviour model for opportunity management

In Figure 71, activities that are in bold appear so because they have been selected for creating the behaviour model. One activity that is not considered as a possible system activity is the "identify opportunity" activity, and this has been left out in the activity chart.

8.2.2.2 Validating models

Suppose that a user wants to check a model (e.g. class model) against a source (e.g., CIM) model from which some elements might have been derived. Right-clicking on the editor area provides a pop-up menu from which a user can select the model validation option as shown in Figure 72.



Quotation (Data) – found in VCLL model
Opportunity (Data) – found in VCLL model
Office sales assistant (Role) – missing in class model
Customer (Role) found in VCLL model
Transaction (undefined) – not found in VCLL model

Figure 72: Model validation

In Figure 72, most of the classes are related to specific elements in the VCLL model, except one class, namely *Transaction* which is not derived from an element of the CIM model. Such validation can be undertaken out for either the activity chart, or the CIM model itself.

8.3 Summary

This chapter has outlined the specification and functionality of the definitive prototype based on our experiences with the exploratory prototype and the subsequent feedback that we have received. The original Description of Work did not allow for the building of any of this interface. However, it is expected that some of the core functionality will be demonstrated as part of Deliverable 9.1.

9. Visual Code Editor for State-Visualisation Syntax[1]

The state-visualisation variant is intended for software designers (as defined in D1.1). They use the VIDE editor for modelling the first level of behaviour, but leave the details to be refined in later stages. Because of their strong background in conceptual modelling and UML class diagrams, not only the graphical notation but also the user interface they apply for creating their models is different from the VIDE programmers' interface. That user interface is oriented at state-of-the art modelling GUIs as present in state-of-the-art IDEs, in particular Eclipse.

9.1 Requirements

6.1.1 Relevant Requirements from D1.1

The requirements summarised in deliverable D1.1which are relevant for the design of the visual code editor are as follows. We respectively indicate how they are reflected in design decisions.

- **REQ – User 1 Flexibility and interoperability of VIDE language and tools (SHOULD).** This requirement is reflected by the fact that the editor will be able to interoperate with the development tool Eclipse and its EMF, GMF, and GEF frameworks. It is smoothly integrated by referring to existing plugins into Eclipse such as existing UML editors.
- **REQ – User 2 Reuse of UML Standard (SHOULD).** The editor not only re-uses the UML standard by using notations known from UML instance diagrams but also integrates into existing UML tools.
- **REQ – Lang 4 Compliance with Standards (SHOULD).** The editor complies with the de facto standards EMF and GMF.
- **REQ – Tool 1 Usage of Industrially Adopted Tools (MUST).** The editor the industrially adopted meta-modelling standards EMF and GMF.
- **REQ – Tool 2 Meta-modelling Framework (MUST)**.VIDE uses EMF as its modelling framework.
- **REQ – Tool 7 Graphical modelling Framework (SHOULD)** VIDE SHOULD use GMF as its graphical modelling framework.
- **REQ – Tool 8 Use of OCL (SHOULD). The** VIDE editor uses OCL as expression language as prescribed by the state-visualisation notation.
- **REQ – Tool 11 Framework for CIM, PIM, PSM modelling (SHOULD).** The editor uses EMF as its framework for PIM modelling and adopts EMF as the meta-modelling framework.
- **REQ – Tool 13 Integration and metadata interchange (SHOULD).** VIDE provides model and meta-data interchange capability by adopting the XMI standard coming for free from using the EMF and GMF framework.
- **REQ – Tool 14 Model driven approach (MUST).** The VIDE tool strictly follows a model driven approach as supported by employing EMF. Moreover VIDE itself is, by employing GMF, developed in a model-driven way.

[1] For the text in Sect. 8 (the description of the visual code editor for state-visualisation variant) the following copyright disclaimer holds:

Copyright 2007 SAP AG. All Rights Reserved.
No part of this publication may be reproduced or transmitted in any form or for any purpose without the express permission of SAP AG.
The information in this document is proprietary to SAP AG. No part of this document may be reproduced, copied, or transmitted in any form or for any purpose without the express prior written permission of SAP AG.
This document is a preliminary version and not subject to your license agreement or any other agreement with SAP. This document contains only intended strategies, developments, and functionalities of the SAP® product and is not intended to be binding upon SAP to any particular course of business, product strategy, and/or development. Please note that this document is subject to change and may be changed by SAP at any time without notice.
SAP assumes no responsibility for errors or omissions in this document.
SAP does not warrant the accuracy or completeness of the information, text, graphics, links, or other items contained within this material. This document is provided without a warranty of any kind, either express or implied, including but not limited to the implied warranties of merchantability, fitness for a particular purpose, or non-infringement.
SAP shall have no liability for damages of any kind including without limitation direct, special, indirect, or consequential damages that may result from the use of these materials. This limitation shall not apply in cases of intent or gross negligence.
The statutory liability for personal injury and defective products is not affected. SAP has no control over the information that you may access through the use of hot links contained in these materials and does not endorse your use of third-party Web pages nor provide any warranty whatsoever relating to third-party Web pages.

6.1.2 Refined User Adequateness Requirements

We infer in particular the following requirements for the state-visualisation editor targeted at software designers (see Deliverable D2.1):

1. Close integration into existing (UML) modelling tools. For instance:
    a. Sharing model elements across the boundary of different modelling types should be possible.
    b. Dragging an action from an activity diagram editor into the VIDE editor and refinement there should be possible.
2. Since VIDE models target also people who are more familiar with textual code, textual display and modifications of the models should be allowed both in graphical mode and textual mode.
3. The guidelines of the chosen platform are respected in order to guarantee a uniform user experience[61]. State-of-the-art ways to interact with the editor like in existing modelling tools of the chosen platform should be available in VIDE, as for instance:
    a. Drag & drop from a palette;
    b. Autocompletion and syntax highlighting when editing text for expressions;
    c. Textual input instead of mouse gestures;
    d. Drill-down, collapse/expand features (to cope with scalability issues of visual languages);
    e. Property pane (for displaying and editing additional information for which there is no place in the graphics);
    f. Free positioning of model elements;
    g. Standard Import/export mechanisms (e.g. to graphics format)

9.2 Eclipse Concepts for VIDE

In this section we describe the concepts of the target platform of VIDE, i.e. Eclipse together with the GEF/GMF frameworks, as chosen during the work on WP1, w.r.t. graphical user interfaces. We focus on features relevant for VIDE and describe where and in which way the VIDE editor for the state-visualisation syntax makes use of these concepts.

9.2.1 GMF/GEF Editors

The Graphical Editing Framework (GEF) is a framework which allows for creating a rich graphical editor from an existing application model within the Eclipse Modelling Framework EMF. Many common operations are defined by GEF, which can be adapted for specific needs. The Graphical Modeling Framework (GMF) allows developers to generate GEF compliant code from a declarative specification of the domain model, the graphical model, and a mapping definition, and provides a runtime infrastructure.

9.2.1.1 EMF

EMF is the Eclipse Modelling Framework for the creation of tools based on structured models. It facilitates the access to models in terms of an object tree, which can be modified by means of a Java API. For this purpose, EMF generates a set of classes for the model, that is, the Java classes represent meta-model elements. XMI files serve as input to EMF

9.2.1.2 Graphical Editing Framework (GEF)

Much functionality of graphical domain-specific editors remains the same across the domains, as for instance, loading and storing of models. The Graphical Editing Framework (GEF) [134] is a framework which makes creation of graphical editors significantly easier by taking over automatically generating the routine tasks. The features of GEF are the graphical representation of arbitrary data models, support for tool palette for editing, support for graphical zoom, printing, copy&paste etc. The architectural basis of GEF is the Model-View-Controller pattern (MVC). Views are provided by *figures*. These use the graphics API *Draw2D*, an abstraction layer on top of SWT, allowing for customized graphical components. According to the MVC pattern, a figure playing the role of the View does not have knowledge about the model which it represents the data for. The controller part is played by *Edit Parts*, coordinating the data flow between model and view. The model is provided by the application. It is required that the application supports the observer pattern in order to notify the controller about changes of the model, which is fulfilled when using EMF. For realising interactive diagram editors there is currently no solution which integrates as good in Eclipse as GEF [135].

9.2.1.3 Graphical Modelling Framework (GMF)

Though the Graphical Editing Framework (GEF) eases the development of graphical editors significantly, the Graphical Modelling Framework (GMF) goes a step further. It automatically generates graphical editors for

EMF meta models. The output of GMF serves as input for GEF [136].The first step in the GMF based development of an editor is the creation of an EMF model. Then three further “configuration models” are needed [137]:

- a model for the definition of the graphical elements (the GEF figures),
- a model for the definition of the palette (“tooling”), and
- a model for binding the graphical elements to elements of the meta model (“mapping”).

After having defined these models, the editor can be generated “by push-button”. By using special extension points [138], the editor can be modified manually while being kept up-to-date by subsequent modifying changes to the configuration models.

9.2.2 GMF/GEF Graphical User Interface Components and VIDE specifics

Plante [139] describes several entities which are available when using the GEF/GMF framework to create a graphical editor. Since VIDE and in particular the editor for the state-visualisation diagram follows this framework, also the supported entities will be present in the editor. For the VIDE user this eases the use and facilitates the transition from other modelling tools based on this framework to VIDE because of the well-known user experience. In the sequel we follow the overview by Plante,[139] in order to reflect the standard elements of the VIDE state-visualisation editor.

9.2.2.1 Eclipse Views and Perspectives

When working with Eclipse the user sees exactly one *perspective* consisting of several *views* [140]. A view is displayed in a rectangular part of the screen when Eclipse. Typical views are a code editor, an explorer for the project structure, a display for occurring errors, etc. Perspectives are containers for a set of views, though arbitrary views can be added while a perspective can be opened.

The VIDE editor consists of a VIDE perspective. The VIDE perspective contains at least the following views:

- The graphical editor pane where VIDE models are displayed graphically in the state-visualisation syntax.
- A message view where error messages are displayed.
- A property view where information on model elements are displayed which would otherwise scatter their graphical representation.
- A project explorer displaying the project and model structure as in the standard Eclipse perspective.

Views can be placed arbitrarily on the screen. By default, the arrangement in Figure 73 is proposed:

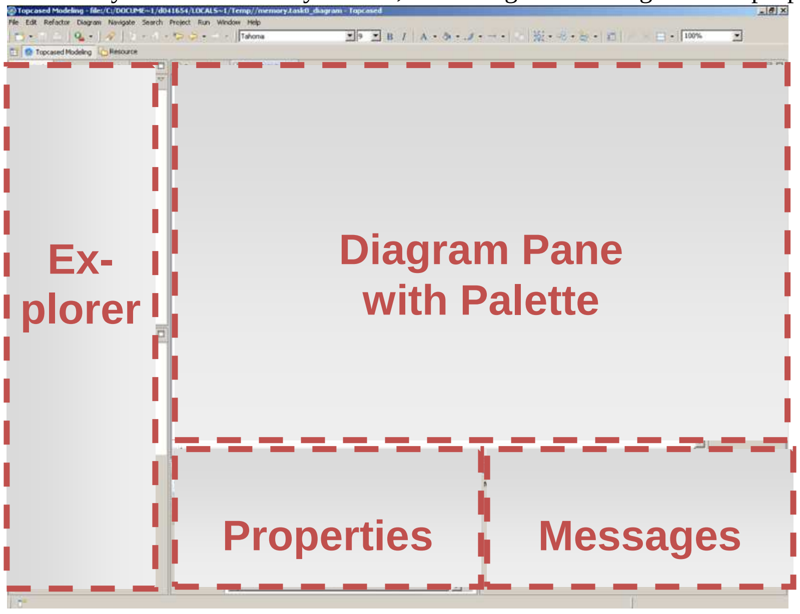


Figure 73 : Default views

9.2.2.2 Diagram Pane

The diagram pane is the central part of the editor where the model is assembled. The pane is arbitrarily large and the user may scroll horizontally and vertically to find space to place model elements. Placing model elements is done by "drag&dropping" model elements with the mouse, either those

- which are already placed on the diagram pane,
- which are to be created by dragging them from the palette (see below), or
- which exist but do not occur on this particular diagram by dragging them from the model tree.

Model elements are allowed to be freely resized by dragging the frame of a model element.
A grid can be displayed in the background of the pane in order to facilitate well-aligned manual layouts. Optionally placing a model element on the pane triggers a process of aligning the model element to the grid ("snap to grid feature").All of this is enabled in the VIDE editor for the state-visualisation syntax.

In addition to the – quite standard – features from above, the GEF/GMF framework offers more advanced features discussed below.

9.2.2.2.1 Context Sensitive Buttons

Context sensitive buttons (Diagram Assistants, [139] allow model editors to select only the relevant model elements when moving the mouse over a model element. Like a tooltip a graphical menu is displayed after a short delay. GMF offers two kinds of such buttons: Pop-up Bars and Connection Handles.

9.2.2.2.1.1 Pop-up Bars

Popup-Bars are buttons displayed in a bubble-like shape. GMF devotes these buttons to create sub-elements of the diagram element which is selected by positioning the mouse pointer over it. For structured VIDE actions, such as SequenceNodes, such a button offers the creation of some common standard actions such as AddStructuralFeatureActions or OperationCallActions. Which actions are displayed at this point will be a matter of more experimentation when a first prototype is available. It is definitely counterproductive if all possible actions are offered since this would be too overwhelming for a user. In case that a more rarely used model element should be added the user should use the palette (see below).At conditional nodes popup-bars allow the user to insert additional clauses, since – by default – the palette entry introduces a fixed number of clauses.

9.2.2.2.1.2 Connection Handles

Connection Handles are buttons for adding incoming and outgoing connections to / from the selected object. The handles are symbolised as floating arrow shape buttons. By double-clicking or dragging the mouse to the other model element the creation of the connectors is initiated. If there are several alternative connection types a menu is displayed. The user has, again by a menu, the possibility to trigger the creation of a new model element of a certain type. In the state visualisation variant of VIDE there is only one connection type needed, depicting the control flow from one action to the other. When a user thus selects, for instance, an outgoing connection handle, he or she can either drag the connection end to an existing model element which can be a subsequent action according to the VIDE language and drop it there, or drag the connection end and drop it on the diagram pane or double click on the connection handle; in both of these cases a menu pops up, which asks for the model element to be created and to be connected with the new model element. The following pictures illustrate this behaviour.

9.2.2.2.2 Direct Editing and Autocompletion

When referencing existing model elements and when this reference works by indicating its name by typing, the direct editing and autocompletion features of Eclipse may be used [139].

When clicking on a label of some model element, the label text can be edited directly. In addition to that, a menu is displayed which allows for the selection of existing model elements. After confirming the input an existing model element fitting to the chosen text is put in place.

In the VIDE context we may illustrate this behaviour for an AddStructuralFeatureAction. When a user clicks on the structural feature label of such an action it may directly edit the name of the structural feature the action refers to. (see Figure 74).

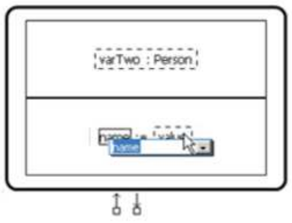


Figure 74: AddStructuralFeatureAction showing name editor

Moreover all structural features of the object indicated in the AddStructuralFeatureAction are displayed on request.(see Figure 75):

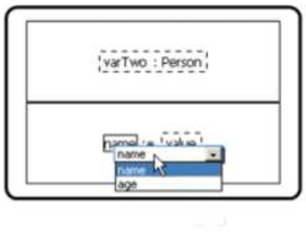


Figure 75: AddStructuralFeatureAction showing object structural features

After confirming (by clicking on an other model element or by pressing the return button) the structural model is looked up for a fitting model element. In case that such a model element is found the model is changed accordingly. If it could not be found an error message is displayed and the model is not changed.

The input of text according to the language defined for the state visualisation variant requires expressions conforming to OCL syntax, and in general even *any* VIDE statement can be put into an action box. Since such input follows the VIDE textual syntax, syntax highlighting is available and is desired to be displayed when entering the VIDE statements or expressions. Furthermore strong typing facilitates allow that autocompletion is available while entering the text. It is available as described before by reusing functionality of the textual editor.

9.2.2.3 Palette and its Entries

A GEF/GMF based editor has a palette bar popping up when the user touches the right border of the diagram pane as shown in Figure 76. Here the modeller can

- select a model element type; by subsequently clicking or marking an area on the diagram pane a new model element of the selected type is created and graphically displayed at the specified position and (in case of marking an area) in the specified size.

- choose common model-independent tools. By default there are
  - The selection tool, which allows for selecting a model element or, by marking a whole region, a number of model elements
  - A note tool, which allows for creating a text box without formal modelling content and for attaching informal notes to model elements

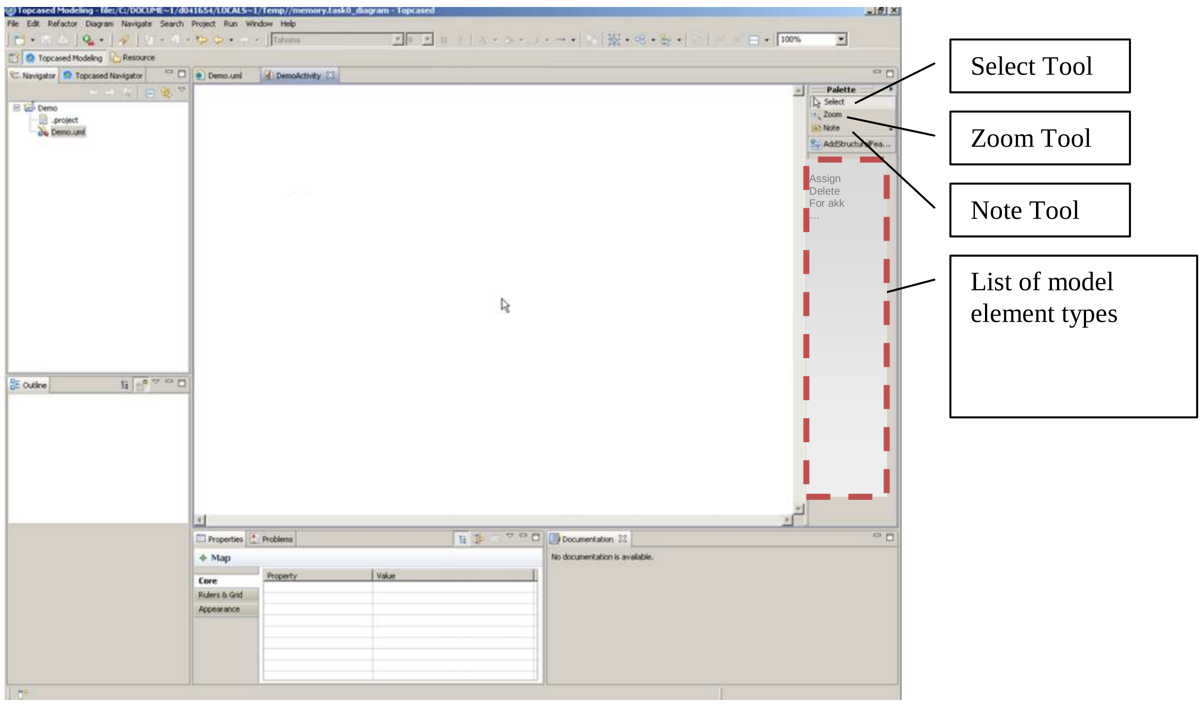


Figure 76: A GEF/GMF based editor showing palette

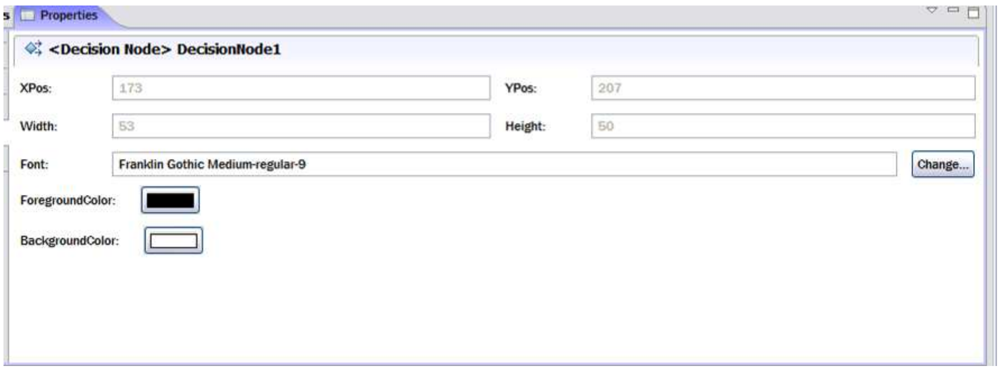
In the VIDE editor, the model elements which can be selected are patterns built around the basic UML actions. The technical names are however hidden from the user in the following way:

- Structural Features
  - AddStructuralFeatureValueAction → Assign Feature
  - ClearStructuralFeatureValueAction → Remove All Feature Values
  - RemoveStructuralFeatureValueAction → Remove Feature Value
- Control
  - ConditionalNode + Clause(s) → If-then-else
  - ExpansionRegion → For Each Element
  - LoopNode → Loop
  - CallOperationAction → Call Operation
  - ReplyAction → Reply
- Exceptions
  - ExceptionHandler → Handle Exception
  - RaiseExceptionAction → Raise Exception
- Objects
  - CreateObjectAction → Create Object
  - DestroyObjectAction → Destroy Object
- Links
  - CreateLinkAction → Create Link
  - DestroyLinkAction → Destroy Link
  - ClearAssociationAction → Destroy All Links
- Variables
  - AddVariableValueAction → Assign Variable
  - ClearVariableValueAction → Remove All Variable Values
  - RemoveVariableValueAction → Remove Variable Value
- Groups
  - SequenceNode → Block
- Connections
  - ControlFlow → Flow

The model elements created when selecting the palette entries do not only comprise the model element listed above for the respective entry, but also "surrounding" elements. For instance, when selecting AddStructuralFeatureValueAction not only an instance of that meta-class is created but also two input pins, one for the object and one for the value to be processed. As another example, for a conditional node, clauses are created as well. By default 2 clauses are created, while further can be added using context sensitive pop-up bars (see above).Entries in the palette may be grouped. For VIDE, the groups as indicated in the list above are in place.

9.2.2.4 Property View

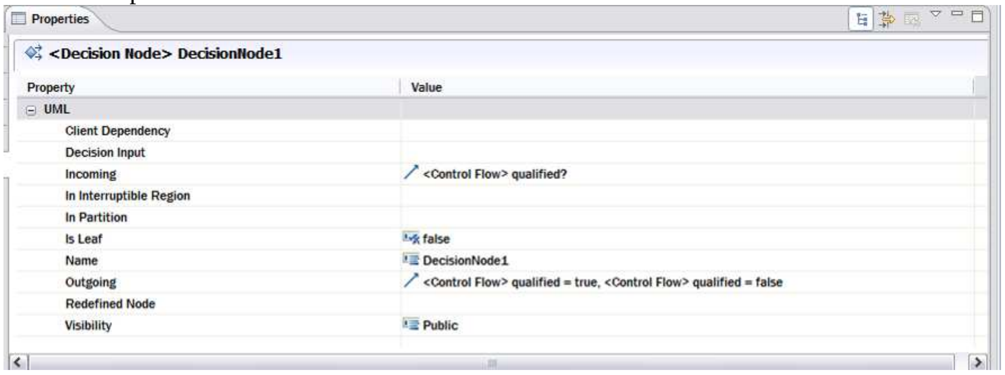
GMF offers a default implementation for the editing of appearance related attributes applied to diagram elements using the Properties View as shown in Figure 77.



XPos:	173	YPos:	207
Width:	53	Height:	50
Font:	Franklin Gothic Medium-regular-9	Change...	
ForegroundColor:	■		
BackgroundColor:	□		

Figure 77: Property View

VIDE should also offer a custom property view for each model element, in which all attributes defined by the underlying UML metaclass can be edited. This property view will certainly not be used by designers targeting at a coarse model because it is quite technical, but may be filled in by implementers targeting at an executable model without modifying the graphical elements. Figure 78 illustrates the appearance of such a custom property view as realised in Topcased.



Property	Value
UML	
Client Dependency	
Decision Input	
Incoming	<Control Flow> qualified?
In Interruptible Region	
In Partition	
Is Leaf	false
Name	DecisionNode1
Outgoing	<Control Flow> qualified = true, <Control Flow> qualified = false
Redefined Node	
Visibility	Public

Figure 78: Example of a Property view for a UML action (Screenshot taken from Topcased)

9.2.2.5 Problems View

GMF provides a default view for displaying errors or warnings of the models. VIDE will use this and advanced ways to interact about problems with the user as described in Deliverable D4.2.

9.2.2.6 Other Concepts

Plante [139] lists standard concepts supported by each GMF based editor. The VIDE state-visualisation editor thus has a number of features which are available as toolbox entries, as menu entries, or context menu entries:

a) Means for adding graphical hints with vague semantics or allow for manual beautifications:
   (a) Use of custom font by a font menu for every diagram element
   (b) Custom fill colours and line colour for the selected diagram element's interior and lines respectively
   (c) Custom line style for modifying the routing style of the selected diagram connector elements (e.g. rectilinear, oblique, tree style routing)

   These are properties which can configured by the appearance related property view as described above.

2) Means for selecting and arranging diagram elements:
   - Select: selects all diagram elements, all shapes, or all connectors.

- Align: aligns all selected diagram elements to: the left, the right, the top, the bottom, or the centre of the selection.
- Auto Size: resets the size of the selected diagram elements to the default.
- Make Same Size: sets the size of the selected diagram elements to that of the last selected element.
- View: shows or hides various diagram features: ruler, grid, page breaks, and the snap to grid behaviour.
- Zoom: changes the diagram zoom in, out, 100%, To Fit, To Width, To Height or To Selection.
- Print and Print Preview including Enhanced Print Dialog, Global or per Diagram Page Setup Options, Page Breaks
- Graphics Export: Exports diagrams in common formats, e.g. SVG, GIF, BMP, and JPEG
- Undo/Redo: The last performed action can be made undone by a keystroke; after undoing a one keystroke redo of the undone action.
- Clipboard: The system clipboard is supported for importing and exporting model elements by copy/cut&paste.

All these features make up a modern graphical editor which modellers are accustomed to. Fortunately, due to the selection of GMF as framework, these features all come for free, and thus do not cause implementation effort, while satisfying modellers' needs.

9.2.2.7 Deviations from Standard Eclipse GMF/GEF

The following features of a standard GMF/GEF editor are disabled or modified in VIDE:

- The Order command of GMF/GEF re-orders the selected diagram elements to the front, the back, forward once, or backward once. Layering of model elements is fixed in the VIDE editor, thus this feature is disabled.
- The standard Arrange button of GMF/GEF applies a simple layout algorithm to diagram elements. We realise a much more advanced domain-specific autolayouting functionality, which is described below.

9.3 Challenges and Approaches

In this section we list challenges of the VIDE approach with respect to the user interface and potential solutions we have investigated.

9.3.1 Reduction of Graphical Complexity

Modelling graphically potentially may provide a better overview over a piece of the model because of a more intuitive representation of control structures, such as conditionals or expansions. The advantages of graphical modelling are however restricted for a number of reasons:

- Large models do not fit on one screen, thus the overview aimed at is lost.
- Complex graphical models, with lots of connections, rather confuse than help understanding.
- Many developers are used to textual (code like) representation of behaviour, rather than graphical.
- There is apparently no good way to depict queries and expressions graphically in general.

It is thus the more important to maintain the advantages of both ways for accessing models, i.e. textual and graphical, while ruling out the disadvantages. We must aim at a suitable combination between both approaches. Suitable instruments are discussed in the following sections:

- Collapsing graphical model elements and showing a textual representation instead.
- Hiding some model elements completely.
- Enhanced navigation capabilities.

9.3.1.1 Collapsing and Expanding Nodes

GMF provides the functionality to collapse or expand composite figures within the editor by clicking on an icon [139].
This is illustrated at the example of the ExpansionRegion model element in Figure 79.

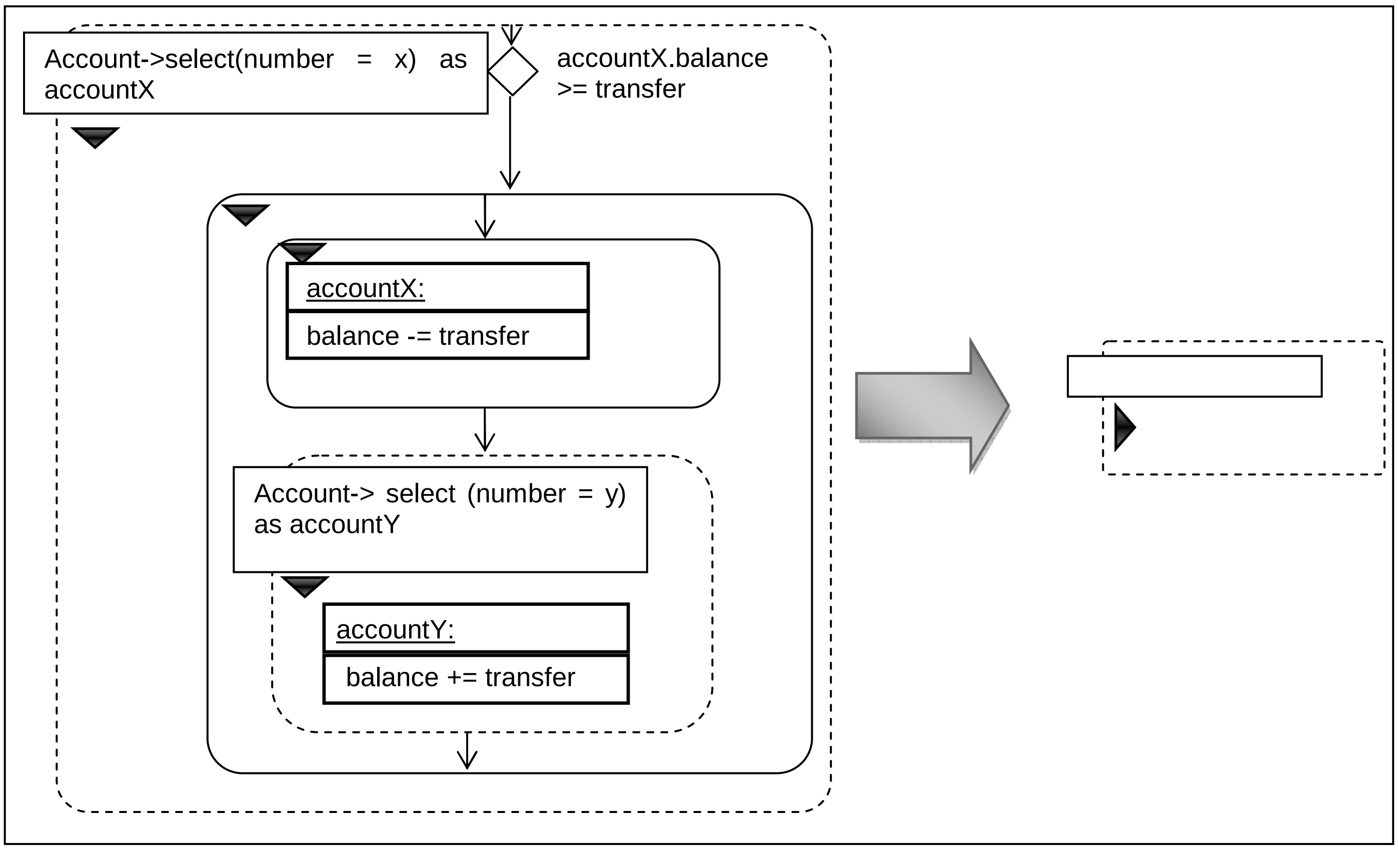

Account->select(number = x) as accountX
accountX.balance >= transfer
accountX:
balance -= transfer
Account-> select (number = y) as accountY
accountY:
balance += transfer

Figure 79:Collapsing an Expansion Region

The mouse click triggers the disappearance of contents inside the figure or a compartment of the figure. By default the figure is not (as stipulated in Figure 79) resized nor is the layout of surrounding figures modified. We discuss this and other problems in Sect. 9.3.1.1.4. First we show how important collapsing and expanding is in the context of the graphical VIDE editor.

9.3.1.1.1 Use Cases

There are a number of use cases where the collapsing and expanding functionality is crucial in the context of VIDE:

1) A modeller is editing or reading a part of a behaviour description and is not interested in behaviour happening before that part nor in that after that part. In that case only the construct to be edited is expanded, all other parts are collapsed.
2) A modeller is interested in a coarse overview on the behaviour only. Details are irrelevant in order to obtain the overview. Then all irrelevant parts can be hidden by collapsing the containers. The collapsed view of the model element does not necessarily need to (formally) reflect the whole semantics of the element. Instead of its contents just a comment in natural language or even nothing except the name/type of the model element may be displayed.
3) VIDE relies on the capability to edit in textual and in graphical mode simultaneously. When displaying the whole behaviour graphically, but only some parts textually, a hybrid view is needed, where the textual part can be seen as an activity which is collapsed. Thus, a user might add behaviour textually and then use the expand functionality to display and check the graphical representation.

9.3.1.1.2 Collapsing Modes

From the use cases listed above we can infer 3 modes of a collapsed node:

1) Inside the collapsed node nothing is displayed; such behaviour would be useful in 1) and 2).
2) Inside the collapsed node the name (if applicable) or the type of the node is displayed; again such behaviour would be needed in 1) and 2).

3) A user provided textual comment (in natural language) is displayed inside the collapsed node; such behaviour is applicable for 2).

Note that for 3), the collapsed node needs to display the textual VIDE statement.

Figure 80 shows the different options for collapsing nodes for the example of Figure 79.

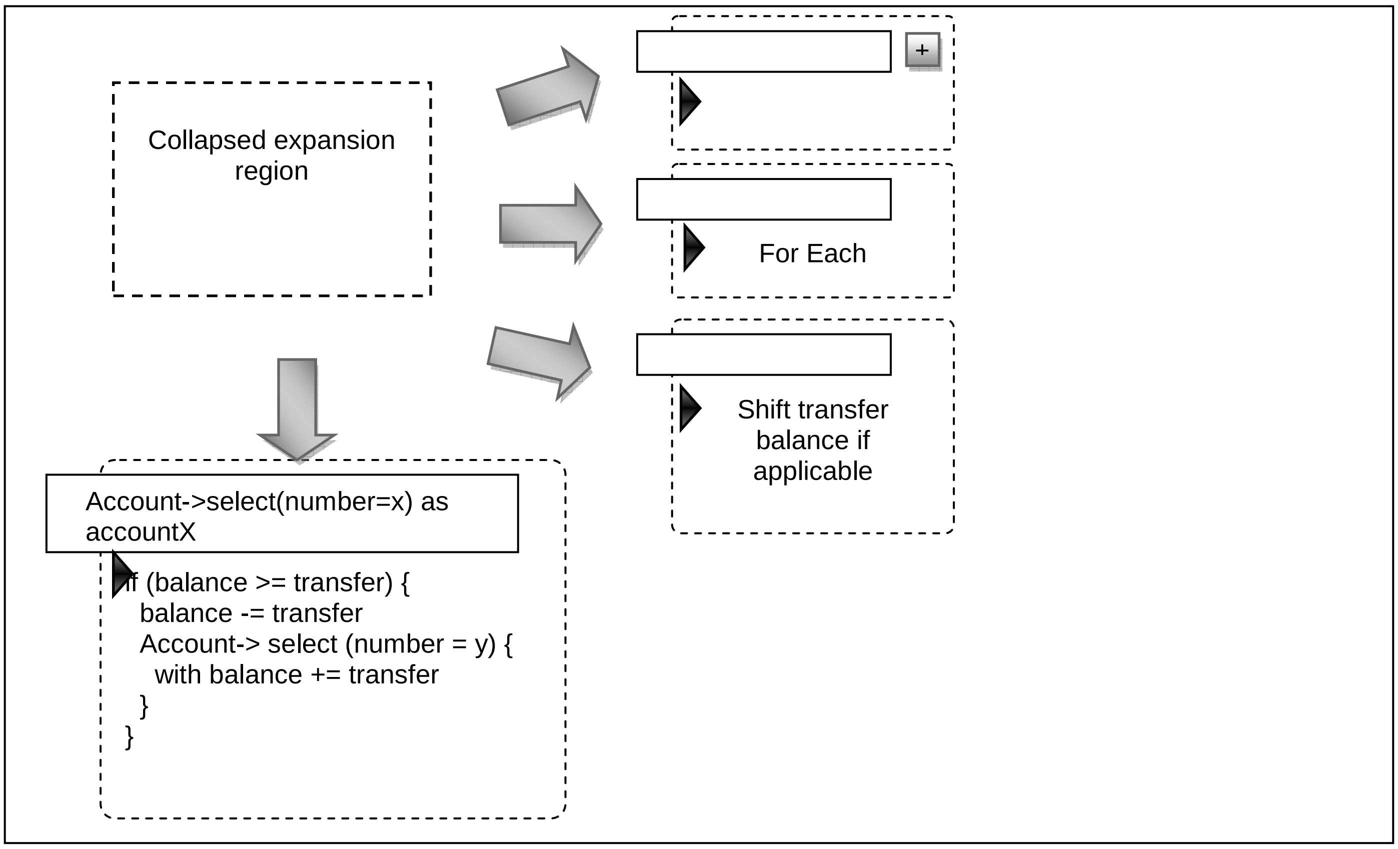

Collapsed expansion region
+
For Each
Shift transfer balance if applicable
Account->select(number=x) as accountX
if (balance >= transfer) {
balance -= transfer
Account-> select (number = y) {
with balance += transfer
}
}

Figure 80: Expanding at the example of Expansion region

The different styles of collapsing are offered to the user by providing different buttons for collapsing, these are either offered as buttons or in the context menu displayed on mouse right-click.

9.3.1.1.3 Automation of Collapse/Expand

According to 1), a modeller might not be interested in behaviour displayed in a model element of the same hierarchical level as the one edited which appears before or after that focussed element. The editor will automatically collapse these surrounding model elements.

Such automatic features might be annoying, since in certain cases the appearance of surrounding elements might – contradicting Use Case 1 – be desired. It is thus necessary that this feature is optional. Its usefulness can only be judged after extensive user oriented evaluation.

9.3.1.1.4 Problems

There are a number of problems with collapse/expand functionality:

1) When the interior of a compound node is collapsed the size of the node must be minimised. Otherwise a lot of distracting white space would be displayed, and this would destroy the effect that collapsing leads to a better overall picture. Minimising the size however affects the objects located around the collapsed node, they must be newly arranged.
2) When expanding a collapsed node the node must necessarily be enlarged. Objects in the neighbourhood of the expanded node are potentially in the way und must be moved.
3) Orthogonal to these two items the coarse layout of all components should be kept unmodified when collapsing or expanding, in order not to confuse the modeller who is performing the operations.

All these problems require use of autolayout, which is specific to the modelling language and incremental. This issue is discussed in Sect. 9.3.4.

9.3.1.2 Hiding/Showing Nodes

We have already discussed that modellers might be interested only in the container node they are currently be modifying or looking at. Instead of hiding the interior of the nodes which are not in focus, we may hide the nodes completely. Note, that in this case, information to understand the entire model disappears completely; no "substitute" representing the hidden elements textually is displayed. We thus offer modellers, the options "Hide Actions After" and "Hide Actions Before", on each model element. These hide elements which appear in the control flow before (or after, respectively) the selected model element, unless they contain the selected model element. Moreover we might allow for hiding nodes of a certain type or semantic annotation. For instance – in order to get a better general picture on the interaction with the behaviour's outside – we could allow users to hide all nodes but operation calls.

9.3.1.3 Enhanced Navigation Capabilities

A graphical environment allows for easier navigation by "drilling-down" into a model element or navigating to its definition. The context of the originating diagram is lost and thus the elements not focussed on do not confuse. The graphical editor should support enhanced navigation capabilities as follows:

- By double-clicking a compound node, the same functionality as for "Hide Actions After" and "Hide Actions Before" should be achieved. However, in addition, the surrounding box of the compound node disappears.
- By double-clicking an action representing an operation call, the context is changed completely and the diagram describing the behaviour of the clicked method call is displayed.

9.3.2 Simultaneous Textual and Graphical Editing

VIDE is supposed to enable simultaneous textual and graphical editing. Thus a user may read and edit the same behaviour textually as well as graphically.From a user interface point of view, we provide a way to switch between different editors. Performing this switch can be realised in several ways:

- Within an opened editor right-click the background and obtain a context menu where "Select in … Editor" can be selected
- From the explorer tree right-click on the entry for the respective method and choose "Select in … Editor".

Here … can stand for "textual" or the name of one of the graphical editors (i.e. state-visualisation, linear"). Moreover there is a hybrid way to editing VIDE behaviour textually within the graphical state-visualisation editor as already mentioned above. For this, a generic action depicted as a rounded rectangle is provided which contains the text.

9.3.2.1 Text Input Support

When editing text within the generic action, the same functionality should be available for the user as in textual editors:

- Syntax highlighting
- Direct input
- Autocompletion

9.3.2.2 Need for Autolayout

When a user has edited behaviour textually and switches to the state-visualisation editor the layout of the edited parts is not determined. Thus no coordinates for these parts are fixed. If the edited part is large it cannot be expected from the user that he moves the model elements manually until an appropriate layout results. This holds especially if the one who has edited textually and the one who displays have different roles, such as a modeller (who creates the model) and a business expert (who reviews it).

9.3.3 Interconnection with other Editors

Modellers usually work with a set of tools, however often integrated in a common workbench, such as Eclipse. For instance a modeller typically creates a structure diagram with tools like Rational Architect or Topcased and then defines behaviour with VIDE. We assume it as to be typical that the iteration cycles between creating/modifying structure and creating/modifying behaviour is rather small, i.e. modellers change structure according to the experience they have made while modelling behaviour and vice versa.With this assumption in background we infer that a tight tool integration between VIDE and the other tools in the workbench is needed. This is manifested in the functionality that

- users may navigate between editors/viewers depicting a model element used in VIDE and the VIDE editor, and
- we allow for dragging elements from the structure diagram of any EMF based editor and dropping it on the diagram pane of the VIDE state-visualisation editor

This functionality is described in the remainder of this section.

9.3.3.1 Navigation

We allow for navigating from the VIDE state-visualisation editor to other EMF based editors and back.
We illustrate the behaviour at the case of operations and operation calls, since this is the most obvious one. There may however also be other possibilities, such as attribute access in form of AddStructuralFeatureValueAction for instance.

When being in a structure diagram editor, such as Topcased, it is desirable for a modeller to right click an operation of a class and navigate from an entry therein into a VIDE model of its behaviour. If there is no behaviour specified an empty VIDE model should be displayed. The modeller may then edit the model and jump back to the structure diagram. If the tool, i.e. Topcased in our case, offers an appropriate Eclipse extension point, it is sufficient to provide an entry in the context menu and an appropriate point of reference within the VIDE editor plugin. For coupling the plugins loosely we should provide a separate little “VIDE for Topcased” plugin depending on Topcased and VIDE.

The other way round, VIDE is supposed to provide at each operation call a context menu entry for jumping to the “definition” of the referenced operation in the Topcased tool.

9.3.3.2 Drag&Drop from Other Editors

Similarly as navigation, we provide drag&drop functionality from other editors into VIDE. In order to model a method call within the VIDE state-visualisation editor it is possible to drag a method call entry from the structure diagram editor such as Topcased and drop it on the VIDE diagram pane. The VIDE shape for depicting operation calls is inserted at the specified position. The modeller just needs to insert the arguments of the call manually.

9.3.4 Autolayout

9.3.4.1 The need for Autolayouting

We have already seen that autolayouting is a crucial part of an easy to use graphical model editor supporting for textual and graphical input. The following features require autolayouting:

1. Expanding and collapsing
2. Hiding and Showing nodes
3. Simultaneous textual and graphical editing

We can conclude that without a powerful autolayouting component it is impossible to realise the VIDE code editor.

9.3.4.2 Autolayouting Challenges

Autolayouting is a complex task since it

1. deals with many NP hard problems which often require heuristic non-optimal solutions
2. aims at a “beautiful” appearance; since beauty depends – in many cases – on the observer, it is often impossible to define clear metrics
3. aims at a layout which is optimal for a certain domain or a certain user group; there is thus no “general” solution to autolayout

9.3.4.2.1 Aesthetical Layout

As already mentioned, it is hard to define what is a „nice“ layout. There are several factors defined in literature [141, 142] which we mention briefly:

- Minimised edge crossings [142] produce a clearer picture of dependencies;
- Minimised bendpoints of edges;
- Orthogonal edge routing, and even edge routing along a grid;
- Edges should not be routed over nodes;
- Equal distribution of nodes in diagram space;

- There should be enough space between nodes in order to separate them visually; no overlaps between normal nodes are allowed; nodes must only overlap with group nodes they are contained in;
- Nodes connected by an edge should be as closely together as possible, in order to allow for proper edge routing and in to symbolise that the nodes belong together;
- Usage of diagram space should be minimal in order to provide a comprehensive overview on the whole model. This partially contradicts the other requirements, such as minimal edge crossings. By optimising with both goals, complexity of algorithms increases.

9.3.4.2.2 Domain Specifics

Being specific to a domain is crucial to meaningful autolayout. Models should be layout differently depending on the model type. For instance, VIDE models should likely be layout differently than pure activity diagrams. Thus, it is important that layouts can be adapted to the user's domain by configuring the layout appropriately.

9.3.4.2.3 Types of Layout Algorithms

There are a huge number of different layout algorithms available in literature, as well as being implemented in graph libraries (see below), such as layouts for trees, circular layouts, organic layouts, orthogonal layouts, etc. For VIDE we consider only hierarchical layouts and incremental approaches as relevant.

Hierarchical Layouts focus on highlighting the flow in directed graphs. They are well suited for modelling processes, workflows, and can thus be used in VIDE to layout the flow in a VIDE state-visualisation diagram.
Nodes are placed in hierarchical layers, i.e. consecutive nodes are always placed one after the other in flow direction unless the graph contains cycles. In that case the cycles are resolved as good as possible, leading to backward arrows. By ordering nodes within each layer other optimisations such as minimising of edge crossings is small.

Incremental Layouts. When developing models as in VIDE we often have to deal with extensions or small changes which should be displayed again. This leads to a sequence of graphs of a model. By laying out each and every model change from scratch it may be the case that a diagram is layout completely differently than before the (even tiny) change. In order to keep the „mental map" of a user (i.e. knowing where the nodes can be found), successive layouts of the same model should be similar to each other. In particular nodes and edges should be moved not too much and the general appearance of the graph should be kept as consistent as possible. This decreases the modeller's effort to understand the model after a change [143]. Layout algorithms that calculate similar layouts are called incremental or interactive. Figure 81 shows an incremental layout process where red nodes are newly added nodes.

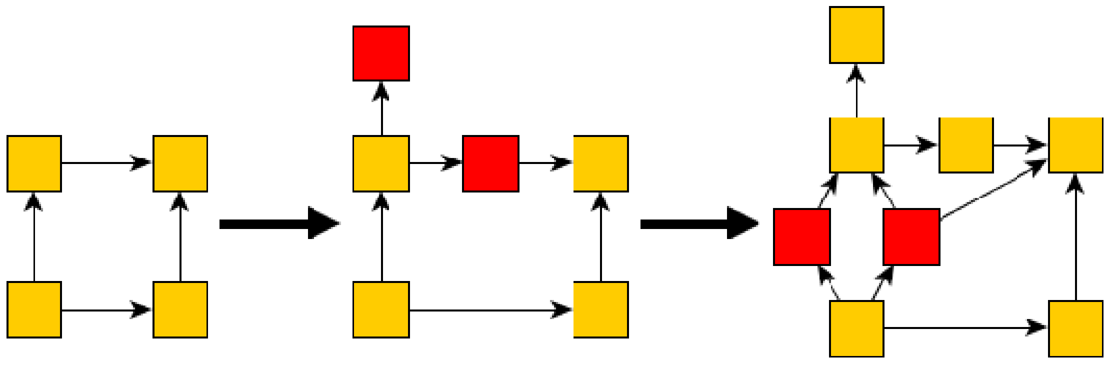


Figure 81: Example of incremental layout

For VIDE incremental layout is crucial: Assume we have created a VIDE model in the state-visualisation editor, switch to the textual editor and add a view model elements there. Then we move back to the state-visualisation editor, the new model elements should be layout incrementally, such that the modeller sees the old elements located approximately at the same place.

9.3.4.2.4 Libraries

We have investigated several libraries supporting autolayout:

- *prefuse* [144] is an extensible Java framework for the development of applications which visualise information in general. It offers a series of pre-defined layouting algorithms, among them circular layouters, tree layouters, etc. It does not support group nodes,which are required by the VIDE editor.

- *SugiBib* [145] is a graph visualisation framework developed at Universität Würzburg. The main algorithm corresponds to the Sugiyama-Algorithm [146]. Again, no group nodes are supported.
- *GraphViz* [147, 148] is an Open Source software package for drawing graphs. The built-in graph description language DOT only supports one hierarchical layout algorithm.
- *Tom Sawyer* [136] delivers components for analysing, layout and visualising of graph structures. The autolayout component contains algorithms based on algorithmic, as well as on interactive/incremental approaches.
- *ILOG JViews* [149] is a Java suite comprising a diagrammer tool for creating and modifying graphs. Several layout algorithms are supported, among them a layouter for hierarchies and trees. Group nodes and incremental approaches are supported for layouting.
- *yFiles* [150] is a Java library providing components and algorithms for analysing, visualising and laying out graphs. It supports a huge number of different algorithms which can be parameterised in many ways. It contains both incremental and hierarchical layout algorithms. Group nodes are supported.

Because of the unique amount of functionality we have chosen *yFiles* as the layouting functionality of choice to be used in the VIDE state-visualisation editor.

9.3.4.3 Autolayouting for VIDE

The basic layout rules of VIDE are as follows:
- The main layout flow is from top to bottom.
- Nodes contained in a sequence node or in the top-level activity are put one below the other in the order specified by the sequence or the control flow.
- Conditional nodes are layouted as follows:
  - If there is one branch: The subsequent sequence node is put below the decision node. The condition is put to the right of the decision node.
  - If there are two or more branches: One branch is put below the decision node, the condition is put to the right of the edge leading to the sequence node for the first branch; the other branches are put on the right next to the first branch, their condition is on the right of the edge leading to the sequence node of the branch.
  - Branches are aligned on top. The nodes of parallel branches are horizontally aligned in layers.
- Operation calls to operations offered by Web services should be in a separate column on the right of the main flow, in order to graphically distinguish these calls from normal ones. Edges to these model elements should be routed orthogonally.
- Newly inserted nodes should be layouted incrementally (as described above).

Moreover it should be possible to configure the layouting procedure declaratively by user defined rules. This would allow modellers to adapt the basic rules specially, such as to let clauses of a conditional flow from left to right if the number of nodes contained in the sequence node is less than 2.

Especially for adaptations of VIDE to domain-specific languages such custom layout rules have a great potential for many visualisation applications within SAP.

9.4 Conclusions

The state-visualisation editor adopts the user interface concepts developed for Eclipse [140] and GEF/GMF [139]. Since it is designed for software designers (see D1.1) familiar with conceptual modelling and UML class diagrams, this fosters their acceptance of the VIDE editor due to a similar way of interaction.
Challenges specific to VIDE that go beyond the standard concepts are
- reducing graphical complexity, which is addressed by collapsing and expanding, showing and hiding, and navigation functionality
- textual and graphical editing in parallel, which is addressed by using advanced autolayout features
- interconnection with existing editors of the platform, where we rely on functionality of the Eclipse platform and the common EMF core
- autolayout which is needed in several places.

We have stated requirements for the further development of the tool in WP9 and made investigations especially concerning the autolayout functionality which is crucial for the success of the VIDE approach.

10. Visual Expression Builder

Visual Expression Builder (VEB) is a visual mechanism offering users a way to specify their VIDE expressions using graphical metaphors. VEB is implemented as a graphical editor plug-in for the Eclipse environment and is used with the VIDE textual and graphical editors. User is able to specify VIDE expressions declaratively, in UML instance diagram fashion inspired by the Query By Example approach, called here Object Query By Example (OQBE).

10.1 Rationale

The VEB can be considered an addition to the core VIDE PIM language and editors foreseen during the early stages of the project. The base assumption of the language design was assuring the compliance with a common model repository and interchangeability of visual and textual syntaxes. The reasons of that approach can be summarized as follows:

- Using a common, standard compliant model repository allows for model interchange, eases the integration work of the toolset and possibility of supporting part of functionality by existing UML tools. It also eases the application of existing technologies for model transformation assumed by MDA.
- Availability of textual syntax is essential as this constitutes the most widespread and proven approach of specifying details of application behaviour.
- Development of visual syntax is important for achieving a better expressiveness of some constructs and for making the language more familiar to the users knowing the visual modelling tools based on UML.

The above assumptions make it easier to maintain the integrity of the language design and to allow flexible combinations of visual and textual representation of code – to suit the needs of various stakeholders dealing with software development. However, the very nature of visual syntax raises the expectations of achieving a more declarative and high level way of specifying the model.
Another observation was, in the VIDE PIM language, the OCL expressions constitute the most complex part, and thus can be considered a primary subject for improving the expressiveness in the visual syntax.
A good illustration of work towards that direction is the concept of condensed representation of expression described in Deliverable D2.1, section 5.3.3.5.6 "Condensed Representation by State Description Merging". Such condensed syntax provides a more conceptual and less redundant view of the expression, however, at the same time, its mapping onto VIDE abstract syntax is not straightforward and can be completed in various ways. Thus, as stated in D2.1, such representation could be made available only for read-only purpose.

10.2 Interaction with other tools of VIDE

VEB is integrated into a VIDE toolset through a PIM repository and through embedding its functionality into textual and visual VIDE PIM editors. As stated above, the notions of OQBE used by VIDE require the introduction of several metamodel constructs apart from UML/OCL metamodel that is the base of VIDE language.
The constructs introduced for VEB allow to represent a structure similar to UML instance diagram, and include the notions of Example (object example, link example), Condition, Sort criteria and Output. These constructs are connected with UML metamodel (mainly, with the constructs of UML Structures unit), to identify data sources (e.g. method parameters, attributes, variables) the expression will refer to, and their types.
The way the VEB functionality will be introduced into VIDE editors is described in the subsequent section in the context of textual editor. However, the integration with visual editor can be performed in an analogous way. The main ideas of such integration are summarized below:

- The option allowing specification of expression using VEB is available during the edition of a method body.
- Selecting that option provides the VEB with the context of the expression, that is, with the information on the attributes, operations and associations available from a current (self) object and the parameters and local variables of a given method.
- After the work with VEB is completed, a respective location inside method's body includes the VIDE code (visual or textual) produced from VEB's OQBE expression, as well as a link that makes it possible to enter VEB and update the expression through it.
- Editing the produced expression outside VEB (that is, using directly the textual OCL or its VIDE visual counterpart) is allowed, but invalidates the OQBE model elements of it, so that the VEB diagram is removed then.

Reading assistance:

- Every time number in the brackets is used (eg. (2)) it means reference to the diagram from section (@)1.1

10.3 Requirements

10.3.1 Relevant requirements from D1.1

The following table lists only the requirements relevant for Visual Expression Builder module. Omitting a D1.1 requirement in this table means it was found not relevant for the VEB module scope.

Requirement Number	**Name**	**Priority**	**Comment**
REQ – NonFunc 4	Clear and unambiguous notation – VIDE should have clear, comprehensible and unambiguous semantic description suited to the users of the VIDE tools	Should	VEB attempts to realize this requirement for PIM level notation by providing a visual metaphor that is fully consistent with widely known UML instance diagram notation. This improves the clarity of the constructs used to specify expression. A benefit to this extent is especially big in case VEB is used with textual VIDE editor.
REQ – NonFunc 5	Model view saliency – VIDE models views must be user-oriented.	Should	VEB provides an increased level of abstraction and allows the specification of query expressions by the users without the need of dealing with OCL.
REQ – NonFunc 6	Appropriate textual/graphical fidelity – VIDE must provide appropriate textual and graphical modalities for its users.	Should	VEB broadens the selection of means for specifying behaviour at PIM level. User working with textual or visual VIDE editor can switch to VEB to construct the expression in a way that is expected to be more intuitive for users with UML modelling background.
REQ – NonFunc 8	Runnable and testable VIDE prototypes	Should	VEB generates a pure VIDE model, hence the model execution features are available for VEB-specified code.
REQ – User 1	Flexibility and interoperability of VIDE language and tools - The VIDE language and tools SHOULD have flexibility and be interoperable with some existing tools.	Should	VEB abstract syntax provides several proprietary elements, however, they are designed to integrate with UML 2.1 standard metamodel and are transformed to OCL 2.0 standard metamodel instances.
REQ – User 2	Reuse of UML Standard – end users are very sensitive to using standards. A key aspect is that the VIDE language reuses as much as possible the UML standard.	Should	For VEB abstract syntax – see REQ – User 1. The concrete syntax used by VEB closely follows the syntax of UML instance diagrams.
REQ – Semantics 1	Semantics of VIDE Inte rnal Communication – a precise description of the semantics is needed sufficient for internal communication purposes within	Should	VEB constructs will be translated to the VIDE code. Hence, this requirement generates a need for precise definition of the transformation. This has been already provided in this document, however will require updates and refinement as the tool features evolve in the course of the final prototype development.

	implementation stakeholders in the development of the VIDE tool.		
REQ – Lang 1	Usage of UML2 Behaviour ("Action Semantics") – VIDE should use the behavioural model elements of UML2 (earlier known as "UML Action Semantics"), unless proven insufficient.	Should	VEB transforms its constructs to VIDE, which (as specified in D2.1) fulfils this requirement.
REQ – Lang 3	User Language & Concepts – the VIDE language and VIDE tools presented to a certain user groups SHOULD employ the language that is understood by the user group.	Should	VEB provides additional support for fulfilling this requirements by providing an additional utility which seems especially useful for users familiar with UML but having not much experience with OCL.
REQ – Lang 4	Compliance with Standards – VIDE should not compete with existing adopted modelling standards, especially those adopted by the OMG, such as UML or BPMN.	Should	As stated above – the constructs used by VEB are inspired by and integrated with UML 2.1.
REQ – Lang 6	Modularisation and extensibility – it should be possible to replace parts of the language with different artefacts and add additional language constructs for special business specific patterns. This requires the language to be structured in modules.	Should	The VEB is potentially capable of integrating with any UML 2.1 compliant editor for the purpose of building OCL expressions inside UML models.
REQ – Tool 1	Usage of industrially adopted tools – VIDE must use industrially adopted meta-modelling standards where applicable.	Must	As stated above, VEB maintains a compliance with UML 2.1 metamodel.
REQ – Tool 2	Meta-modelling Framework – VIDE must use EMF as its modelling framework.	Must	VEB is based on the EMF model repository and also uses related frameworks for the editor implementation.

REQ – Tool 3	Meta-modelling Concepts – VIDE meta-models should be constructed to be compatible with MOF concepts.	Should	Abstract syntax elements specific to VEB are expressed in terms of those meta-modelling concepts.
REQ – Tool 7	Meta-modelling Framework – VIDE SHOULD use GMF as it’s graphical modelling framework	Should	VEB is being implemented using GMF.
REQ – Tool 8	Use of OCL – VIDE should re-use existing standards as UML (REQ – User 1), and in particular OC;. the goal is to achieve a seamless integration with the concrete syntax of the action language to be developed.	Should	VEB produces OCL code.
REQ – Tool 9	CIM modelling standards.	May	Outside D2.1 scope. To be addressed in D7.1.
REQ – Tool 10	PIM, PSM modelling standards – VIDE SHOULD provide support for PIM modelling with UML and action semantics; the meta-modelling standard for VIDE should be Ecore. VIDE SHOULD support well known PSM modelling standards (e.g. XMI for model and meta-model interchange, JMI for Java based PSM).	Should	As explained above, this requirement is fulfilled by the VEB itself and by the VIDE language that is generated by VEB module.
REQ – Tool 11	Framework for CIM, PIM, PSM modelling – VIDE SHOULD adopt the ATL framework as its transformation framework, and should use XPAND for model to text transformations. VIDE SHOULD adopt EMF as its framework for PIM modelling VIDE SHOULD adopt EMF as the meta-modelling framework.	Should	VEB fulfils relevant part of this requirements as it is based on EMF.

REQ – Tool 12	VIDE extensibility	Should	Outside D2.1 scope. To be addressed by D9.3.
REQ – Tool 13	Integration and metadata interchange – VIDE should provide model and meta-data interchange capability by adopting the XMI standard.	Should	The choices with respect to mapping VIDE onto metamodel meet that requirement.
REQ – Tool 14	Model driven approach The VIDE tool strictly follows a model driven approach as stipulated in figure 9 page 120 of the D.1.1 deliverable	Must	VEB fulfils this requirement. The tool works inside the PIM layer and produces the code that is standard compliant.

Table 24: Requirements for Visual expression Builder

10.3.2 Refined requirements

The requirement refinement relevant for the VEB can be described in the form of a single requirement, already outlined in the introduction of this chapter: “Providing Analysts/Designer with additional, intuitive means supporting the construction of OCL queries”. That requirement can be detailed into following remarks:

- As concluded in the D2.1 requirement analysis, the users of profile Analyst/Designer are expected to have a good knowledge of UML diagrams, but are not necessarily focused with programming languages.
- Although OCL is an OMG standard and seems to be an optimum choice for a powerful expression/query language for UML behaviour, the knowledge of this language is not very widespread.
- Moreover, the OCL expression are potentially the most complex part of VIDE language.
- Visualisation offers higher expressiveness compared to textual syntax. Particularly, common UML diagrams could be adopted to represent expressions in a more intuitive and a bit more abstract way compared to textual OCL code.
- Above considerations suggest the development of additional mean of specifying VIDE expressions that is based on UML instance diagrams and extends in a uniform way the constructs already included in VIDE visual notation.

10.4 VEB environment overview

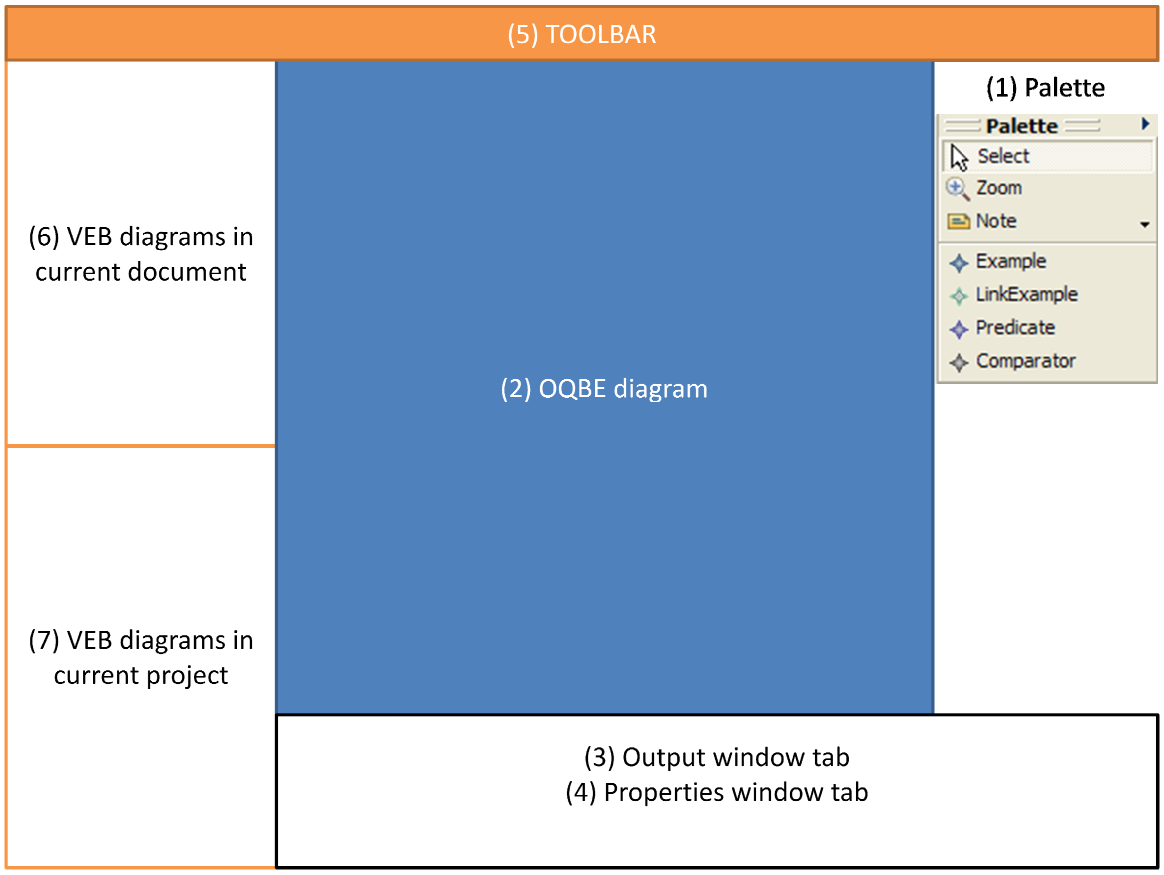


Figure 82 Overview of VEB elements

10.4.1 Palette

This box contains graphical elements representing OQBE building blocks. Each element can be dragged and dropped on **(2) OQBE diagram** area. Following elements are available in this stencil:

- Example symbol
- Link used to connect two examples
- Predicate
- Comparator
- OutputFlag marker
- SortedBy marker

Additionally user can switch to zooming tool or add a UML note using this toolbox.

Please refer to section (@)2 for complete description of the OQBE syntax.

10.4.2 OQBE diagram

Elements dragged from **(1) Palette** can be freely repositioned on the screen area. Layout of all elements placed on this area is stored and restored every time user opens saved VEB file. Behaviour of this editor area is similar to UML editors found in mainstream modelling tools.

10.4.3 Output tab

This window is used to communicate all events and errors encountered during editing of an OQBE diagram.

10.4.4 Properties tab

Using this tab, users can specify properties of elements that are selected on the diagram. After user's selection of any graphical element on the **(2) OQBE diagram** area, properties window shows available parameters of the selected element and their actual data. By entering new values for these parameters, user can alter the shape of resulting query.

10.4.5 Toolbar

Standard Eclipse toolbar extended with:

- Add new visual expression button

While editing expression in VEB there will be additional buttons available:

- Generate OCL expression
- Save diagram
- Delete this expression
- Unlink
- Delete example
- Delete comparator
- Mark as an output
- Sort by

10.4.6 VEB diagrams in current document

User can browse all diagrams that are currently defined for this document and open them by double clicking VEB diagram's name on the list.

10.4.7 VEB diagrams in current project

User can browse all diagrams that are currently defined for currently opened project and open them by double clicking VEB diagram's name on the list. This list is missing elements displayed in *VEB diagrams in current document* view. Diagrams from current project can not be dropped directly on the current document. Instead, they must be copied first to the current document.

10.5 Functional requirements

Use case name and ID:	**10.5.1 Adding new visual expression**
Description:	User can add new visual expression anywhere inside method's implementation.
Precondition:	None
Scenario:	1. User right-clicks code line in VIDE text editor and chooses *"Insert new visual expression"* option or user selects *"Insert new visual expression"* button from Eclipse's toolbar (5). 2. Editor opens new VEB window and blank document is presented to the user. Eclipse panels are updated to present stencils with OQBE elements and properties window.
Extensions:	None
Related GUI elements:	Please see section (@)1.1 for GUI definition related to this use-case.

Use case name and ID:	**10.5.2 Saving VEB diagram**
Description:	User can save newly created file to store his/her work and give a meaningful name for that visual expression
Precondition:	None
Scenario:	1. User clicks *"Save diagram"* button placed on standard Eclipse toolbar

	2. Editor displays automatically generated name and asks user if he/she wants to give his/her own meaningful name for the diagram 3. User changes the name or selects auto-generated one 4. Editor saves diagram file and displays its name on the (6) *VEB diagrams in current document* window
Extensions:	4. Editor checks that name already exists or is missing 5. User is redirected to Step 3.
Related GUI elements:	None

Use case name and ID:	**10.5.3 Generating code from visual expression**
Description:	User can generate OCL representation of the VEB expression after completing definition of examples' attributes values and linking examples with each other.
Precondition:	At least one example is added to the diagram and the diagram is saved. If not, user is asked to do so and UC (@) 1.2.2 is triggered.
Scenario:	1. User clicks *"Generate OCL expression"* button placed on standard Eclipse toolbar 2. Editor generates textual representation of VEB expression and changes focus to the editor window with newly generated code, scrolling cursor to the position where this code is placed. Generated code is read-only and editor disables any editing actions on the code.
Extensions:	None
Related GUI elements:	Please see section (@)1.1 for GUI definition related to this use-case.

Use case name and ID:	**10.5.4 Editing and viewing existing visual expression**
Description:	User can edit any visual expression that is already defined anywhere inside current document.
Precondition:	At least one VEB diagram must be attached to the document
Scenario:	1. User double-clicks diagram's name on the (6) *VEB diagrams in the current document* window or clicks a link placed inside textual code generated from the diagram 2. Editor opens VEB editor window and restores all saved elements along with their previous positioning
Extensions:	None

Related GUI elements:	`Click here to open diagram`
	`//Generated code (path/file.veb_diagram)`
	`salesForecast.expectedRevenueAmount :=`
	`      items -> collect (quantity * product.price) -> sum();`
	`//End (path/file.veb_diagram)`

Use case name and ID:	**10.5.5 Viewing external visual expression**
Description:	User can open in read-only mode any visual expression that is defined in any document from the current project.
Precondition:	At least one diagram must be present in the (6) *VEB diagrams in the current*

	document window.
Scenario:	1. User double-clicks diagram’s name on the (7) *VEB diagrams in the current project* window 2. Editor opens VEB editor window and restores all saved elements along with their previous positioning. All editing capabilities are disabled and diagram is opened only for viewing.
Extensions:	None
Related GUI elements:	None

Use case name and ID:	**10.5.6 Deleting visual expression**
Description:	User can delete any visual expression defined in the current document
Precondition:	At least one VEB diagram must be attached to the document
Scenario:	1. User right clicks diagram’s name on the (6) *VEB diagrams in the current document* window and selects *Delete this expression* from the context menu or clicks *Delete this expression* button from Eclipse’s toolbar 2. Editor asks user if he/she wants to keep generated code attached to this diagram in the textual document or to remove it 3. User selects to delete diagram along with generated code 4. Editor deletes visual diagram and generated code
Extensions:	3. User selects to delete diagram but keep generated code 4. Editor deletes visual diagram leaving generated textual code as if it were hand typed code
Related GUI elements:	None

Use case name and ID:	**10.5.7 Inserting visual expression into textual code**
Description:	User can: • use diagrams that are currently detached and not used in the document • link already used diagrams to many places in the same document
Precondition:	At least one diagram must be present in the (6) *VEB diagrams in the current document* window.
Scenario:	1. User drag and drops diagram name from (6) *VEB diagrams in the current document* window to a new line in textual code 2. Editor creates a link to this diagram and commented space for generated code but it doesn’t generate any code at this moment (because VEB diagram may be incomplete)
Extensions:	None

Related GUI elements:	After linking diagram the code should look like this:
	`Click here to open diagram`
	`//Generated code (path/file.veb_diagram)`
	`//End (path/file.veb_diagram)`

Use case name and ID:	**10.5.8 Importing visual expression defined in other file**
Description:	User can import a diagram that is already defined in the other file inside currently

	opened project. Imported projects are copied and no back reference is maintained.
Precondition:	At least one diagram must be present in the (7) *VEB diagrams in the current project* window.
Scenario:	1. User drag and drops diagram from (7) *VEB diagrams in the current project* window to (6) *VEB diagrams in the current document* window 2. Editor copies the diagram and asks user for a name for it 3. User enters the name 4. Editor saves diagram file and displays its name on the (6) *VEB diagrams in current document* window
Extensions:	4. Editor checks that name already exists or is missing 5. User is redirected to Step 3.
Related GUI elements:	None

Use case name and ID:	**10.5.9 Detaching generated code from its diagram**
Description:	User can decide to detach code generated by the VEB and edit it manually without using the builder.
Precondition:	At least one diagram must be linked with the textual document and there must be textual code generated from this diagram.
Scenario:	1. User right clicks on the link inside the code and chooses *Detach code from its diagram* option 2. Editor removes link from the textual code along with comments symbolising range of generated code. Detached code can be edited and it is treated as any textual code entered manually.
Extensions:	None
Related GUI elements:	None

Use case name and ID:	**10.5.10 Adding new example on a diagram**
Description:	User can add new example on a diagram to specify filtering conditions.
Precondition:	New or existing visual diagram must be opened in editing mode. At least one UML class diagram must be attached to the project.
Scenario:	1. User drag and drops new example icon on a diagram from (1) *Palette* window 2. Editor draws empty example shape on a diagram area and asks user to choose type of this example 3. User selects class name from the list. This list is populated from UML diagrams attached to currently opened project
Extensions:	None
Related GUI elements:	None

Use case name and ID:	**10.5.11 Setting values of attributes of an example**
Description:	User is able to set values of any example already added to a diagram.
Precondition:	At least one example is present on a diagram.

Scenario:	1. User selects an example from a diagram. 2. Editor focuses on *(4) Properties window tab* and displays all primitive type attributes defined in example's class 3. User enters values for attributes he/she wants to specify and selects desired operators. 4. Editor saves entered data and present it on the diagram in corresponding section of example's shape
Extensions:	3. Editor checks that data entered does not match type of an attribute. 4. User is redirected to Step 3
Related GUI elements:	None

Use case name and ID:	**10.5.12 Deleting example**
Description:	User can remove any example already added to the diagram.
Precondition:	At least one example must be added to the diagram.
Scenario:	1. User clicks on an example to select it and clicks *Delete example* button on the (5) toolbar. Alternatively user can right-click example and select *Delete example* option from the context menu. 2. Editor removes example from the diagram and from the model with all links connected to it.
Extensions:	Text here
Related GUI elements:	Text here

Use case name and ID:	**10.5.13 Linking two examples**
Description:	User can associate two examples to notify that expression should look for instances that are connected with each other
Precondition:	At least two examples must be present on a diagram
Scenario:	1. User clicks on one example and drags from the new link shortcut icon to another example 2. Editor draws a solid line with an arrow between these examples to symbolise that they have been linked
Extensions:	None
Related GUI elements:	None

Use case name and ID:	**10.5.14 Unlinking two examples**
Description:	User can break association between already connected examples
Precondition:	At least one link must be present on a diagram
Scenario:	1. User clicks on a solid line connecting two examples to select it and clicks *Unlink* button on the (5) toolbar. Alternatively user can right-click a solid line and select *Unlink* option from the context menu. 2. Editor removes solid line connecting two examples.
Extensions:	None
Related GUI	None

elements:	

Use case name and ID:	**10.5.15 Setting query output**
Description:	User can set an example and/or attribute(s) to be an output of a query generated by VEB
Precondition:	At least one example must be present on a diagram
Scenario:	1. User selects example by clicking it on a diagram. 2. User clicks *Mark as output* button on the (5) toolbar. Alternatively user can right-click an example and select *Mark as output* option from the context menu. 3. Editor attaches <<output>> stereotype to the example. 4. User optionally gives a name to newly added output. 5. Editor saves added value.
Extensions:	1. User selects attribute by clicking it on a diagram. 2. Proceed with point 2.
Related GUI elements:	None

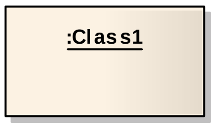
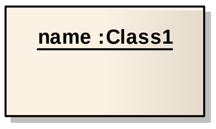
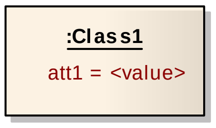
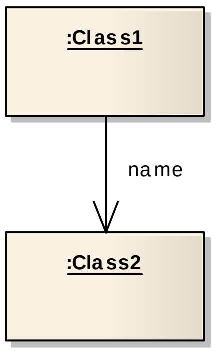
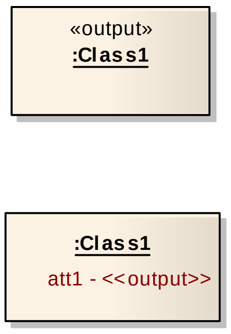
Use case name and ID:	**10.5.16 Setting sort attribute and sort direction**
Description:	User can define what sorting attribute(s) should be used to generate proper order of a result set.
Precondition:	At least one attribute should be available on a diagram
Scenario:	1. User selects attribute on a diagram and clicks *Sort by* button on the (5) toolbar. 2. Editor opens (4) properties window. 3. User specifies sorting direction [asc | desc] and sorting order (natural number). Attributes with lower order value will be used as a primary sorting condition.
Extensions:	None
Related GUI elements:	None

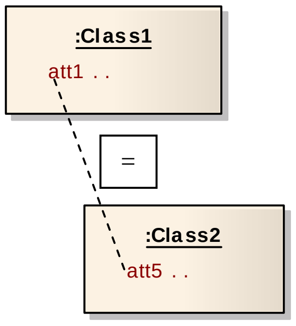
Use case name and ID:	**10.5.17 Adding comparator of attributes from different examples**
Description:	User can put constraints on attributes from different examples to specify their mutual relationship that must be met.
Precondition:	At least two attributes are defined in separate examples on a diagram.
Scenario:	1. User clicks attribute and drags dashed line to an attribute placed inside another example. 2. Editor opens (4) properties window. 3. User specifies comparison operator. Operator is selected from a drop down list populated with operators defined in OCL for attribute type.
Extensions:	None
Related GUI elements:	None

Use case name and ID:	**10.5.18 Deleting comparator of attributes from different examples**
Description:	User can remove comparator from a diagram.
Precondition:	At least one comparator must be on a diagram.
Scenario:	1. User clicks dashed line symbolizing comparator and clicks *Delete comparator* button on (5) toolbar. 2. Editor removes comparator from a diagram.
Extensions:	None
Related GUI elements:	None

10.6 OQBE syntax definition

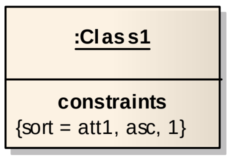
OQBE's syntax is defined using UML instance diagram as a foundation for its notation. Complete syntax of OQBE is presented in following table.

OQBE element	Description and attributes
 **Figure 83 Unnamed example**	Unnamed example is used to select all instances from data store that are the same class type as the example. It can be alternatively named as an extent using UML terms.
 **Figure 84 Named example**	Named example is used to select all instances from the source (defined by *name* parameter) that are the same class type as the example. ***Name*** can be matched to: - local variable; - method's parameter; - object's attribute.
 **Figure 85 Filtering values**	Filtering values can be applied both to named and unnamed examples. They specify what conditions must be satisfied to return certain instance as a result of a query. If there are more than one filtering values specified, they will be treated as separate filters connected with *and* operator. Operators and types of filtering values are defined by OCL type system. Only attributes present on corresponding UML diagram can be used in the context of an example.
 **Figure 86 Link**	OQBE supports only directed links. Links has name (corresponding to the name of association between Class1 and Class2 on UML diagram) and are used to define filtering conditions on mutually connected instances. Instances that satisfy link condition will be selected as a result of a query, i.e. there must by association between instances that has the same name and direction.
 **Figure 87 Query output**	Each query defined on VEB diagram will return at least one result defined by <<output>> stereotype. This stereotype can be attached to example as a whole or to its attribute. If it is associated with an example, the result will be a collection of example's class instances. If it is associated with attribute, the result will be a collection of attribute's types. There can be more than one output defined on a single OQBE expression. In this case the result will be a Cartesian product of all outputs.



Attribute comparator is used to compare values of two attributes. Operators are defined by OCL type system. Only instances satisfying comparator condition will be used to form the query result.

Figure 88 Attribute comparator



Query output can be sorted using multiple sorting criteria. Each criteria is denoted by: attribute name, sorting direction (asc, desc) and sorting priority.

Figure 89 Sorting

10.7 OQBE mappings to OCL

OQBE can be mapped to a number of query/programming languages. In this project OCL is used and thus OQBE will be mapped to ad executed as OCL. Each OQBE query can be mapped to OCL in more than one way. Furthermore, not all OCL constructs can be mapped to OQBE, since it covers only a subset of OCL querying capabilities. Therefore, the reverse mapping (from OCL to OQBE) cannot be defined.

The mapping is constructed in several steps.

1. A minimal set *S* of examples such that all examples are reachable from the examples of this set.
2. For each example *E* from the set *S* a query selecting database objects described by this example is constructed. It has a form *generator*->select(*condition*), where the generator is either the extent of the class of the example *E* (if *E* is not named) or the variable with the same name as the name of example *E*. The condition is the filtering predicate for the example *E*.
3. A Tuple of the queries from the step 2 is constructed:

   Tuple { E_1 = generator_1->select(condition_1),
   …,
   E_n = generator_n->select(condition_n) }

4. Then this Tuple is enriched with fields corresponding to subsequent examples which can be reached from currently available (already reached examples). Assume that query_m the tuple constructed so far consisting of *m* fields representing *m* examples. We are going to add the example E_{m+1} which is reachable from example E_i by link name *linkName*. Let us denote by condition_{m+1} the filtering condition for example E_{m+1}. The query constructed in this step is the following:

   query_m->collect(Tuple{E_1=E_1,…,E_m=E_m, E_{m+1}=E_i.*linkName*->select(condition_{m+1})})

5. Repeat the step 4 until all examples are traversed by the constructed query.
6. The sorting criteria are collected and added at the end of the query obtained after finishing the steps 4-5. The criteria are added from left to right starting from the least significant criterion. The trail added to the query is the following (the criterion_k is most significant, while the criterion_1 is least significant):

   ->sortedBy(criterion_1)->…->sortedBy(criterion_k)

7. Finally the collect operator is added to limit the output only to these items which are marked as query outputs. The result will be a tuple if the number of outputs is greater than 1. Let us assume that output$_1$,…, output$_k$ specify the outputs of the constructed query; each of them is either an example of one of its attributes). The trail added in this step is one of the two:

   ->collect(output$_1$)

      if the number of outputs = 1

   ->collect(Tuple { name1=output1,…, namek=outputk })

      if the number of outputs > 1 where name1,..,namek are the names of items of output tuple. name_i is either the name given the <<output>> stereotype or the name of the output element (attribute or example), if the <<output>> stereotype is not named.

The mapping procedure is quite complex which shows in steps from 3 to 5 because OCL lacks a join operator which has to be simulated somehow.

10.8 Example

Following example presents how user will interact with VEB editor in order to specify sample query. This example is referring to the Opportunity case used as a leading sample scenario in the VIDE project.

Goal: ***Select all UUIDs priority opportunities having the same product in its items.***

OCL Code:
```
Opportunity->collect(o | o.items->collect(i1 | o.items
      ->select(i2 | i1 <> i2 and i1.product = i2.product)
            ->collect(o.UUID)))
```

Sequence of actions with intermediary views of OQBE diagram:

1. User chooses line in textual code and right clicks it and selects *“Insert new visual expression”* option
2. Empty OQBE diagram is opened
3. User drags and drops Example icon from (1) Palette. See Figure 90.

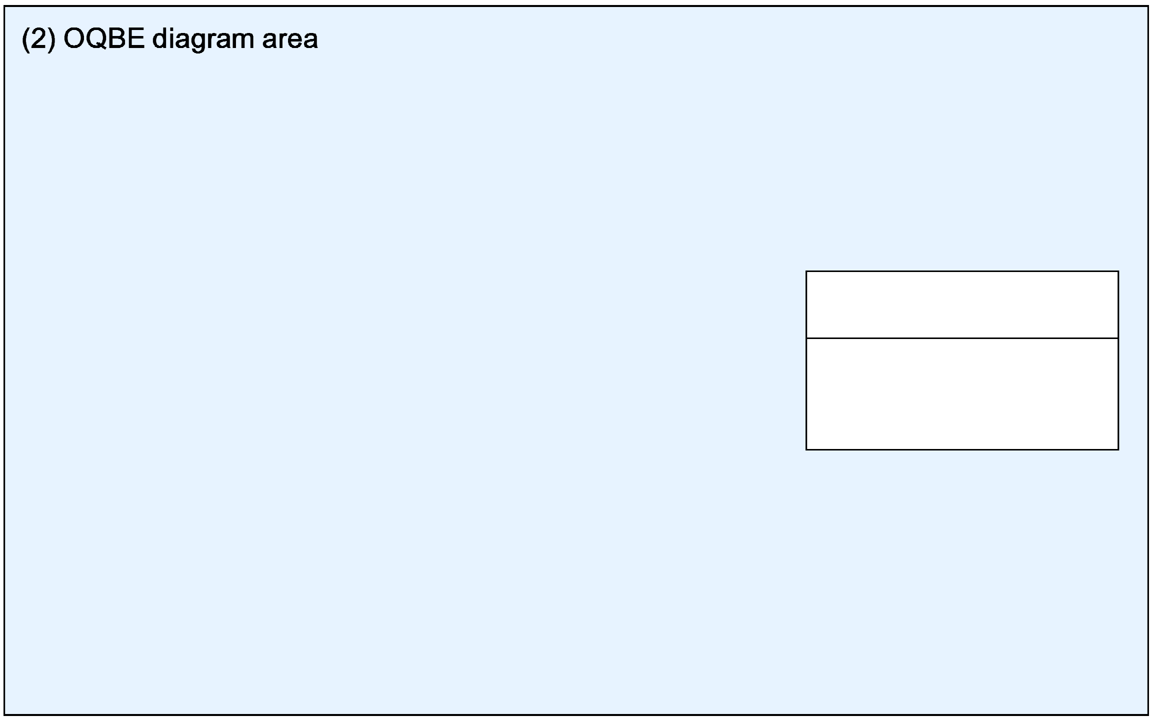


Figure 90: User drags and drops Example icon from (1) Palette

4. User clicks newly added element and then specify its properties in (4) Properties window tab see Figure 91.

- *Opportunity* class is selected as the example's type
- source of the example is set to *extent* – no name is entered
- *priority* attribute is added from the list of Opportunity class attributes and its value is compared using *equality* operator to *"high"*

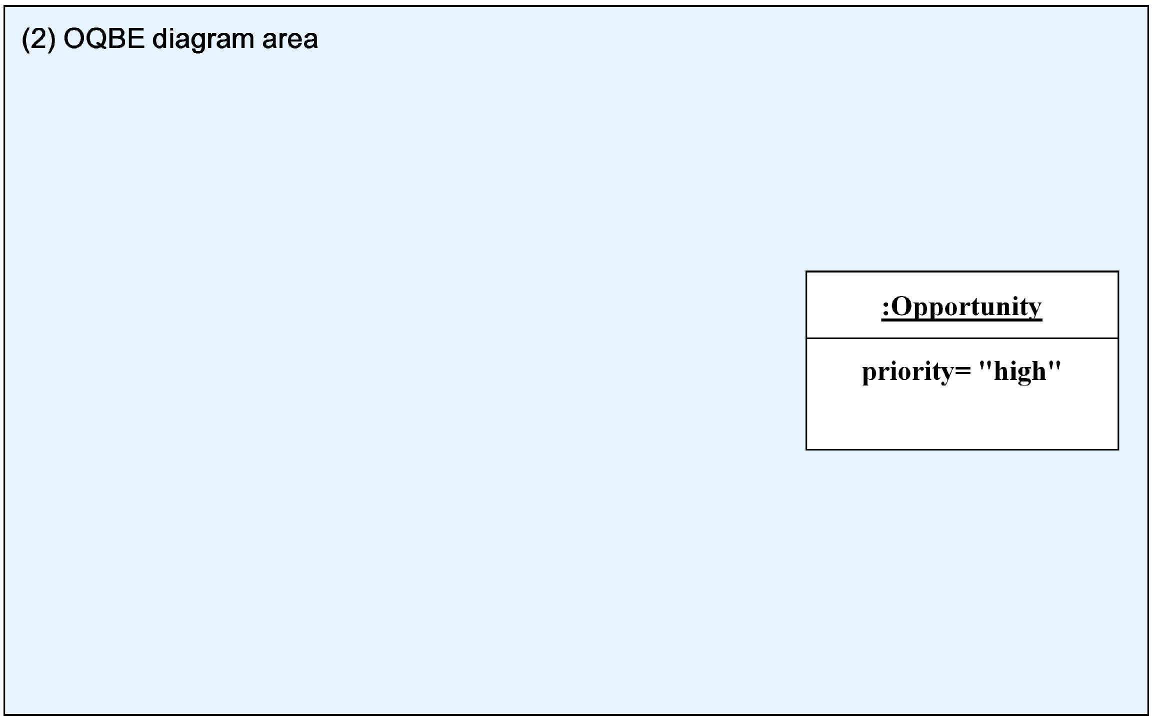


Figure 91: User specifies properties

5. User drags and drops two Example icons from (1) Palette see Figure 92

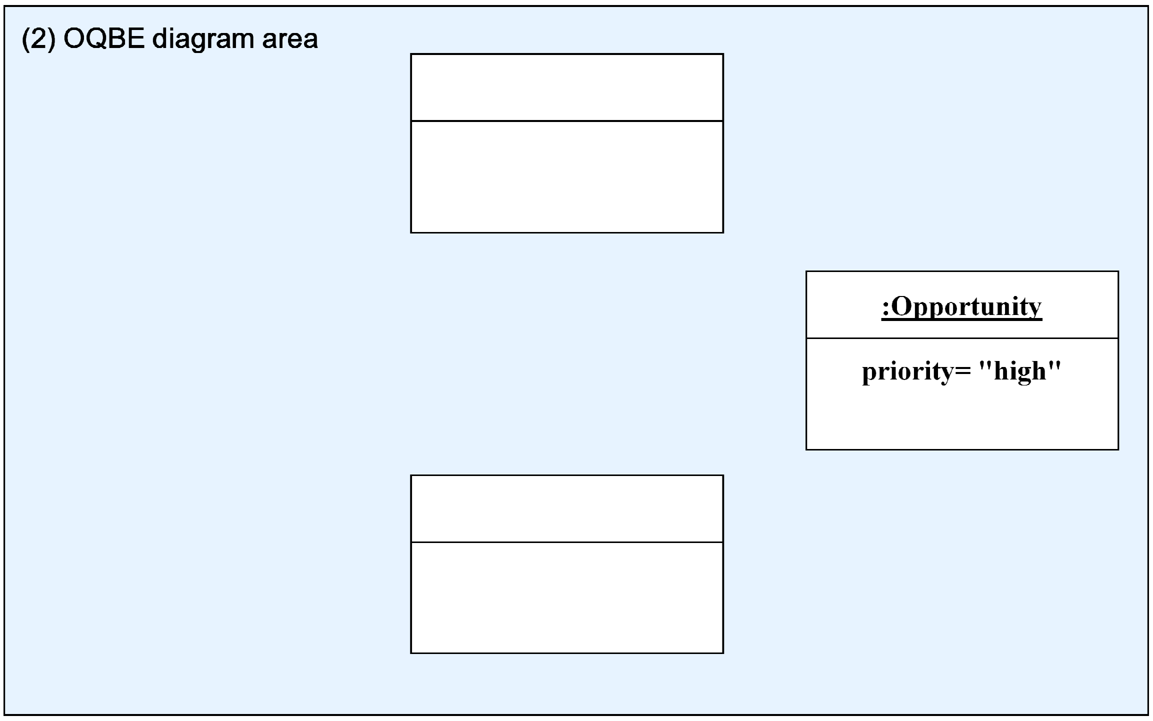


Figure 92: User drags and drops two example icons from (1) Palette

6. User sequentially clicks newly added elements and then specify their properties in (4) Properties window tab see Figure 93
   - *Item* class is selected as the example's type
   - source of the example is set to *extent* – no name is entered

- *quantity* attribute is added from the list of Item class attributes and its value is compared using *grater than* operator to *0*

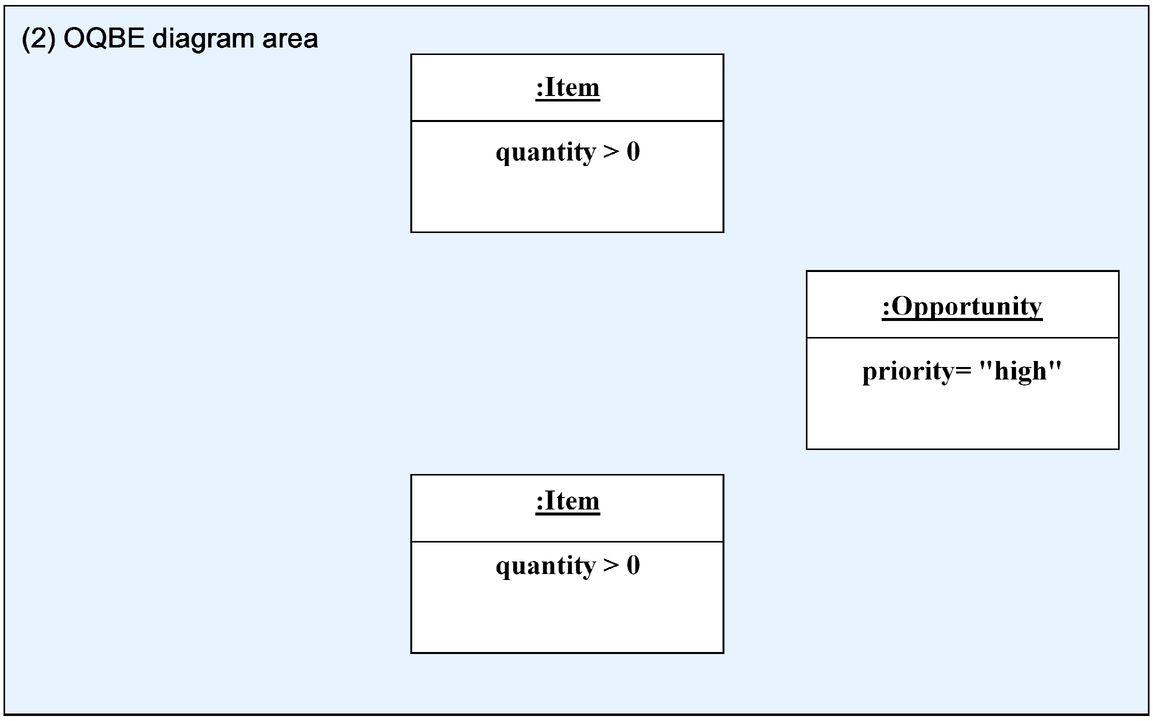


Figure 93: User specifies further properties

7. User links Opportunity example with Item examples by dragging line from Opportunity to Item see Figure 94.

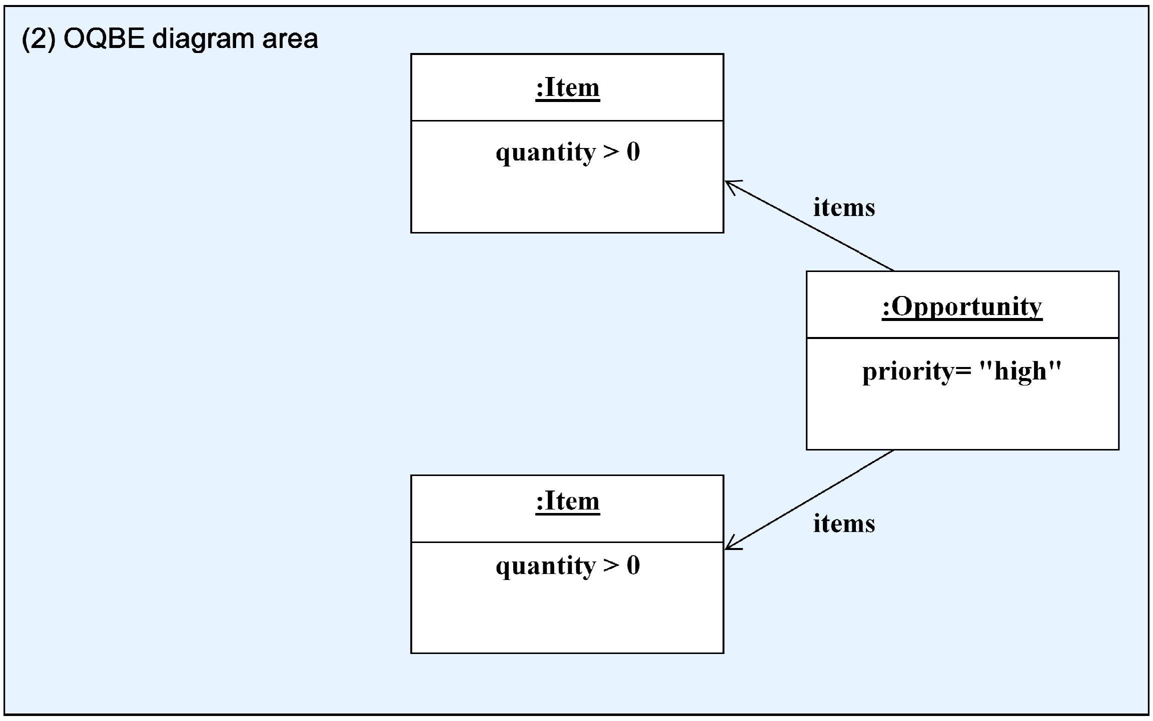


Figure 94: User links Item and Opportunity icons

8. User drags and drops Example icon from (1) Palette see Figure 95

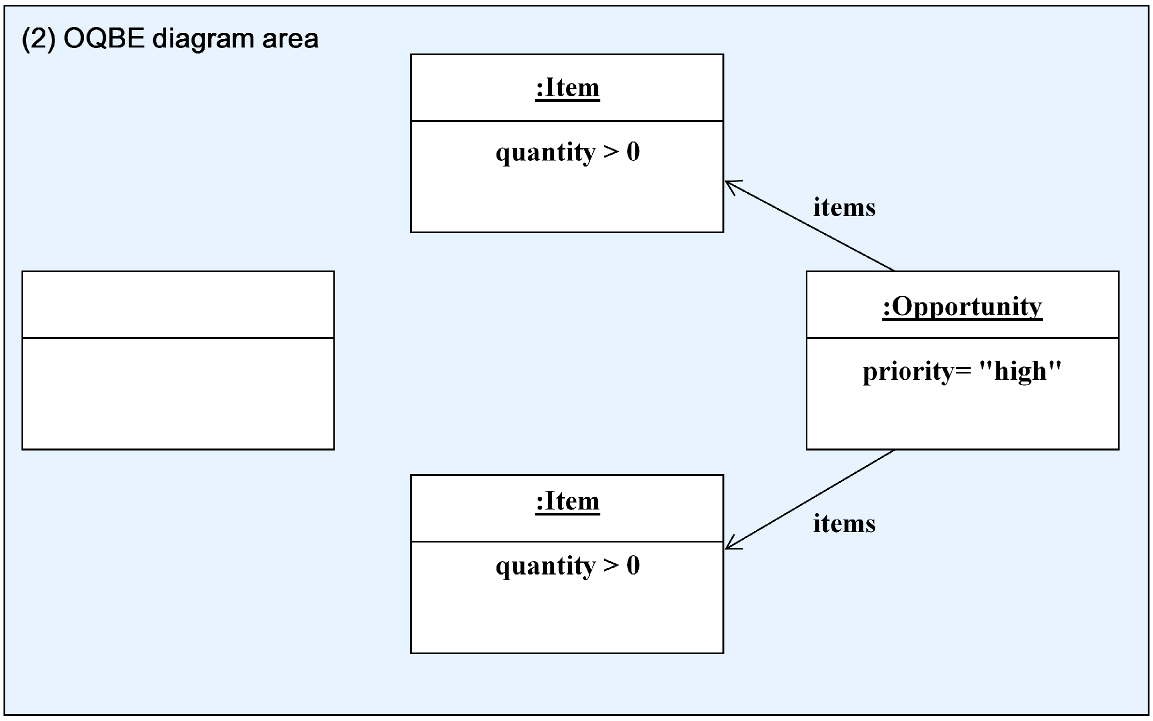


Figure 95: User drag and drops further icon

9. User clicks newly added element and then specify its properties in (4) Properties window tab see Figure 96.
   - *Product* class is selected as the example's type
   - source of the example is set to *extent* – no name is entered

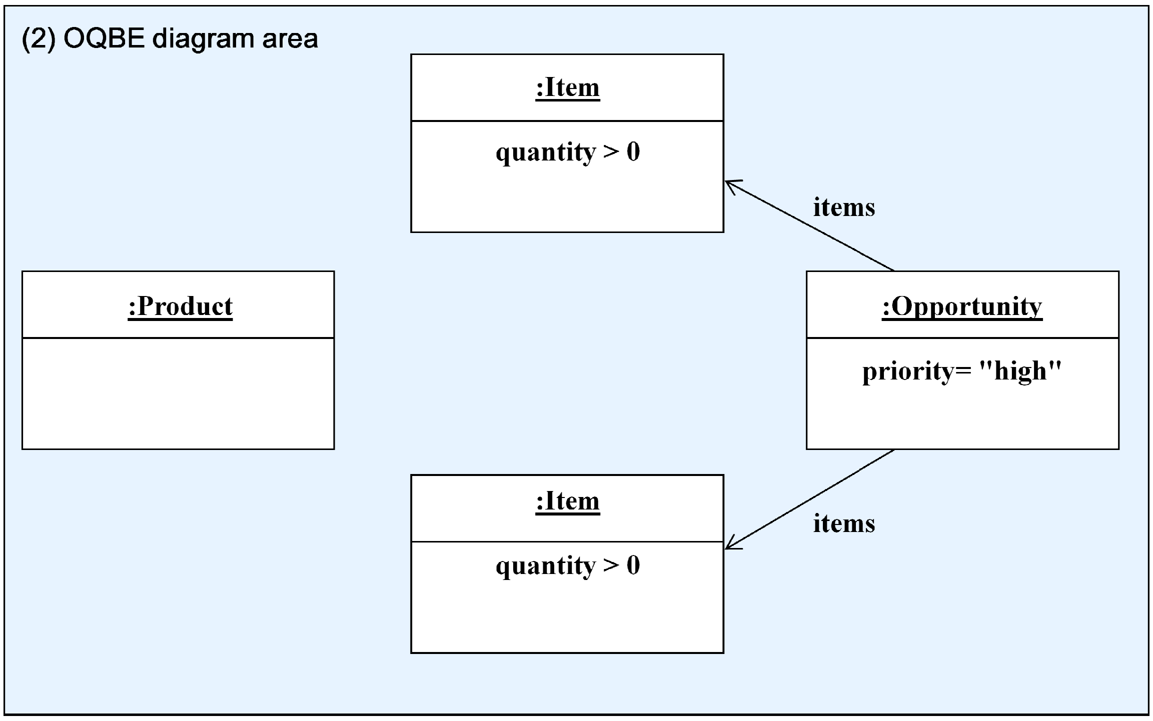


Figure 96: User specifies properties of new icon

10. User links both Items with Product example by dragging line from Items to Product see Figure 97.

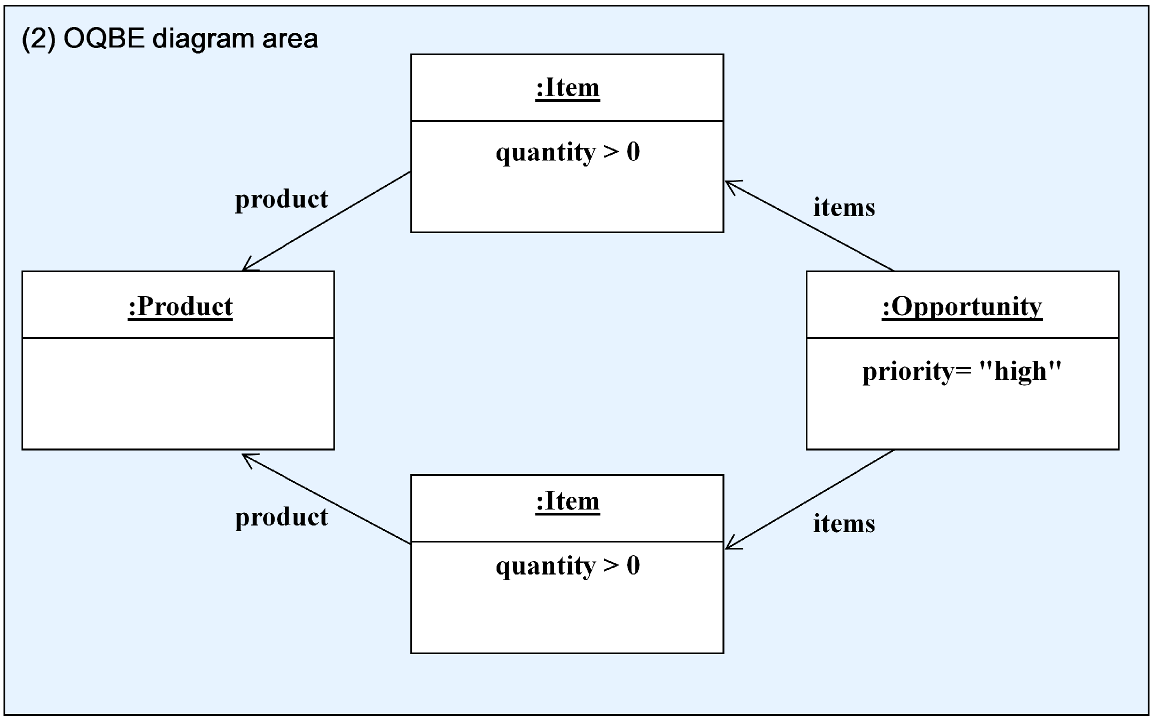


Figure 97: User links Product icon

11. User clicks *"Generate textual expression"* button placed on standard Eclipse toolbar. Diagram is saved and textual representation is opened.

10.9 Future development

Current version of VEB editor could be extended by:

- direct drag and drop support for UML diagrams created in external tools;

10.10 Glossary

Example – symbol representing filtering condition used for browsing data store for objects that have the same attribute values as the example instance

VEB – Visual Expression Builder

OQBE – Object Query By Example

11. Interface to legacy applications

11.1 Introduction

One of the important issues of modelling is taking the value from existing applications. The VIDE toolkit should also be able to document existing applications and possibly, in the context of model execution, merge the existing functionality and data in such a way, that it become a part of the new executable application.
The issue of modelling existing application and integrating them as a part of new one is as complex that can be a subject of large separate project. Thus in this document we take a limited approach narrowing the problem to the aspects that were found directly needed in the course of the development of other work packages.
Moreover, it is necessary to note that the MDA approach with executable modelling as followed by VIDE, establishes certain assumptions on the software development originated with a model. From this particular point of view it seem justified to broaden the definition of legacy software to all the existing application for which the VIDE model does not exists and which therefore require a kind of reverse engineering in order to make them available for integration with VIDE-developed software.

11.1.1 Taking the value from existing applications

The value of existing applications can be perceived from the following point of views:

1. The functionality of an application that can be reused in new application through some interface.
2. The application data model and data stored in it that can be plugged and reused in new solution.

The first case, connected with reusing the existing functionality, requires the mechanism of common API to that functionality, enabling the VIDE to document and use it. Because the number of various programming languages and APIs that the existing solutions use can be very large, the VIDE project has to assume some common, generic interface that can wrap the legacy code and make it available to the VIDE model. A rather straightforward choice for these interfaces is resorting to Web Service technology.
The second case concerning documenting existing data stored in legacy applications is also a challenging problem. The data can be stored in different database management system with different data models and APIs. Although the most popular way of storing persistent data in today’s application are relational database systems, that have a common model and query language, the difference between individual database management systems can be significant. The diversity can be manifested in extensions to or departures from standard SQL syntax or the availability and way of access to data catalogue storing the meta data. Another problem that needs to be handled is connected with a so-called “impedance mismatch” between relational and object-oriented data models. The model presented by a VIDE is based on the UML object model that is different from the relational one.

11.1.2 Interaction with other tools of VIDE

From the VIDE perspective there is a need to:

1. Document the legacy applications. The existing solution should be documented in VIDE using a UML model.
2. Executing the functionality of existing solutions as a part of the new VIDE executable model (to be able to check the interaction with existing software at the level of an executable model).

This aim is addressed through the respective refinement and extension of VIDE PIM language elements (necessary to uniformly represent the existing software elements in the model, but also to handle the additional model information specific to them) and through designing appropriate development environment functionality to support actual importing of those model elements.

11.2 Requirements

11.2.1 Relevant requirements from D1.1

The following table lists only the requirements relevant for the legacy integration functionality described in this chapter. Omitting a D1.1 requirement in this table means it was found not relevant for this part of VIDE tooling.

Requirement Number	**Name**	**Priority**	**Comment**
REQ – NonFunc 4	Clear and unambiguous notation – VIDE should have clear, comprehensible and	Should	The legacy documenting functionality uses stereotypes over the UML constructs, and the actual behaviour of application elements described this way matches the semantics of those base constructs.

	unambiguous semantic description suited to the users of the VIDE tools		
REQ – NonFunc 8	Runnable and testable VIDE prototypes	Should	The functionality will be implemented as a part of VIDE model execution engine. Thus, the issue of interacting with existing software will be testable based on this functionality.
REQ – User 2	Reuse of UML Standard – end users are very sensitive to using standards. A key aspect is that the VIDE language reuses as much as possible the UML standard.	Should	All the concepts representing existing applications are expressed in UML.
REQ – Semantics 1	Semantics of VIDE Inte rnal Communication – a precise description of the semantics is needed sufficient for internal communication purposes within implementation stakeholders in the development of the VIDE tool.	Should	The legacy documenting functionality uses stereotypes over the UML constructs, and the actual behaviour of application elements described this way matches the semantics of those base constructs.
REQ – Tool 14	Model driven approach The VIDE tool strictly follows a model driven approach as stipulated in figure 9 page 120 of the D.1.1 deliverable	Must	The legacy documenting functionality depends on the OMG four level meta-modelling architecture and pays special attention do the separation of platform-independent and platform-specific details into respective model layers.

Table 25: Requirements for legacy applications

11.2.2 Refined requirements

As stated in the introduction, the selection of scope for the functionality described in this chapter directly results from the requirements analysis performed in the course of project – especially in the area of tasks 8.1-8.5 of Workpackage 8, where the VIDE architecture and respective needs for particular partners' tools integration and use are analysed. Those requirements can be summarised in the following list:

- Applications developed using VIDE should be able to communicate with other existing software using the Web Service technology. This involves publishing selected functionality of VIDE-developed application in the form of Web Service operations, as well as consuming such operations provided by other software.
- The functionality being consumed and published this way should be represented in VIDE models in a way uniform with remaining, locally available behaviour.
- Web Service communication (particularly, publishing services) should be supported by VIDE model execution engine, in order to perform the following tasks at the stage of executable platform-independent model:
    - prototyping graphical user interface (e.g. with the TNM:WebFace tool) and running it, so it can communicate with VIDE-specified application logic through Web services,
    - running the business processes and application logic implemented with VIDE to test their behaviour – this requires providing VIDE application functionality to a workflow engine (e.g. Rodan OOWF) in the form of Web Service interfaces.
- VIDE should make it possible to import the metadata of existing relational databases and make them available in the form of UML classes and attributes for querying and processing them with VIDE

language. This way VIDE can be used to define data integration logic in a platform independent way which is the aim of the ONAR tool integration with VIDE tooling.

11.3 Retrieving and documenting schemas of existing data sources

11.3.1 The proposed solution for documenting and interacting with legacy data sources

The documenting logical structure of legacy data in VIDE will be narrowed to relational model. To perform the retrieval of relational database schema into our model, we adopt the functionality of relational schema importer, which is a generic tool being developed as a part of ODRA toolset. It allows us to perform a kind of reverse engineering by representing the imported schema in terms of UML model. The way the schema import is implemented is not relevant for the final user, however, the platform specific details being set here for the sake of this reverse engineering can be also directly useful for interacting with the relational data source from the executable model. The latter step is currently realized by ODRA2RDBMS model execution runtime; however it may potentially be realized also other way, if supported by respective model compiler. For this reason and to allow a uniform conceptual modelling, the mechanism is not opaque. Hence, in the further part of the document, we describe the way RDBMS schema elements are represented in VIDE. Figure 98 depicts the overall architecture of the mentioned ODRA tools.

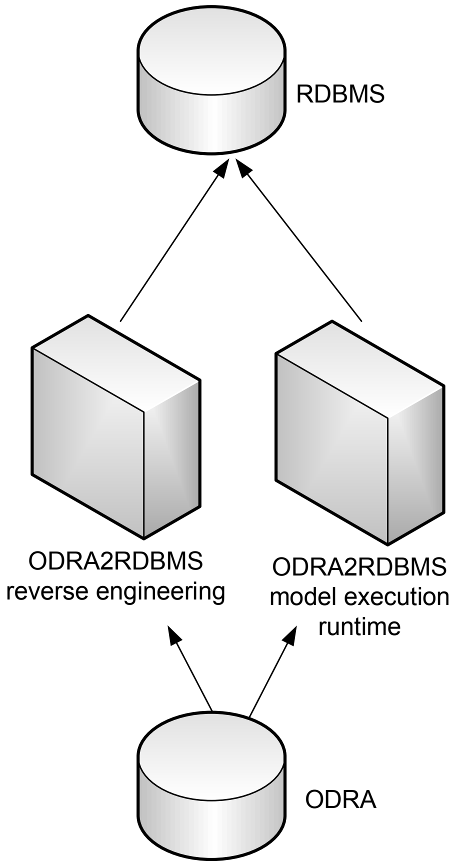

RDBMS
ODRA2RDBMS reverse engineering
ODRA2RDBMS model execution runtime
ODRA

Figure 98: The overall architecture of the ODRA wrapper

To sum up, the purpose of the ODRA tools for relational database access in the VIDE project is:

1. to enable the description of existing relational data sources in the VIDE model. This can be achieved with use of the abovementioned reverse engineering tool. This way relational schemata can be introduced into the VIDE model.
2. to enable transparent querying and updating of the legacy relational data with VIDE language statements. This can be achieved with use of ODRA generic model execution engine.

The sequence of activities needed to import the schema and connection details for existing data sources into the VIDE model is described in the following subsections.

11.3.2 Step 1: Connection configuration

The relational schema of logical data will be read from individual RDBMS thus the basic connection information has to be provided. ODRA reverse engineering tool is based on the JDBC connection that requires the following information.

1. (JDBC) adapter type – the name denotes the target RDBMS (e.g. postgres).
2. (JDBC) connection driver – the name of the JDBC connection driver (e.g. org.postgresql.Driver)
3. connection URL – the URL of the database resource (e.g. jdbc:postgresql://127.0.0.1:5432/sample)
4. username – the name of the database user
5. password

The process of reading the legacy schema concerns the PIM model but the configuration details that needs to be provided are connected with PSM level. It is caused by the process itself.

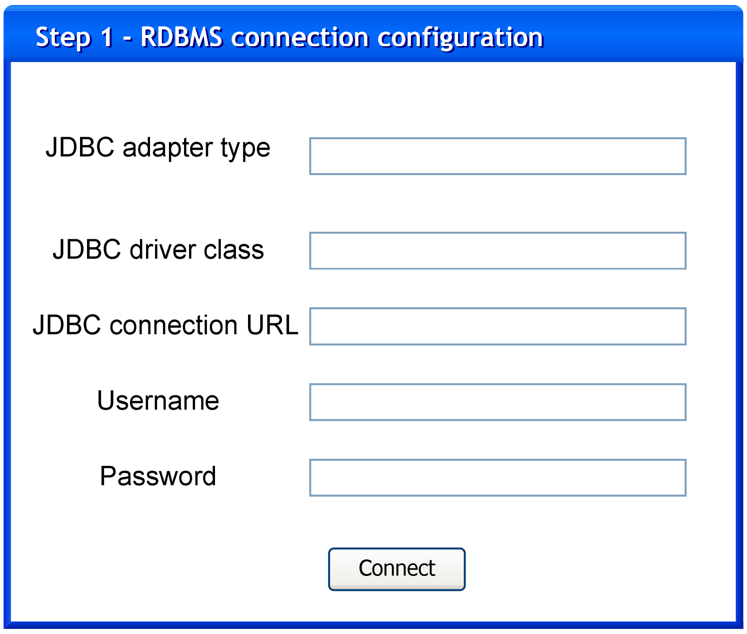


Figure 99: Connection configuration details

11.3.2.1 The details of reading the legacy model

After the user defines the properties for connecting to the legacy database the importer uses the functionality of ODRA2RDBMS reverse engineering tool to create a target system independent description of the data. The input for the ODRA2RDMBS are the connection information, the output is an XML file with a relational schema description. The overall generation process of the schema description is depicted in Figure 100.

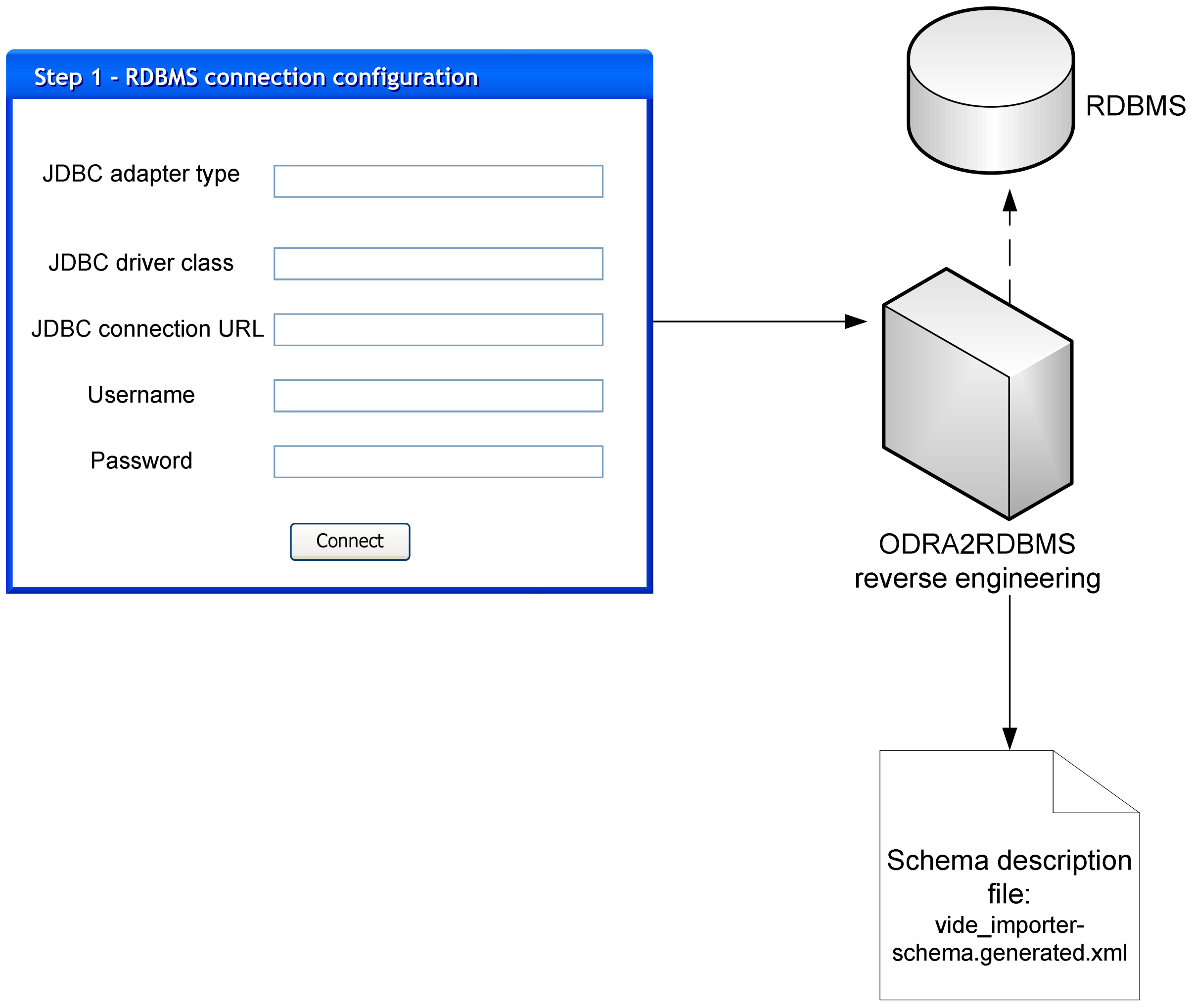


Figure 100: Generation of legacy data model description

According to the connection details, the reverse engineering module connects to a target database and reads the schema[2]. The description of the schema is saved in the form of XML file named: vide_importer-schema.generated.xml. The file is used by the importer in the process of creating VIDE legacy model. Additionally, the XML description together with the connection details are then retained for the purpose of the further use by the ODRA2RDBMS model execution engine in for actual accessing the data source.

11.3.3 Step 2: Generation of the legacy model in VIDE

This step supplements the previous one with the name for the module that is to be created.

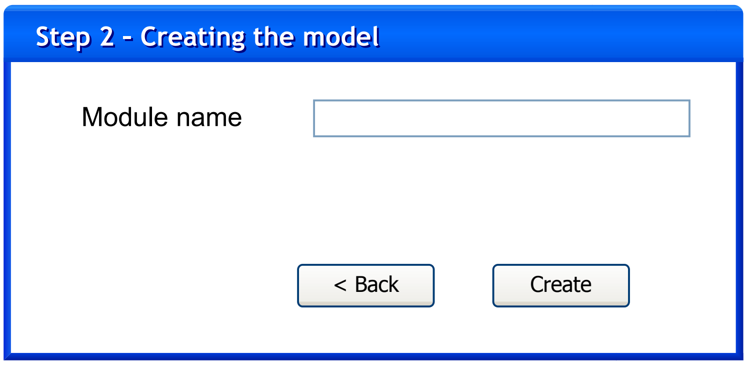


Figure 101: Setting the module name

[2] The method is thus dependent on the availability of the schema exposed by the RDBMS.

Finally, the importer creates the VIDE model representing the legacy relational data. The model details are described in the following section.

11.3.4 Presentation and use of the legacy schema in VIDE model

In VIDE model the legacy relational schema introduced as described above will be represented with use of the UML package. The name of the package is the name of the module given in the step 2 of the importing process. Figure 102 shows sample package storing generated model named "legacy_data".

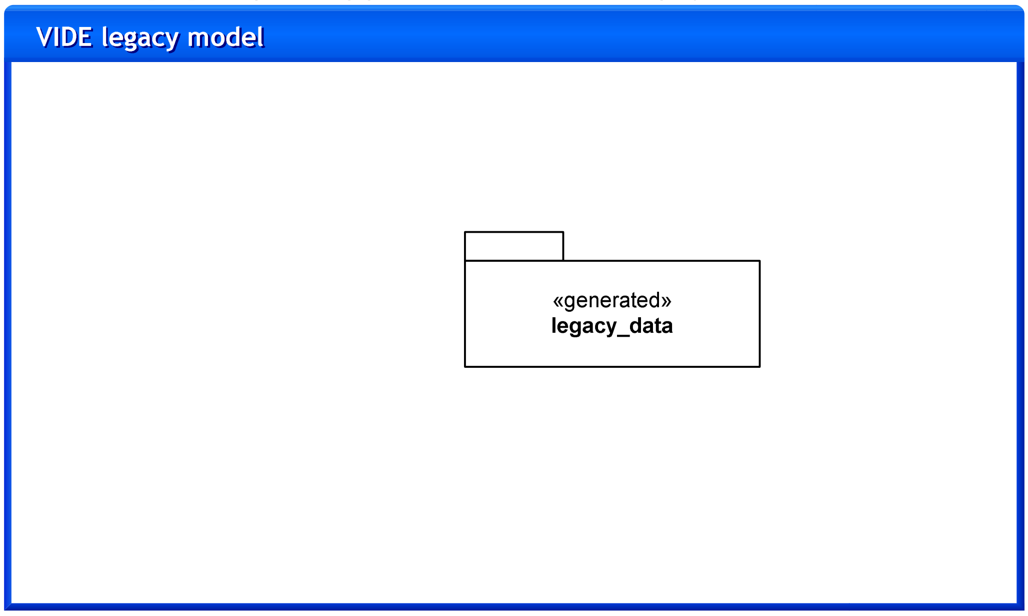


Figure 102 Package with «generated» stereotype

Inside a package the relational model is expressed with use of the following assumptions:

1. Definition of the schema:
    a. Each relational table/view in the source legacy model is represented with use of the UML class. The name of the class is the name of the source table/view with suffix 'Def'. Additionally, the class is given the stereotype «view»[3].
    b. Each table/view attribute is represented as the attribute of a corresponding class.
    c. The class attribute that corresponds to a primary key attribute is additionally marked with «primarykey» stereotype.
    d. The nullability of an attribute is represented by the [0..1] cardinality of the corresponding class attribute.
    e. Further integrity constrains (e.g. type of the primary key, foreign keys, etc.) are currently not represented, however creating OCL constraints to describe them in VIDE model is considered.
    f. : The type mapping are presented in the

SQL	*VIDE*
varchar **varchar2** **char** **text** **memo** **Clob**	**String**
integer **int** **int2** **int4** **int8**	**Integer**

[3] We do not make any distinction between tables and views in the legacy relational model. Moreover we prefer to use the view concept that is consistent with the ANSI/SPARC database layer architecture that assumes the external level build up from the (possibly updatable) views.

serial smallint bigint byte Serial	
number float real numeric Decimal	Real
bool boolean Bit	Boolean
date timestamp	Date

Table 26 Type mapping between SQL and UML in VIDE

2. Declaration of instances collections:
    a. The package contains a class, named same as the package, with a stereotype «module». Each table is represented by an attribute of this class with cardinality [0..*].

Figure 103 shows interior of a sample packages with generated model based on sample relational data.

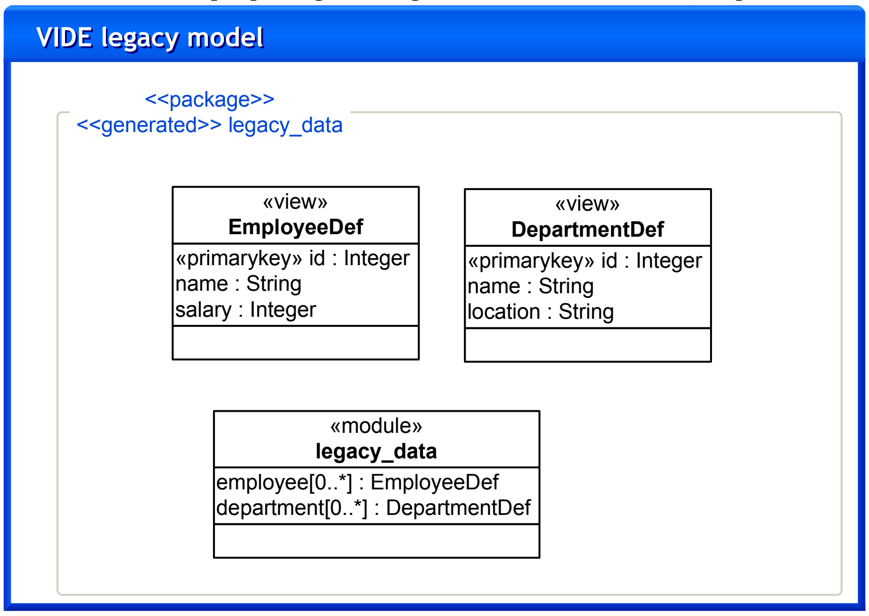


Figure 103: Sample legacy model elements

After importing such a data source, it can be transparently used by in other parts of the VIDE model code. In particular we assume that other parts of the VIDE model can, as a part of its behaviour, execute the generated model parts representing legacy data (effectively performing reads and updates of actual data in those data sources).

11.4 Documenting interacting with existing software available through Web Services

Nowadays, Web Services are important part of computer systems. They make communication across their boundaries easier. The purpose of including Web Services support in VIDE is to allow its users to take advantage of service oriented architectures and to reuse existing services inside Line-of-Business applications. Another reason worth mentioning is TNM:WebFace prototype, being used for GUI prototyping in VIDE, which implements communication with VIDE via Web Services.

11.4.1 Functional aspect

Web Service implements directed communication between client and server. Hence, we distinguish two functional aspects of their usage. Web Services can be consumed in order to include a remotely accessible functionalities into VIDE systems (i.e. currency exchange). Certain model elements can be published as Web Service to allow external connectivity with them. In that case (remote) clients can trigger execution of VIDE code from the outside.

11.4.2 Concepts mapping

In this section we describe how WSDL and VIDE models are mapped to each other. Both for Web Services consuming and publishing scenarios we provide detailed information regarding extended constructs introduced to VIDE to handle Web Service integration properly.

The de-facto standard for Web Services description is WSDL 1.1 (http://www.w3.org/TR/wsdl). WSDL contracts consist of two parts: abstract and concrete. The first one contains interface and types involved in communication details. The second part describes implementation details such as supported messaging and transport protocols.

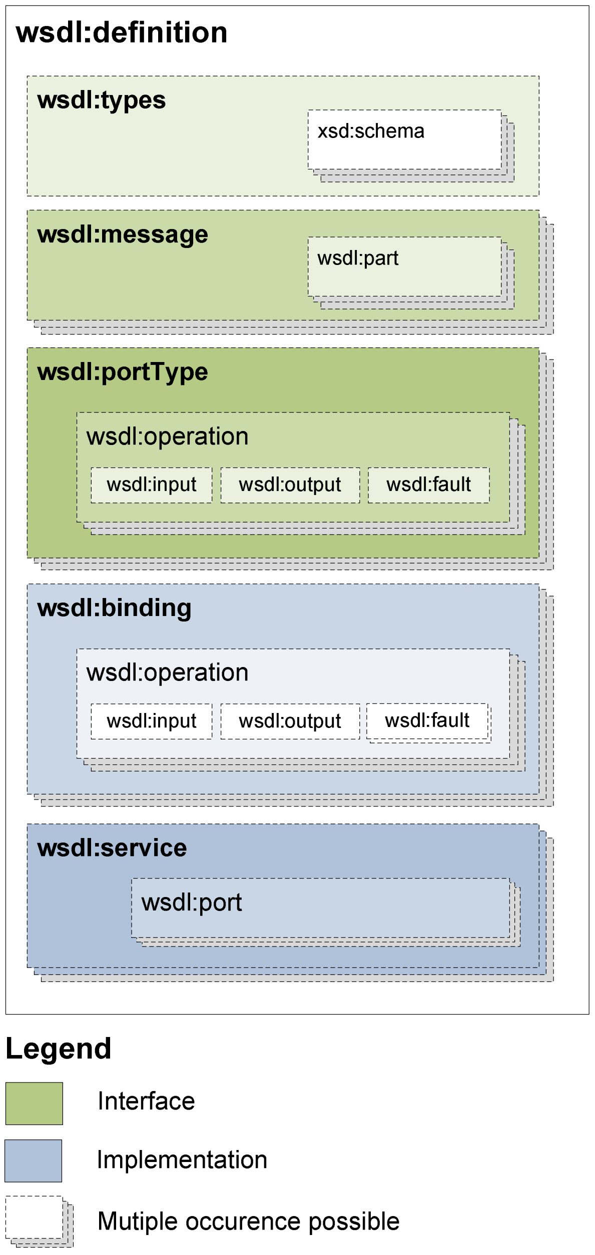


Figure 104 WSDL 1.1 contract diagram

We base VIDE Web Services integration on same (as in WSDL) separation between abstract and concrete part. Mapping between WSDL abstract part and VIDE model is described in the next section.

11.4.2.1 WSDL definition to VIDE mapping

Publishing and consuming Web Services can be considered as symmetrical tasks. In mapping from WSDL to VIDE model and from VIDE to WSDL we try to follow that symmetry. On PIM level only abstract WSDL part is taken into account. This includes `types` and (specified) `portType` section. Types are described as XML Schema and are imported to the system to provide strongly typed access to a Web Service. Interface on the other hand is imported as specialized class to allow transparent remote methods invocations.
Since one WSDL contract maps to many VIDE entities we need to use container for them. For that purpose standard UML package is used. Container content is auto-generated and hence should not be edited by a user. Because it is impossible to forbid editing in a generic way on model level it is only an advice for VIDE users. They should not tweak such imported services manually in any way (but the restriction is not forced).
`PortType` is imported into the container package as a class marked with «`ConsumedService`» stereotype. Only one such element is allowed to be included in the package. The Web Service proxy class has no attributes but it contains an operation for each Web method from target Web Service. Operation parameters are based on input/output web method messages and raisedExceptions association is based on WSDL web method faults. Types described in WSDL are imported as well. The mapping is handled by dedicated component. The types import produces UML types with optional stereotypes assigned to reflect their exact XML Schema meaning. We are not describing details of XSD types import because external tool - XmlModeling hyperModel 3.0 (http://www.xmlmodeling.com/) - will be used for that purpose. The tool bases on complete implementation of UML profile for XML Schema. It is used to annotate UML models with specific preferences for schema customization (reasonable defaults are assumed for un-annotated elements).

Stereotype «ConsumedService»
Designates that class will be a proxy to remote Web Service conforming to certain WSDL contract. Its operations are associated to remote Web method calls of given Web Service. Exact shape of this stereotype will be specified based on the design decisions in model compilers development in VIDE.
Generalizations
Class

During types and portType import to model, the following naming conventions are used:

WSDL	**VIDE**
(encoded) target namespace	containing package names
Port type	(proxy) class name
operations names	operations names

Table 27 Naming conventions used in WSDL to VIDE mapping

11.4.2.2 VIDE to WSDL definition mapping

The second scenario of the Web Services integration in VIDE applies to exposing model elements as Web Services. Any class without attributes declared can be published. Exposing is achieved by adding «`PublishedService`» and «`PublishedOperation`» stereotypes respectively to a class and its operations.

«PublishedService»
Tells system that class should be exposed as Web Service endpoint.
Generalizations
Class

«PublishedOperation»
Marks those operations, which should be available as Web methods of that endpoint.
Generalizations
Operation

Exact shape of these stereotypes will be specified based on the design decisions in model compilers development in VIDE.
Same as in the consuming case name relevance between VIDE and WSDL models is maintained. It is summarized in Table 28:

VIDE	WSDL
Name of class being exposed	portType name
operations names	operations names

Table 28 Naming conventions used in VIDE to WSDL mapping

11.4.2.3 Limitations

WSDL 1.1 supports attaching multiple interfaces (`portType`) realizations (`port`) to one endpoint (service). Such approach introduces interoperability issues and is inconsistent with the next WSDL version design. In order to deal with this problem for Web Service consumption, we require user to specify the interface to be used. For publishing we take minimalistic approach to allow compilers create the most interoperable descriptions as possible.

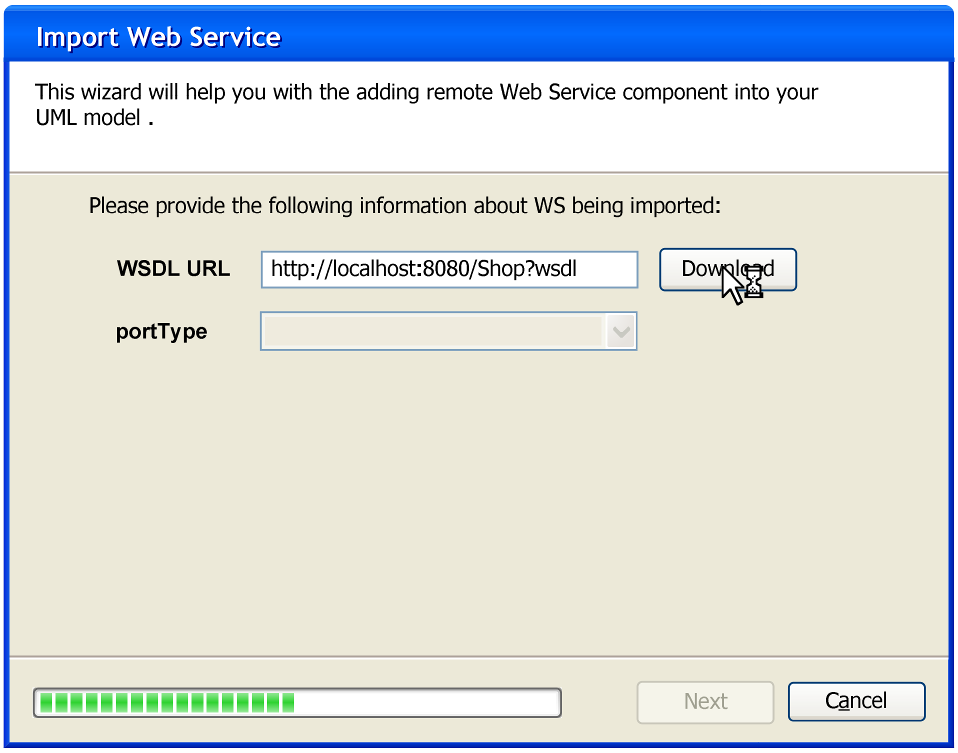
11.4.3 Visual support for handling services

Since visual programming is central concept in VIDE, we dedicate the last section of this document to discuss how Web Services facilities can be embedded inside Visual VIDE Editor. Since Web Services integration in VIDE is separated between abstract and concrete part we decided to implement consumption and publishing processes using wizards. They ease definition process by automatically applying necessary modifications to VIDE model and generation of concrete Web Services options files.
In our opinion the best way to explain visual support is to describe the process with full usage scenarios accompanied with (prototype) diagrams.

11.4.3.1 Scenario 1: Importing remote service into the model

The task is to import Web Service definition into model in order to be able to use it to invoke remote methods transparently from VIDE code.

Step 1: User chooses `Import Web Service` option from main VIDE menu. Wizard appears. On its first screen user provides information about contract of Web Service to be consumed seeFigure 105.



This wizard will help you with the adding remote Web Service component into your UML model .

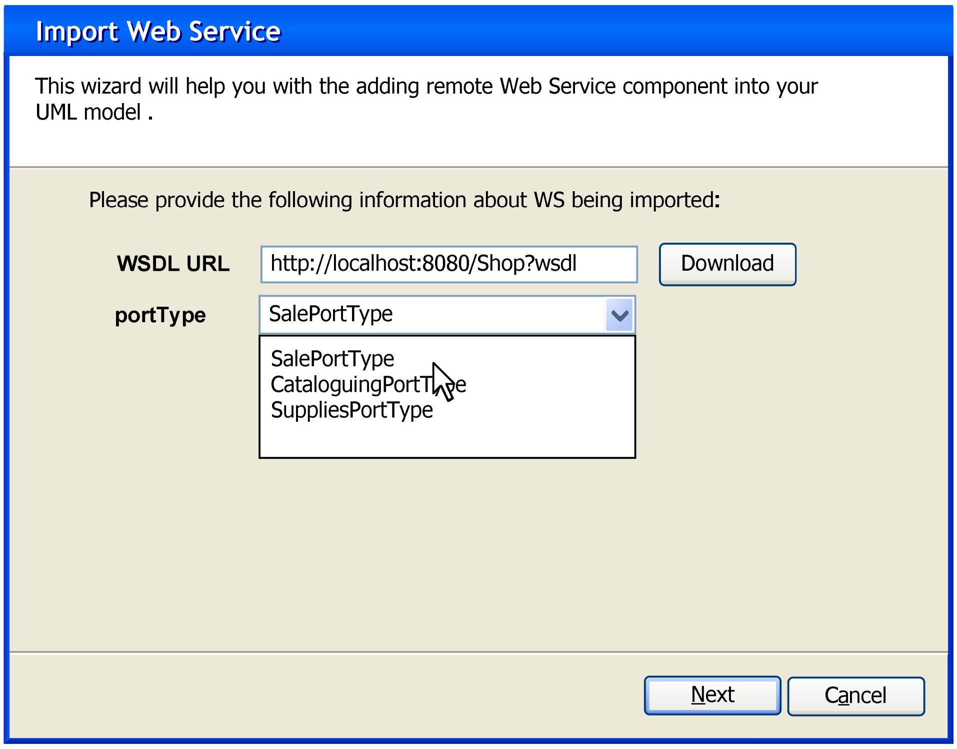
Please provide the following information about WS being imported:

WSDL URL: http://localhost:8080/Shop?wsdl

portType:

Figure 105: Import Web Service Option

Step 2: User clicks `Download` button. After WSDL contract is downloaded successfully - interface drop down list gets enabled. It is populated with all available port types from the Web Service contract. User chooses the interface, which he wants to be imported to the model see Figure 106.



Import Web Service

This wizard will help you with the adding remote Web Service component into your UML model .

Please provide the following information about WS being imported:

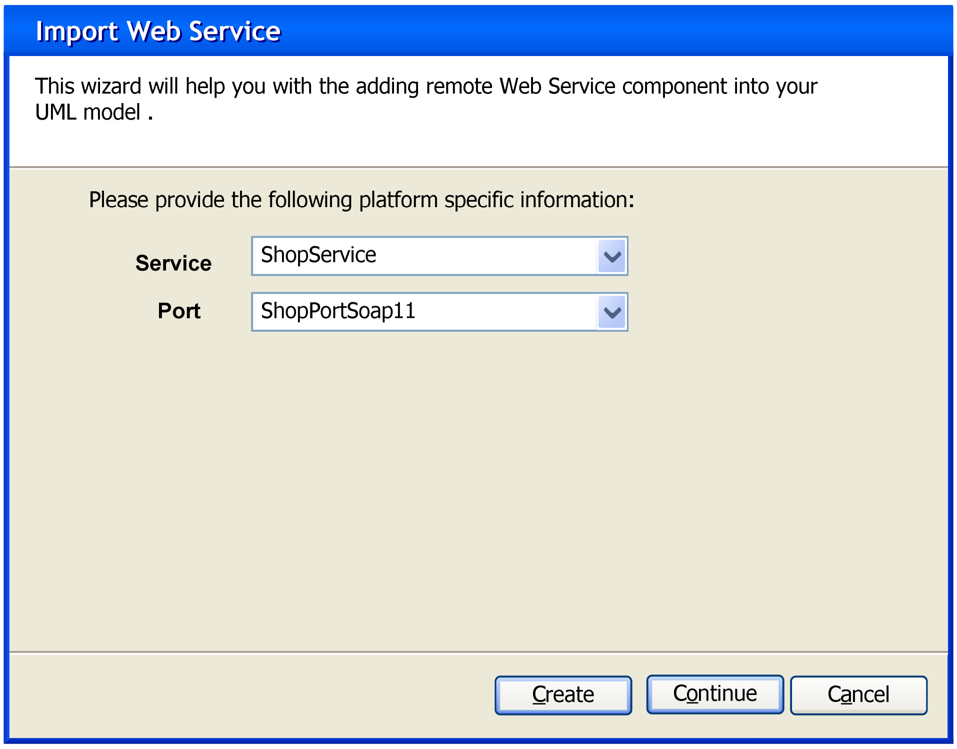
WSDL URL: http://localhost:8080/Shop?wsdl

portType: SalePortType

SalePortType
CataloguingPortType
SuppliesPortType

Figure 106: Import web Service Option showing port types

Step 3: User provides platform specific information about Web Service being imported. This include service and port to use (only those are listed, which contain previously chosen interface implementation) see Figure 107



Import Web Service

This wizard will help you with the adding remote Web Service component into your UML model .

Please provide the following platform specific information:

Service: ShopService

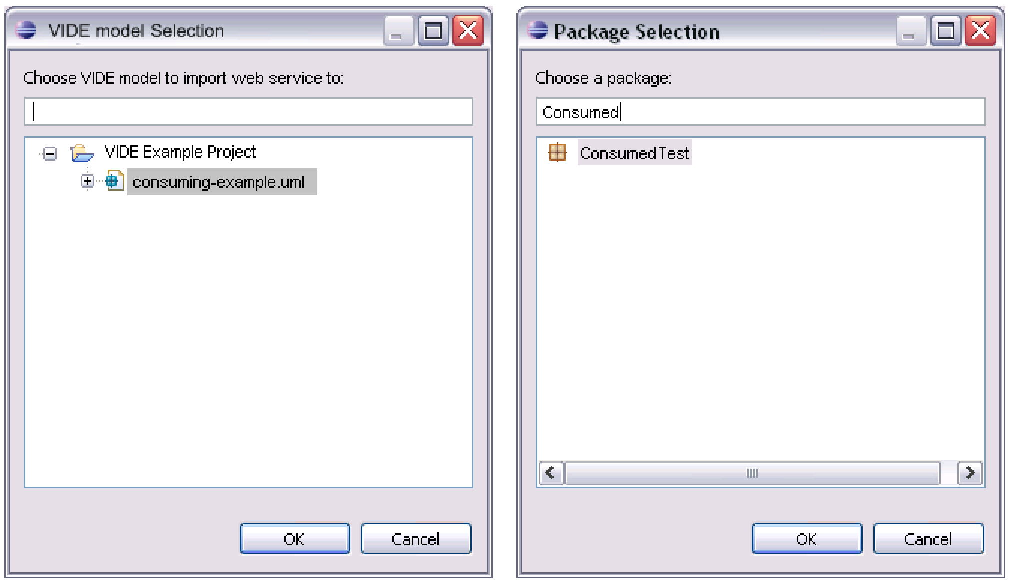
Port: ShopPortSoap11

Figure 107: User provides platform specific information

Step 4: User may choose `Create` or `Continue` option. The first one is active if and only if active package element from VIDE model was active on wizard start. Choosing it will create Web Service proxy inside chosen (active) package. On the other hand clicking Continue button allows to specify target container see Figure 108

Figure 108: Specifying target container

Step 5: All requested options are required. Additionally they may be pre populated with some defaults. For example Model may be filled in if on wizard start active file in IDE explorer pane was VIDE model file. Similarly for package it will be filled in if it was an active model element in the moment of wizard execution. In order to enter or modify model and package fields user can use associated `Browse` button.


VIDE model Selection

Choose VIDE model to import web service to:

VIDE Example Project
consuming-example.uml

Package Selection

Choose a package:

Consumed

ConsumedTest

Figure 109: Pre-populated fill for required options

Container is always prefilled with (filtered) consumed Web Service namespace but can be changed by a user see Figure 109.

Step 6: After clicking `Create` button wizard modifies model (`consuming-example.uml` file Figure 110) by importing Web Service proxy stub into it.

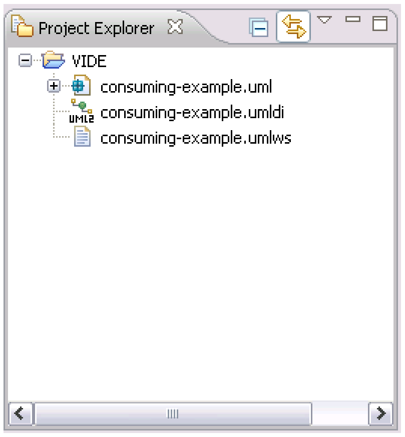


Figure 110: Wizard modifies model

All added model elements are contained in the package. It includes (exactly one) class marked with «ConsumedService» stereotype named after port type and types definitions its methods operate on. From now on we will refer to the whole component as (imported) Web Service proxy.

Additionally to the model modification wizard creates (or extends if it already exists) platform specific Web Services options file (`consuming-example.umlws` on the above picture). Further information regarding the file format will be included in related WP6 document.

Step 7: Users add recently imported Web Service into diagram from Outline pane seeFigure 111.

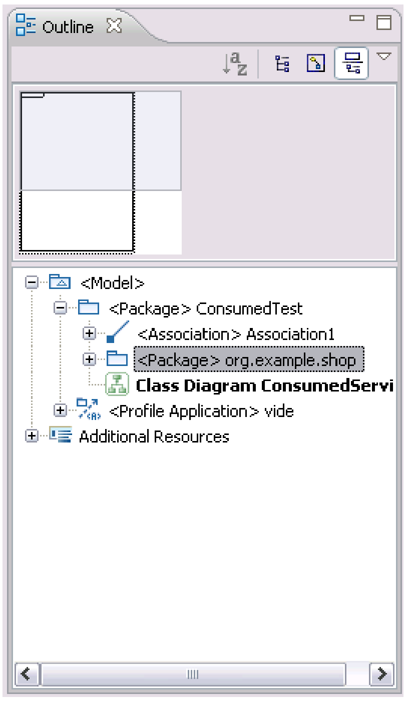

Outline
<Model>
<Package> ConsumedTest
<Association> Association1
<Package> org.example.shop
Class Diagram ConsumedServi
<Profile Application> vide
Additional Resources

Figure 111: User adds recently imported web service

Diagram after addition of `org.example.shop` package is depicted in Figure 112:

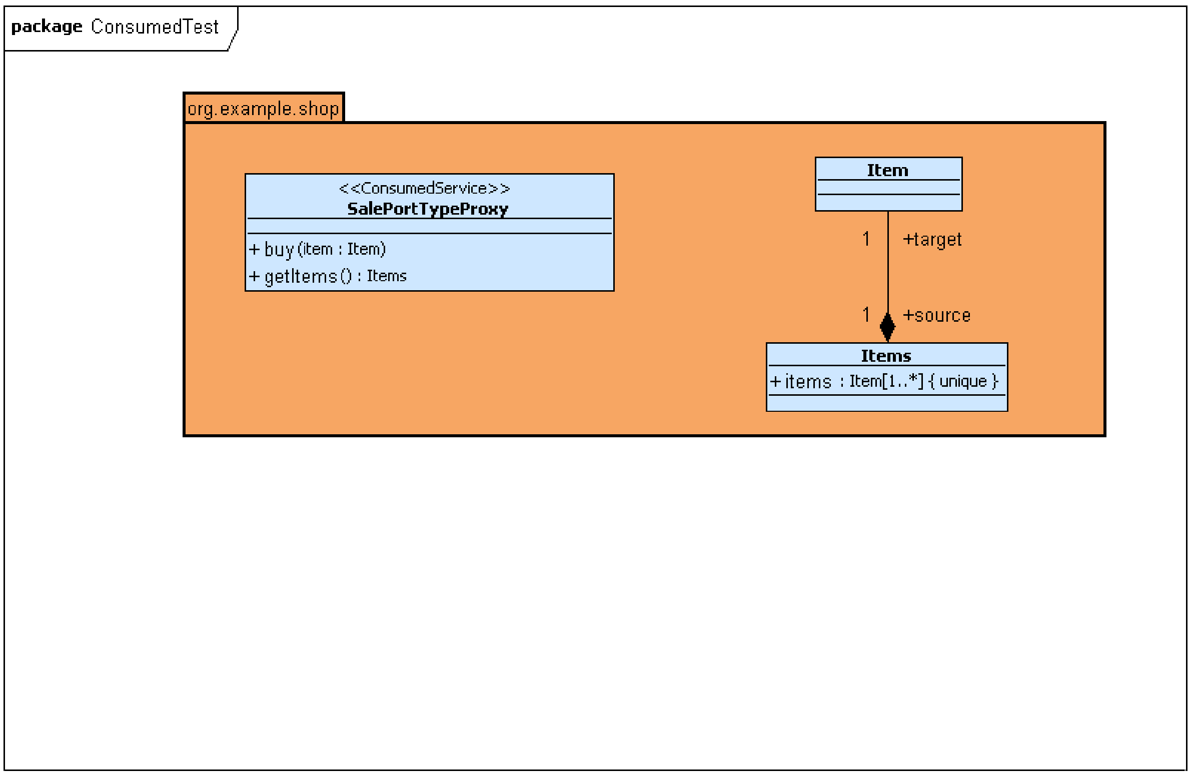


Figure 112: Diagram after addition of org.example.shop package

Step 8: User adds packages and contained classes (see Figure 113), which will use consumed `SalePortTypeProxy`:

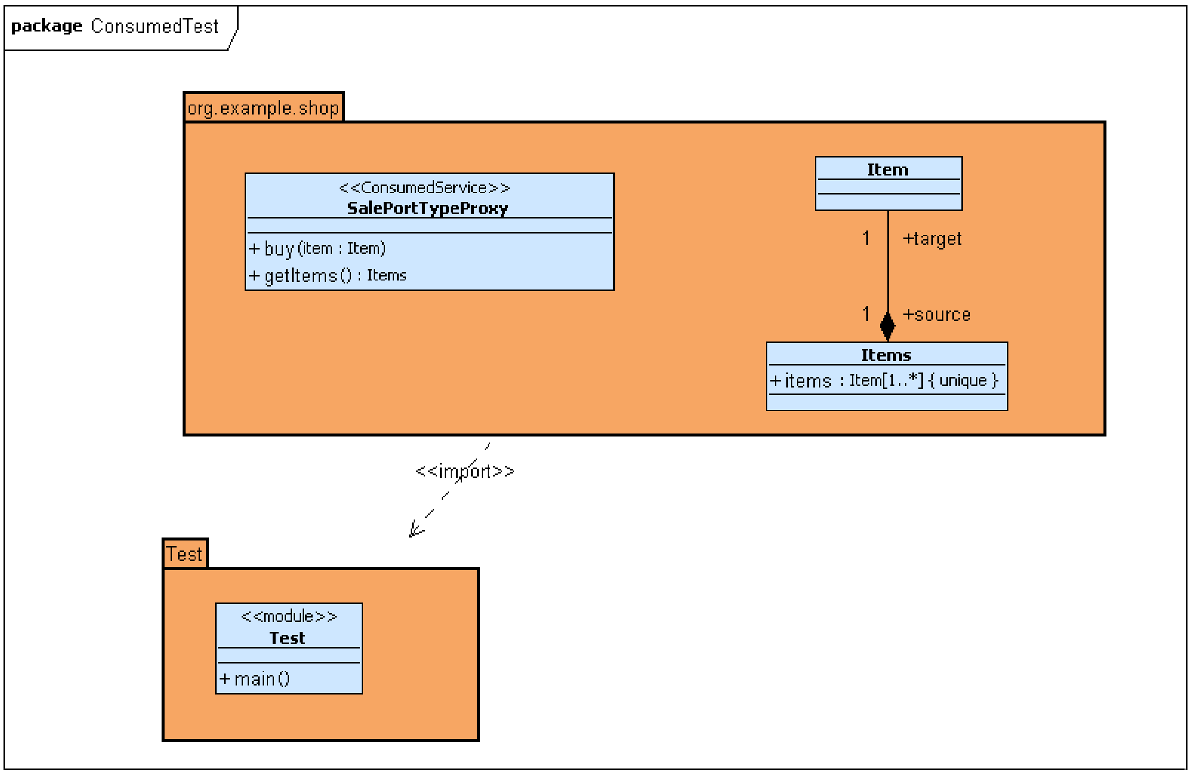


Figure 113: User adds package and contained classes

11.4.3.2 Scenario 2: Publishing class as a Web Service

The task is to expose class from the model as a Web Service, which then can be called from the outside of VIDE system.

Step 1: User creates class to be exposed see Figure 114. Naming of model elements, which will be exposed (class and containing package) should follow naming conventions described in the previous section.

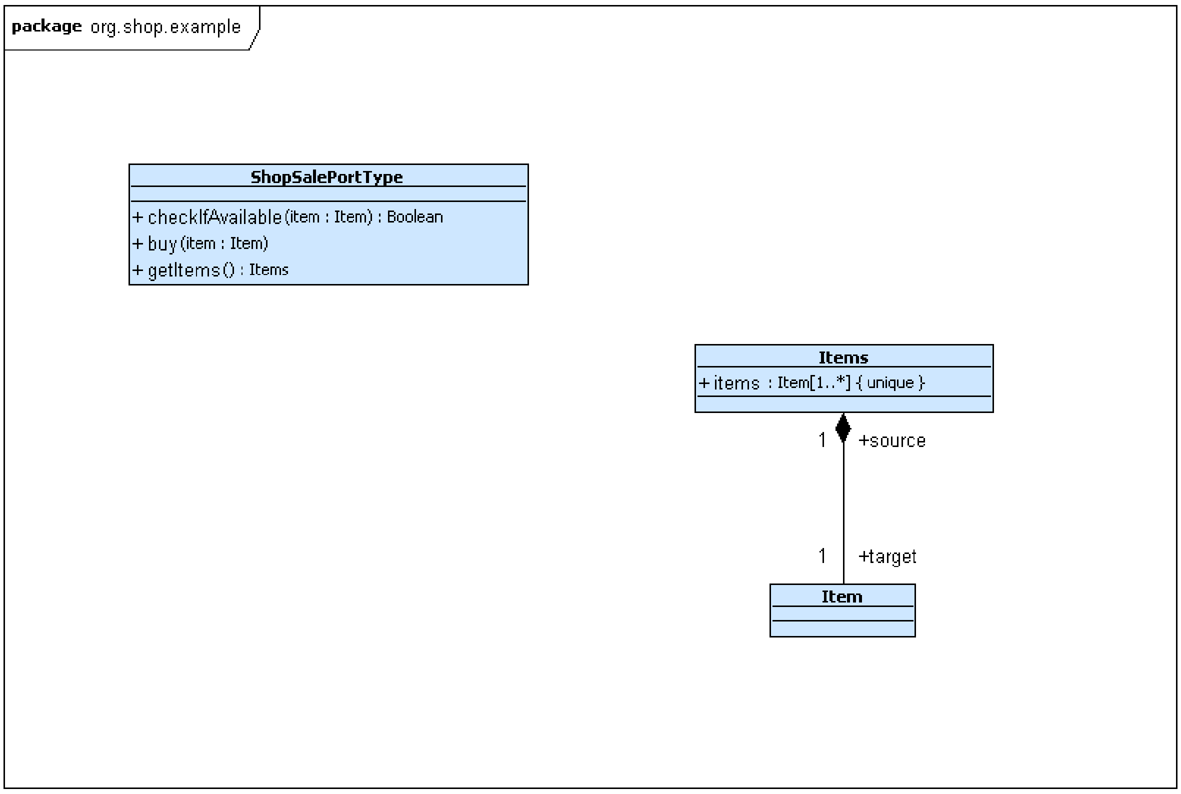
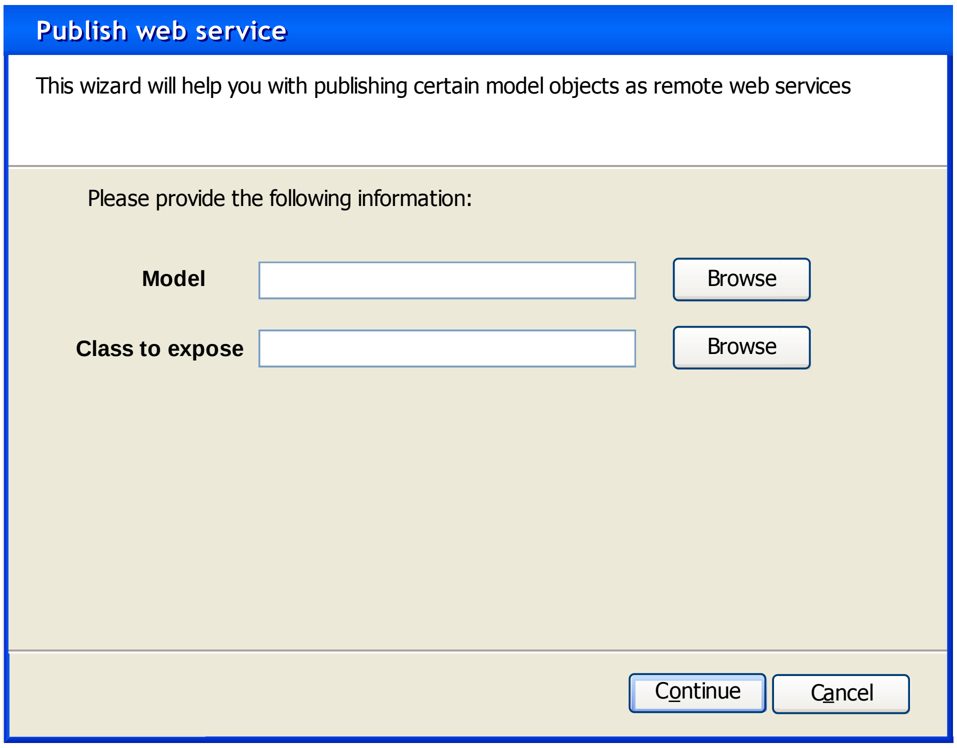


Figure 114: User creates class to be exposed

In contrast to the consuming - in this scenario the container package may include multiple classes exposed (as Web Services). User is not restricted in any way here except that it is not possible to expose `«ConsumedService»` or already exposed one. It is required by compilation schema which will be described in WP6 part of Web Services in VIDE documentation.

Step 2: User chooses `Publish as Web Service` option from main VIDE menu see Figure 115. Wizard appears. On its first screen user provides information about class to be exposed. Similarly as for consuming fields from this step may be pre filled if appropriate element was active on wizard start.



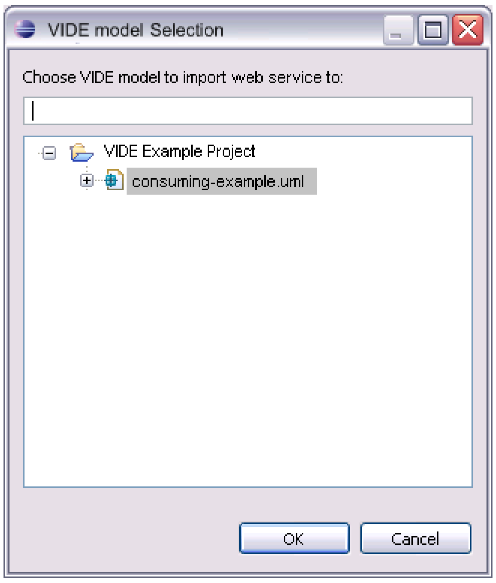
This wizard will help you with publishing certain model objects as remote web services

Please provide the following information:

Model
Class to expose

Figure 115: Publish as Web Service option

Step 3: User fills in `Model` and `Class to expose` parameters using selection windows triggered by associated browse button. Selection window for model shows all files with `uml` extension from current workspace:



Choose VIDE model to import web service to:

VIDE Example Project
consuming-example.uml

Figure 116: VIDE model selection screen

Class Selection window filters only classes from previously chosen model:

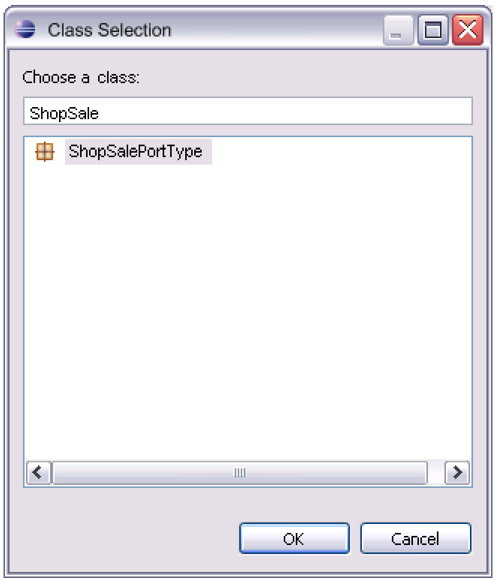
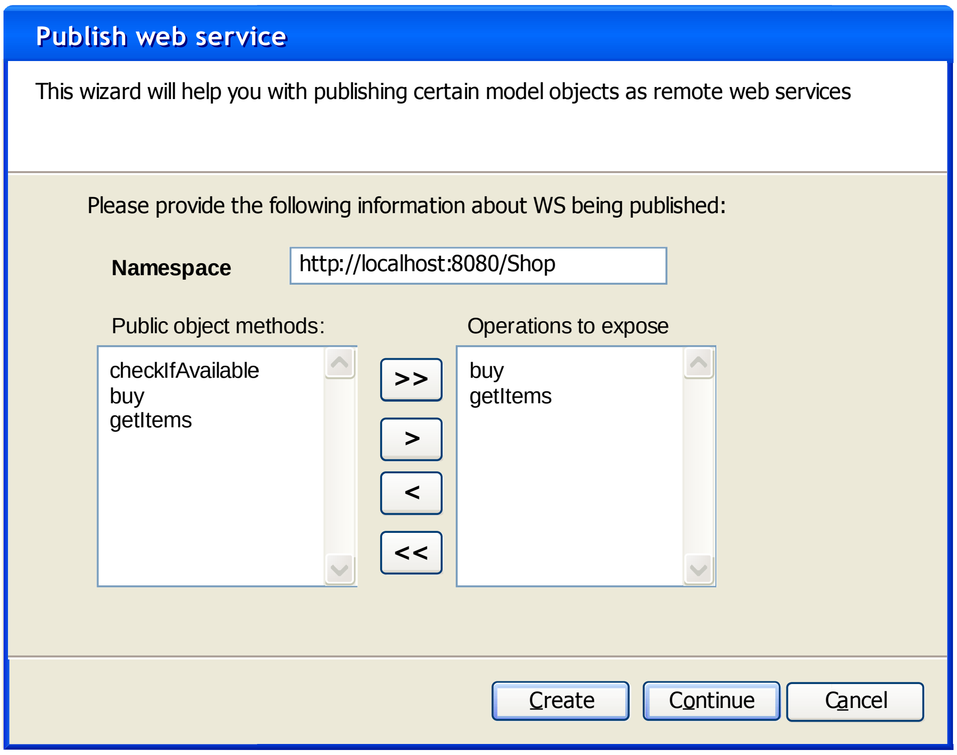


Figure 117: VIDE Class Selection screen

Step 4: On the next step namespace textbox value is pre-populated with exposed class container package's (filtered) global name. User tweaks it to meet his/hers needs and then specifies operations to be exposed see Figure 118. Unless at least one method is moved to `Operations to expose` panel `Create` and `Continue` buttons are disabled.



Publish web service

This wizard will help you with publishing certain model objects as remote web services

Please provide the following information about WS being published:

Namespace http://localhost:8080/Shop

Public object methods: checkIfAvailable, buy, getItems

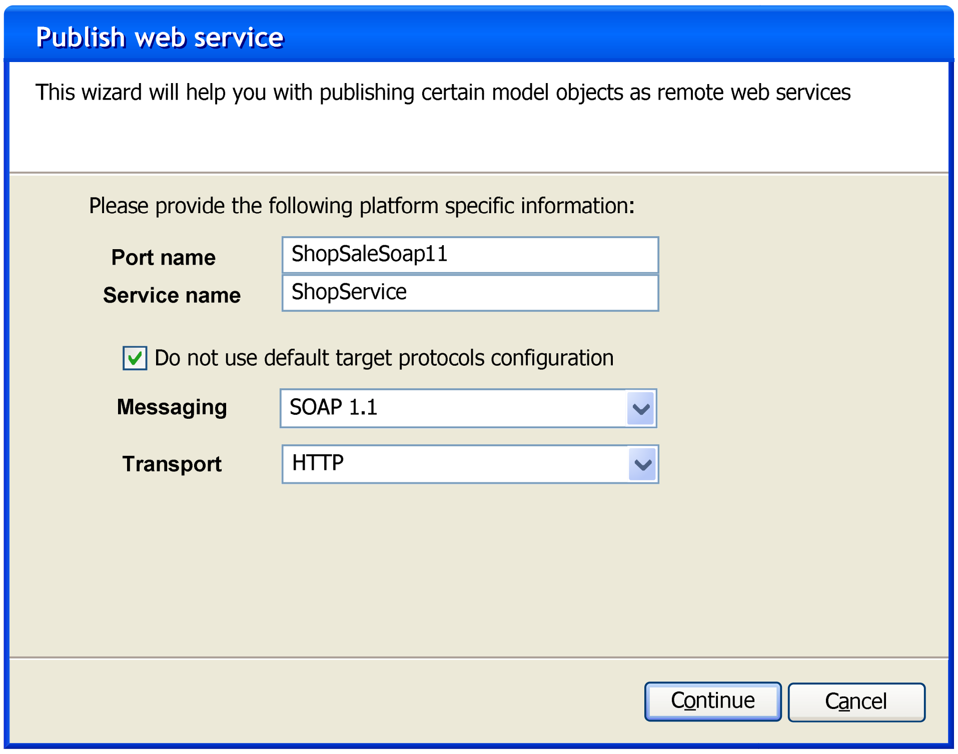
Operations to expose: buy, getItems

Figure 118: User specifies operations to be exposed

Step 5: User may choose Create or Continue option see Figure 119. The first one allows to expose Web Service with set of default (concrete / platform specific) options. The second one allow user to review and change them.

Platform specific options include communication protocols, port and service names. Default options are SOAP 1.1 as messaging protocol (document wrapped/literal style) and HTTP 1.0 (with published class name suffixed with “Endpoint” as URL address) for transport layer. Port defaults to published class name with protocol name appended. Service defaults to class name with “Service” suffix appended.

User specifies `port` and `service` names. Then he/she chooses to specify custom communication protocols for transport and messaging layers. It is done by checking appropriate checkbox. In case of HTTP a URL is provided then.



Publish web service

This wizard will help you with publishing certain model objects as remote web services

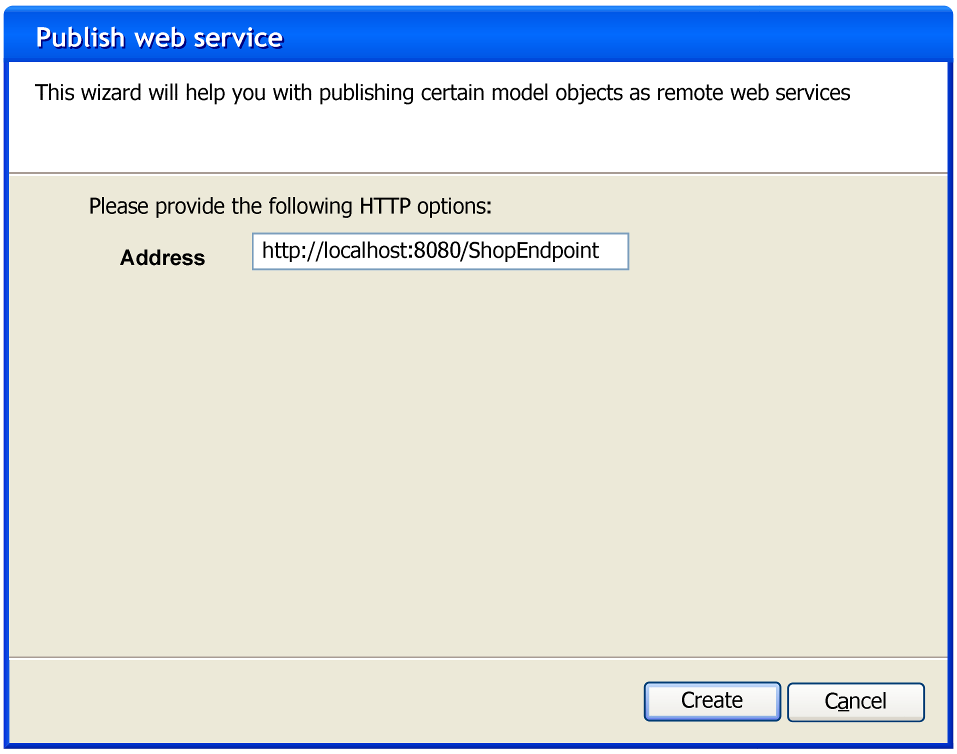
Please provide the following platform specific information:

Port name	ShopSaleSoap11
Service name	ShopService

[✓] Do not use default target protocols configuration

Messaging	SOAP 1.1
Transport	HTTP

Figure 119: Specifying protocols



This wizard will help you with publishing certain model objects as remote web services

Please provide the following HTTP options:

Address: http://localhost:8080/ShopEndpoint

Figure 120: URL address

Step 6: After clicking Create button wizard modifies model (`publishing-example.uml` file below) by annotating chosen class with set of defined stereotypes. They are automatically viewable in designer (after diagram refresh) as sown in Figure 121:

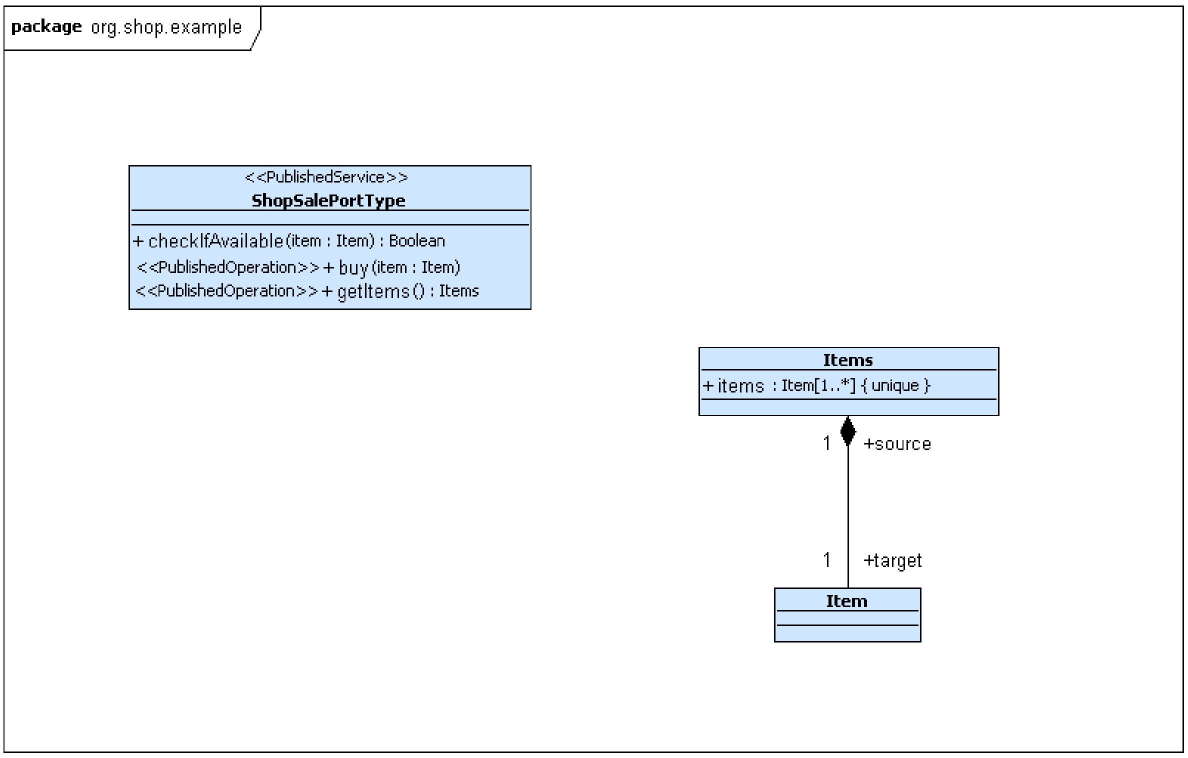

package org.shop.example
<<PublishedService>>
ShopSalePortType
+ checkIfAvailable (item : Item) : Boolean
<<PublishedOperation>> + buy (item : Item)
<<PublishedOperation>> + getItems () : Items
Items
+ items : Item[1..*] { unique }
1
+source
1
+target
Item

Figure 121: Adding defined stereotypes

Step 7: After user completes the above step – Web Service exposing process is complete and model is ready to be executed. Description of execution and consuming using external clients is outside of the scope of this document.

12. Conclusions

In this deliverable we have outlined the work that has been completed as part of Work Package 5. This involved the in depth study of 46 MDA tools. The list of tools was taken from the OMG list of MDA compliant tools available from their web site. A list of comparison features was constructed from the literature and the tools were reviewed. The results are shown in Chapter 2 along with some requirements for the VIDE interface that had not been specifically outlined in Deliverable D1.

The following two chapters reported the result of research into the literature. Chapter 3 discussed methods of evaluation that would allow the Consortium to investigate the exploratory prototype. A number of methods were reviewed and both Focus Groups and the Cognitive Dimensions Framework were selected for the evaluation. Chapter 4 reviewed the literature in Software Visualisation and made a number of recommendations for requirements with respect to component design in particular. The Visual Programming and Diagramming research had some useful recommendations from the area of class diagram layout and the use of secondary notation.

Chapter 5 discussed the work that was completed investigating the graphical user interfaces of four established modeling environments. Various UML models were created in each of the tools and the process investigated to highlight any shortcomings. This was discussed in the work using the Cognitive Dimensions Framework and allowed a number of requirements to be added. The following chapter, Chapter 6, outlined the specification of the Exploratory prototype that was developed using the initial requirements.

Chapter 7 drew together the feedback and evaluation of the exploratory prototype and the requirements that were obtained in the preceding work. These requirements were collated and cross referenced with the requirements from D1 to give a complete list. This was used to create the definitive prototype specified in Chapter 8.

Chapter 9 provided the specification of the Visual Code Editor for State Visualisation Syntax. The work outlined some of the research in the area which informed the design. A number of screen shots are detailed. This is followed by Chapter 10 which described the Visual Expression Builder. This is a graphical editor which allows a user to specify expressions using the Object Constraint Language. Finally Chapter 11 explored the VIDE approach to dealing with legacy applications which involve the use of Web Services.

13. References

1. VIDE, *Standards, Technological and Research-Base for the VIDE Project, Project Evaluation Criteria and User Requirements Definition*, in *Deliverable 1.1 of the VIDE Project*. 2007, Framework 6 EU Commission.
2. Softeam. *Objecteering*. 2007 [Last Accessed 20 December 2007]; Available from: http://www.objecteering.com/.
3. PathFinderSolutions. *PathMate MDA* 2007 [Last Accessed 19 December 2007]; Available from: http://www.pathfindermda.com/products/index.php.
4. NeosightTechnologies. *BoldExpress Studio*. 2007 [Last Accessed 19 December 2007]; Available from: http://www.neosight.com.
5. SoftMetaWare. *Generative Model Transformer Project*. 2007 [Last Accessed 20 December 2007]; Available from: http://www.softmetaware.com.
6. IBM. *Rational Software Architect*. 2007 [Last Accessed 20 December 2007]; Available from: http://www-306.ibm.com/siftware/awdtools/architect/swarchitect.
7. SelectBusinessSolutions. *Select Component Factory or Select Solution for MDA*. 2007 [Last Accessed 20 December 2007]; Available from: http://www.selectbs.com.
8. InteractiveObjects. *ArcStyler MDA tool (http://www.arcstyler.com/)*. 2006 [Last Accessed 08/2006 2006]; Available from: http://www.arcstyler.com/.
9. Mia-Software. *Model-In-Action*. 2007 [Last Accessed 19 December 2007 2007]; Available from: http://www.mia-software.com.
10. IBM. *Rational XDE Developer (http://www-306.ibm.com/software/uk/rational/awdtools/swdeveloper.html)*. 2006 [Last Accessed 08/2006 2006]; Available from: http://www-306.ibm.com/software/uk/rational/awdtools/swdeveloper.html.
11. Compuware. *OptimalJ MDA tool* 2007 [Last Accessed 20 December 2007]; Available from: http://www.compuware.com/products/optimalj/.
12. KnowGravity. *CASSANDRA/xUML*. 2007 [Last Accessed 19 December 2007]; Available from: http://www.knowgravity.com/eng/index.htm.
13. CodagenTechnologies. *Codagen Architect for MDA*. 2007 [Last Accessed 20 December 2007]; Available from: http://www.manyeta.com.
14. Telelogic. *TAU Generation2*. 2007 [Last Accessed 20 December 2007]; Available from: http://www.telelogic.com/.
15. Borland. *Together Architect* 2007 [Last Accessed 20 December 2007]; Available from: http://www.borland.com/us/products/together/index.html.
16. Metamaxim. *modelscope*. 2007 [Last Accessed 20 December 2007]; Available from: http://www.metamaxim.com.
17. CodelessTechnology. *Codeless*. 2007 [Last Accessed 20 December 2007]; Available from: http://www.codeless.com.
18. Object Management Group. *Request for Information: MDA Tool Capabilities*. 2006 [Last Accessed 26 July 2007]; Available from: http://www.omg.org/docs/mda-user/06-08-01.pdf.
19. Object Management Group. *Request for Proposal: MDA Tool Component*. 2006 [Last Accessed 26 July 2007]; Available from: http://www.omg.org/docs/ad/06-06-09.pdf.
20. OMG. *OMG MDA Vendor Driectory (http://mda-directory.omg.org/)*. 2006 [Last Accessed 7/2006 2006]; Available from: http://mda-directory.omg.org/.
21. ArchitectureBoardORMSC. *MDA Guide Version 1.0.1*. 2003 [Last Accessed 19 September 2007].
22. Lyngset, T.E. and T. Vasset, *MDA (Model Driven Architecture)*, in *Department of Computer and Information Science*. 2003, Norwegian University of Science and Technology (NTNU): Trondheim.
23. Soley, R. and OMG Staff Strategy Group. *Model Driven Architecture*. 2000 [Last Accessed 24 September 2007]; Available from: ftp://ftp.omg.org/pub/docs/omg/00-11-05.pdf.
24. Architecture Board ORMSC. *Model Driven Architecture (MDA)*. 2001 [Last Accessed 19 September 2007].
25. Poole, J.D. *Model-Driven Architecture: Vision, Standards and Emerging Technologies*. in *European Conference on Object Oriented Programming (ECOOP)*. 2001. Budapest, Hungary.
26. MODATEL Consortium. *Assessment of the Model Driven Technologies - Foundations and Key Technologies*. 2002 [Last Accessed 20 October 2007]; Available from: http://www.modatel.org.
27. Architecture Board MDA Drafting Team. *Model Driven Architecture: A Technical Perspective*. 2001 [Last Accessed 28 August 2007]; Available from: http://www.omg.org/docs/ormsc/01-07-01.pdf.
28. Kleppe, A., J. Warmer, and W. Bast, *MDA Explained: The Model Driven Architecture--Practice and Promise* 2003, Boston: Addison-Wesley Professional.

29. Berrisford, G. *Why IT Veterans are sceptical about MDA*. in *Second European Workshop on Model Driven Architecture (MDA)*. 2004. Canterbury, UK.
30. Bezivin, J. and S. Gerard. *A Preliminary Identification of MDA Components*. 2002 [Last Accessed 22 November 2007]; Available from: http://www.softmetaware.com/oopsla2002/bezivinj.pdf.
31. Tratt, L., *Model transformations and tool integration.* Software Systems Modelling, 2005. **4**(2).
32. Blanc, X., S. Bouzitouna, and M.-P. Gervais. *A Critical Analysis of MDA Standards through an Implementation: the ModFact Tool*. in *Proc. of the First European Workshop on Model Driven Architecture with Emphasis on Industrial Applications (EWMDA-IA'04)*. 2004. Enschede, the Netherlands.
33. Desfray, P. *MDA – When a major software industry trend meets our toolset, implemented since 1994*. 2001 [Last Accessed 10 December 2007]; Available from: http://www.omg.org/mda/mda_files/MDA-Softeam-WhitePaper.pdf.
34. Flater, D. *Impact of Model-Driven Standards*. in *35th International Conference on System Sciences*. 2002. Hawaii
35. Ambrosio, J. (2003) *Tools for the Code Generation*. Application Development Trends **Volume**,
36. Seidewitz, E. *A real-world example of MDA without automation*. 2004 [Last Accessed 28 September 2007]; Available from: http://www.omg.org/docs/mda-user/04-08-02.pdf.
37. Mellor, S.J., et al., *MDA Distilled: Principles of Model-Driven Architecture*. 2004: Addison-Wesley.
38. OMG. *MDA Committed Products*. 2007 [Last Accessed 1 December 2007]; Available from: http://www.omg.org/mda/committed-products.htm.
39. Adaptive. *Adaptive Software (http://www.adaptive.com/homelinks/modelmgt.html)*. 2006 09/2006 [Last Accessed.
40. Aonix. *Ameos toolset for MDA (http://www.aonix.com/ameos.html)*. 2006 [Last Accessed 08/2006 2006]; Available from: http://www.aonix.com/ameos.html.
41. Artisan. *Real Time Studio*. 2007 [Last Accessed 20 December 2007]; Available from: http://www.artisansw.com.
42. BITPlan. *smartGenerator*. 2007 [Last Accessed 20 December 2007]; Available from: http://bitplan.com.
43. CalkeyTechnologies. *Caboom*. 2007 [Last Accessed 20 December 2007]; Available from: http://wwwcalkey.com/caboom.htm.
44. Calytrix. *SIMplicity*. 2007 [Last Accessed 20 December 2007]; Available from: http://www.calytrix.com.
45. Consyst. *REP++ Studio*. 2007 [Last Accessed 20 December 2007]; Available from: http://consyst-sql.com/a/WWW/Accueil/Accueil.html.
46. DataAccessTechnologies. *Component -X*. 2007 [Last Accessed 20 December 2007]; Available from: http://www.enterprise-component.com.
47. DomainSolutions. *CodeGenie*. 2007 [Last Accessed 20 December 2007]; Available from: http://www.ooagenerator.com/codegenie.htm or http://www.domsols.com.
48. DotNetBuilders. *Constructor toolset for MDRAD*. 2006 [Last Accessed 20 December 2007]; Available from: http://www.dotnetbuilders.com/constructor.aspx.
49. E2E. *Bridge*. 2007 [Last Accessed 20 December 2007 2007]; Available from: http://www.e2ebridge.com.
50. Gentastic. *e-GEN*. 2007 [Last Accessed 20 December 2007 2007]; Available from: http://www.gentastic.com/e_GEN/e_GenApproach.html.
51. IKV++. *Medini Product Family (was m2C)*. 2007 [Last Accessed 20 December 2007 2007]; Available from: http://www.ikv.de.
52. Telelogic. *Rhapsody*. 2007 [Last Accessed 20 December 2007]; Available from: http://modeling.telelogic.com.
53. innoQ. *iQgen*. 2007 [Last Accessed 19 December 2007]; Available from: http://www.inoq.com/iqgen.
54. KabiraTechnologiesInc. *Kabira Transaction Platform and Kabira Accelerator*. 2007 [Last Accessed 19 December 2007]; Available from: http://www.kabira.com.
55. KennedyCarter. *iUML*. 2007 [Last Accessed 20 December 2007]; Available from: http://kc.com.
56. KennedyCarter. *iCCG*. 2007 [Last Accessed 20 December 2007]; Available from: http://www.kc.com.
57. Liantis. *Xcoder*. 2007 [Last Accessed 20 December 2007]; Available from: http://www.liantis.com.
58. MentorGraphics. *BridgePoint/xtUML or EDGE UML Suite*. 2007 [Last Accessed 19 December 2007]; Available from: http://wwww.mentor.com.
59. JBoss. *MetaMatrix Data Services Platform*. 2007 [Last Accessed 19 December 2007]; Available from: http://ww.redhat.com/metamatrix/.
60. MID. *Innovator*. 2007 [Last Accessed 20 December 2007]; Available from: http://www.mid.de.

61. Netfective. *Blu Age*. 2007 [Last Accessed 20 December 2007]; Available from: http://www.bluage.com.
62. ObjectFrontier. *FrontierSuite*. 2007 [Last Accessed 19 December 2007]; Available from: http://www.objectfrontier.com.
63. OutlineSystemsInc. *PowerRAD*. 2007 [Last Accessed 20 December 2007]; Available from: http://www.outlinesys.com.
64. PlasticSoftware. *agora Plastic 2005*. 2007 [Last Accessed 20 December 2007]; Available from: http://www.plasticsoftware.com/.
65. realMethods. *Framework*. 2007 [Last Accessed 20 December 2007]; Available from: http://www.realmethods.com/index.html.
66. SoftarisPtyLtd. *MetaBoss*. 2007 [Last Accessed 20 December 2007 2007]; Available from: http://www.metaboss.com.
67. CARETechnologiesSA/SOSYInc. *OlivaNova Model Execution System*. 2007 [Last Accessed 20 December 2007]; Available from: http://www.sosyin.com.
68. SparxSystems. *Enterprise Architect* 2007 [Last Accessed 20 December 2007]; Available from: http://www.sparxsystems.com.
69. TataConsultancyServices. *MasterCraft*. 2007 [Last Accessed 20 December 2007]; Available from: http://www.tatamastercraft.com/index.htm / http://www.tata.com/index.htm.
70. TechOne. *ACE*. 2007 [Last Accessed 20 December 2007]; Available from: http://www.techone.com.
71. Gediga, G., K.-C. Hamborg, and I. Duntsch, *Evaluation of Software Systems*, in *Encyclopedia of Computer Science and Technology*, A. Kent and J.G. Williams, Editors. 2001, CRC.
72. Te'eni, D., J. Carey, and P. Zhang, *Human Computer Interaction: Developing Effective Organizational Information Systems*. 2007, Hoboken, New Jersey, US: John Wiley and Sons Inc.
73. De Souza, F. and N. Bevan. *The Use of Guidelines in Menu Interface Design: Evaluation of a draft standard*. in *IFIP INTERACT 90: Human Computer Interaction*. 1990.
74. Bevan, N. *International standards for HCI and usability*. 2006 [Last Accessed 25/03/2007 2007]; Available from: http://www.usabilitynet.org/home.htm.
75. Bevan, N., *Quality in Use: Meeting User Needs for Quality*. Journal of Systems and Software. 1999. **49**(1): p. 89-96.
76. Nielsen, J., *Usability Engineering*. 1993, New York: AP Professional.
77. Ivory, M.Y. and M.A. Hearst, *The State of the Art in Automating Usability Evaluation of User Interfaces*. ACM Computing Surveys, 2001. **33**(4): p. 470-516.
78. Nielsen, J. and R.L. Mack, eds. *Usability Inspection Methods*. 1994, John Wiley and Sons Inc.
79. Vredenburg, K., et al. *A Survey of User-Centered Design Practice*. in *Conference on Human Factors in Computing Systems* 2002. Minneapolis, Minnesota.
80. Schneiderman, B., *Designing the user interface*. 3rd ed. 1998: Addison-Wesley.
81. Sharp, H., Y. Rogers, and J. Preece, *Interaction Design:Beyond Human Computer Interaction*. 2nd ed. 2007, Chichester: John Wiley and Sons.
82. UPA. *Principles for Usable Design*. Usability Body of Knowledge 2005 [Last Accessed 28 November 2007]; Available from: http://www.usabilitybok.org/design/p287.
83. Karat, C.-M., *A Comparison of User Interface Evauluation Methods*, in *Usability Inspection Methods*, J. Nielsen and R.L. Mack, Editors. 1994, John Wiley and Sons: USA.
84. Bailey, B., *The effectiveness of heuristic evaluations vs usability testing*, in *HFI User Interface Design Newsletter*. 2001.
85. Bias, R.G., *The Pluralistic Usability Walkthrough:Coordinated Empathies*, in *Usability Inspection Methods*, J. Nielsen and R.L. Mack, Editors. 1994, John Wiley and Sons. p. 63-76.
86. Nielsen, J. *The Use and Misuse of Focus Groups*. 1997 [Last Accessed 30 October 2007]; Available from: http://www.useit.com/papers/focusgroups.html.
87. Beynon, D., P. Turner, and S. Turner, *Designing Interactive Systems: People,Activities,Contexts,Technologies*. 2005, Harlow, England: Addison-Wesley.
88. Green, T.R.G. and M. Petre, *Usability Analysis of Visual Programming Environments: A 'Cognitive Dimensions' Framework*. Journal of Visual Languages and Computing, 1996. **7**: p. 131-174.
89. Green, T.R.G., *Cognitive dimensions of notations*, in *People and Computers V*, A. Sutcliffe and L. Macaulay, Editors. 1989, Cambridge University Press: Cambridge, UK. p. 443-460.
90. Blackwell, A.F., et al. *Cognitive Dimensions of Notations:Design Tools for Cognitive Technology*. in *Cognitve Technology 2001*. 2001: Springer-Verlag.
91. Blackwell, A.F., et al., *Cognitive Factors in Programming with Diagrams*. Artificial Intelligence Review, 2001. **15**(1/2): p. 95-114.

92. Blackwell, A.F. and T.R.G. Green, *Notational Systems - The Cognitive Dimensions Framework*, in *HCI Models, Theories and Frameworks: Toward a multidisciplinary science*, J. Carroll, Editor. 2003, Morgan Kaufmann: San Fransisco, USA.
93. Wixom, D., *Evaluating Usability Methods.* interactions, 2003. **July/August**: p. 29-34.
94. Price, B.A., I.S. Small, and R.M. Baecker. *A Taxonomy of Software Visualisation*. in *25th Hawaiian International Conference of System Sciences*. 1992. Kauai,HI.
95. Young, P. and M. Munro. *Visualising Software in Virtual Reality*. in *International Workshop on Program Comprehension (IWPC'98)*. 1998. Ischia, Italy.
96. Maletic, J.I., A. Marcus, and M.L. Collard. *A Task Oriented View of Software Visualisation*. in *IEEE First International Workshop on Visualizing Software for Understanding and Analysis*. 2002: IEEE Computer Society Press.
97. Petre, M. and E. de Quincey, *A gentle overview of software visualisation*, in *Psychology of Programming Interest Group (PPIG) Newsletter*. 2006.
98. Berenbach, B. and O. Gotel. *International Workshop on Requirements Engineering Visualization (REV '07)*. 2007 [Last Accessed 2 January 2008]; Available from: http://csis.pace.edu/~ogotel/professional/REV07.html.
99. Benedikt, M., *Cyberspace: Some proposals*, in *Cyberspace: First Steps*. 1991, MIT Press. p. 119-124.
100. Whitley, K.N., *Visual Programming Languages and the Empirical Evidence For and Against.* Journal of Visual Languages and Computing, 1997. **8**: p. 109-142.
101. Pane, J.F. and B.A. Myers, *Usability Issues in the Design of Novice Programming Systems*. 1996, School of Computer Science, Carnegie Mellon University: Pittsburgh, Pennsylvania, USA.
102. Petre, M., *Why Looking Isn't Always Seeing: Readership Skills and Graphical Programming.* Communications of the ACM, 1996. **38**(6): p. 33-44.
103. Sutcliffe, A.G., *Human - Computer Interface Design*. 2nd ed. 1995, Basingstoke: Macmillan Press.
104. Faulkner, X., *Usability Engineering*. Grassroots. 2000, London: Macmillan Press Ltd. 244.
105. Crowle, S., et al., *Users*, in *Deliverable 1.1 of the VIDE Project*. 2007, Framework 6 EU Commission.
106. Robins, A., J. Rountree, and N. Rountree, *Learning and Teaching Programming: A Review and Discussion.* Computer Science Education, 2003. **13**(2): p. 137-172.
107. Eichelberger, H. *Nice Class Diagrams Admit Good Design?* . in *ACM Symposium on Software Visualisation*. 2003. San Diego, California, USA: ACM Press.
108. Diskin, Z. *Visualization vs. specification in diagrammatic notations: A case study with the UML*. in *Diagrammatic Representation and Inference. 2nd Int. Conf. on the Theory and Applications of Diagrams*. 2002: Springer.
109. Eiglsperger, M., M. Kaufmann, and M. Siebenhaller. *A Topology-Shape-Metrics Approach for the Automatic Layout of UML Class Diagrams*. in *ACM Symposium on Software Visualization*. 2003. San Diego, California, USA: ACM Press.
110. Seemann, J. *Extending the Sugiyama Algorithm for Drawing UML Class Diagrams:Towards Automatic Layout of Object-Oriented Software Diagrams*. in *5th International Symposium of Graph Drawing*. 1997. Rome, Italy: Springer-Verlag.
111. Lee, Y.Y., C. Lin, and H. Yen. *Mental Map Preserving Graph Drawing Using Simulated Annealing*. in *Asia Pacific Symposium on Information Visualization*. 2006. Tokyo, Japan.
112. Wood, D., *Minimising the Number of Bends and Volume in 3-Dimensional Orthogonal Graph Drawings with a Diagonal Vertex Layout.* Algorithmica, 2004. **39**: p. 235-253.
113. Mehra, A., J. Grundy, and J. Hosking. *A Generic Approach to Supporting Diagram Differencing and Merging for Collaborative Design*. in *20th International Conference on Automated Software Engineering*. 2005. Long Beach, CA, USA: ACM.
114. Chidamber, S.R. and C.F. Kemerer, *A Metrics Suite for Object Oriented Design.* IEEE Transactions on Software Engineering, 1994. **20**(6): p. 476-493.
115. Purchase, H.C., et al. *Graph drawing aesthetics and the comprehension of UML class diagrams: an empirical study*. in *Australian Symposium on Information Visualisation*. 2001. Sydney, Australia.
116. Gutwenger, C., et al. *A new approach for visualizing UML class diagrams*. in *ACM Symposium on Software Visualization* 2003. San Diego, CA, USA: ACM.
117. Hahn, J. and K. J., *Why Are Some Diagrams Easier to Work With? Effects of Diasgrammatic Representation on the Cognitive Integration Process of Systems Analysis and Design.* ACM Transactions on Computer-Human Interaction, 1999. **6**(3): p. 181-213.
118. Myers, B.A., J.F. Pane, and A. Ko, *Natural Programming Languages and Environments.* Communications of the ACM, 2004. **47**(9): p. 47-52.
119. Maloney, J., et al. *Scratch: A Sneak Preview*. in *Second International Conference on Creating, Connecting and Collaborating through Computing*. 2004. Kyoto, Japan.

120. Anon. *Imagine creating software without a single line of code*. 2007 [Last Accessed 12 December 2007; Presentation]. Available from: http://www.limnor.com.
121. Anon. *MyDesk 2.0* 2007 [Last Accessed 12 December 2007; Available from: http://www.genusoft.net/english/productshow.asp?articleid=139.
122. MyDesk. *MyDesk 2.0* 2007 [Last Accessed 12 December 2007; Available from: http://www.genusoft.net/english/productshow.asp?articleid=139.
123. Scratch. *Scratch - imagine, program, share*. 2007 [Last Accessed 20 July 2007]; Available from: http://scratch.mit.edu/.
124. Newman, M.W., et al., *DENIM: An Informal Web Site design Tool Inspired by Observations of Practice*. Human Computer Interaction, 2003. **18**: p. 259-324.
125. Objecteering. *Objecteering/UML (http://www.objecteering.com/)*. 2007 [Last Accessed 08/2007 2006]; Available from: http://www.objecteering.com/.
126. Borland. *Together Architect (http://www.borland.com/us/products/together/index.html)*. 2006 [Last Accessed 07/2006 2006]; Available from: http://www.borland.com/us/products/together/index.html.
127. Eclipse.Org. *Eclipse project (http://www.eclipse.org/)*. 2006 [Last Accessed 08/2006 2006].
128. Green, T. and A. Blackwell, *Cognitive Dimensions of Information Artefacts:a tutorial*. 1998, Cambridge University & Leeds University.
129. Blackwell, A. and T. Green, *A Cognitive Dimensions Questionnaire*. 2007, Cambridge University & Leeds University.
130. Maciaszek, L.A., *Requirements Analysis and System design*. 3rd ed. 2007, Harlow, England: Addison-Wesley.
131. Bray, I., *An Introduction to Requirements Engineering*. 2002, Harlow, UK: Pearson Education Limited.
132. Sommerville, I., *Software Engineering*. 6th ed. 2001, Harlow, England: Pearson.
133. Boling, E. and T. Frick, *Holistic rapid prototyping for Web Design: Early Usability Testing is Essential*, in *Web-Based Instruction*, B. Khan, Editor. 1997, Educational Technology Publications: Englewood Cliffs, NJ. p. 319-328.
134. Gruhn, V., D. Pieper, and C. Rottgers, *MDA*. 2006: Springer.
135. Daum, B., *Rich-Client-Entiwcklung mit Eclipse 3.2*. 2007: dpunkt-Verlag.
136. TomSawyerSoftware. *Graph Analysis, Layout and Visualisation*. 2007 [Last Accessed 30 October 2007]; Available from: http://www.tomsawyer.com.
137. Stahl, T. and M. Volter, *Model driven software development*. 2006: Wiley.
138. Stahl, T., *Modellgetriebene Softwareentwicklung*. 2007: Dpunkt-verlag.
139. Plante, F. *Introducing the GMF Runtime*. 2006 [Last Accessed 23 October 2007]; Available from: http://www.eclipse.org/articles/Article-Introducing-GMF/article.html.
140. Edgar, N., et al. *Eclipse User Interface Guidelines Version 2.1*. 2004 [Last Accessed 23 October 2007]; Available from: http://www.eclipse.org/articles/Article-UI-Guidelines/contents.html.
141. Purchase, H.C., J. Allder, and D. Carrington. *User Preference of Graph Layout Aesthetics: A UML Study*. in *Graph Drawing*. 2001: Colonial Williamsburg.
142. Purchase, H.C., R.F. Cohen, and M. James. *Validating Graph Drawing Aesthetics*. in *Graph Drawing Symposium*. 1995: Springer- Verlag.
143. Purchase, H.C., E. Hoggan, and C. Gorg. *How Important is the "Mental Map" - an Empirical Investigation of a Dynamic Graph Layout Algorithm*. in *Graph Drawing*. 2007. Karlsruhe, Germany.
144. Heer, J. *prefuse - Information Visualisation Toolkit*. 2007 [Last Accessed 26 November 2007]; Available from: http://prefuse.org.
145. Gudenberg, J.W. and H. Eichelberger. *Sugi-Bib - Automatisches Layout in UML und CASE*. 2007 [Last Accessed 23 October 2007]; Available from: https://wwwi2.informatik.uni-wuerzburg.de/SugiBiB/.
146. Sugiyama, K., S. Tagawa, and M. Toda, *Methods for Visual Understanding of Hierarchical Systems* IEEE Transactions on Systems Man and Cybernetics, 1981. **SMC-11**(2): p. 109-125.
147. GraphVis. *Graph Visualization Software*. 2007 [Last Accessed 23 October 2007]; Available from: http://www.graphviz.org/.
148. Ellson, J., et al. *Graphiz-OpenSource Graph Drawing Tools*. in *Graph Drawing*. 2002. Vienna, Austria: Springer.
149. ILOG. *ILOG JViews8 - Visualisation Components*. 2007 [Last Accessed 30 October 2007]; Available from: http://www.ilog.de/products/jviews.
150. Weise, R., M. Eiglsperger, and M. Kaufmann. *yFiles: Visualisation and Automatic Layout of Graphs*. in *Graph Drawing*. 2002. Vienna, Austria: Springer.

:)