

Applying Cognitive Patterns to Support Software Tool Development

By

Hanna Farah

Thesis

**Presented to the Faculty of Graduate and Postdoctoral Studies in
partial fulfillment of the requirements for the degree
Master of Applied Science**

**Ottawa-Carleton Institute for Electrical and Computer
Engineering**

© Hanna Farah, 2006

Acknowledgements

This research is supported by IBM, NSERC, and the Consortium for Software Engineering Research.

I would like to express my gratefulness to the following:

- Dr. Timothy C. Lethbridge, my supervisor
- Marcellus Mindel, Director of the IBM Ottawa Center for Advanced Studies (CAS)
- CAS Ottawa members
- IBM staff and coop students
- Family and friends

for their support, advice, feedback, and encouragement throughout my research.

Thank you.

Abbreviations

API: Application Program Interface

BTA: Borland Together Architect

CAS: Center for Advanced Studies

CORBA: Common Object Request Broker Architecture

CVS: Concurrent Versioning System

EMF: Eclipse Modeling Framework

GEF: Graphical Editing Framework

GMF: Graphical Modeling Framework (incorporates EMF and GEF)

TME: Temporal Model Explorer

TMEP: Temporal Model Explorer Prototype

RSA: Rational Software Architect

RSD: Rational Systems Developer

RSM: Rational Software Modeler

RSx: either RSA, RSD or RSM

Abstract

This research was motivated by the development of a set of *cognitive patterns* [3], and the hypothesis that those patterns could lead to innovative and useful features in software development environments. Cognitive patterns are descriptions of ways people think and act when exploring or explaining software. In this research we focused on the *Temporal Details* cognitive patterns, which describe the dynamics of the changes in someone's mental model.

The main objective of this research, therefore, is to determine to what extent software engineering tool features could be derived from the cognitive patterns, specifically belonging to the Temporal Details hierarchy.

As the first step in our research, we analysed current tool support for cognitive patterns. The second step was to create and evaluate a list of potential new features based on the cognitive patterns. Thirdly, we developed a prototype for our most promising feature entitled Temporal Model Explorer (TME). This prototype helps people understand and manipulate the history of a software model. Users can use a slider to browse the history of the construction of a UML diagram from its point of creation to its current state. Navigation can be performed at different levels of granularity. Design rationale can be included at any point in the history. The final step was to evaluate the TME prototype with twelve participants from different backgrounds. The participants found the tool useful, and agreed that they would use it if it was available in their work environment.

Table of contents

Acknowledgements	i
Abbreviations	ii
Abstract	iii
Table of contents	iv
List of tables	vii
List of figures	viii
Chapter 1: Introduction	1
1.1 Main contribution	1
1.2 Background	2
1.3 About cognitive patterns.....	3
1.4 Problem and main hypothesis	5
1.5 Motivation.....	6
1.6 Overview of the Temporal Model Explorer feature in the context of Cognitive Patterns	6
1.7 Key results	7
1.8 Outline of the remainder of the thesis	7
Chapter 2: Review of current software development tools	8
2.1 Software development tools history	8
2.2 Current solutions and limitations	9
2.2.1 Physical division.....	10
2.2.2 Temporal division.....	12
2.2.3 Annotations, temporal annotations and design rationale documenting	16
2.2.4 Fine-grained change tracking.....	18
2.2.5 Persistent undo stacks	18
2.3 Support for other Temporal Details patterns	19
2.3.1 Rational Software Architect (RSA) 6.0	19
2.3.2 Borland Together Architect (BTA) 2006 for Eclipse	23
2.4 Concluding remarks.....	26
Chapter 3: On generating new features	27

3.1 Initial list of features.....	27
3.1.1. Details equalizer	28
3.1.2. Elements filter	28
3.1.3. Diagram slider	29
3.1.4. Diagram version slider.....	29
3.1.5. Package outline.....	29
3.1.6. Easy access to versions of a diagram.....	29
3.1.7. Search for feature.....	30
3.1.8. Explanation diagrams.....	30
3.1.9. Features list	30
3.1.10. Alternative diagrams display.....	31
3.1.11. Part swapping	31
3.1.12. Annotations	31
3.1.13. Version player	32
3.1.14. Diagram compare.....	32
3.1.15. Save compare.....	32
3.2 Evaluating and grouping the features	32
3.2.1 Participants feedback	33
3.2.2 Feature scores	36
3.3 Descriptions and analysis of the three main features	39
3.3.1 Diagram equalizer.....	40
3.3.2 Diagram player	41
3.3.3 Diagram guide	42
3.3.4 Chosen feature	43
Chapter 4: Building the TME prototype.....	44
4.1 Prototype description.....	44
4.2 Prototyping with informal tools	45
4.3 Initial prototype.....	45
4.4 Functional prototype.....	49
4.4.1 For which tool, why?	49
4.4.2 Setting up the correct development environment.....	49

4.4.3 Iterations, functionality, and challenges	50
4.4.4 Design alternatives.....	55
4.4.5 Architecture and tool integration.....	60
Chapter 5: Prototype evaluation	65
5.1 Summary of the procedure.....	65
5.2 Details of the experiment setup and procedure for Steps 2 and 3.....	67
5.2.1 Participants.....	67
5.2.2 Independent Variable.....	67
5.2.3 Variables controlled, and blocking.....	67
5.2.4 Setup of the equipment	70
5.2.5 Conduct of the experimental sessions.....	71
5.3 Results of performance improvements tests	72
5.3.1 Time and accuracy answering questions.....	72
5.3.2 Initial understanding time for participants.....	75
5.4 Participant preference	77
5.5 Additional participant feedback	80
5.5.1 Change management.....	81
5.5.2 Visualization.....	82
5.5.3 Operation.....	83
5.5.4 Navigation	83
Chapter 6: Conclusion	85
6.1 Problem statement	85
6.2 Proposed solutions and their effectiveness	85
6.3 Threats to validity.....	86
6.4 Future work.....	87
References.....	89
Appendix 1 – Software systems descriptions and designs	92
A1.1 Elections Management System	92
A1.2 Investments System for OOBank.....	95
A1.3 Airline system.....	97
Appendix 2 – Recruitment text.....	100

Appendix 3 – Informed consent, step 1	102
Appendix 4 – Informed consent step 3,4.....	104
Appendix 5 – Preference questionnaire.....	106
Appendix 6 – Raw and normalized data from user study	108
A6.1 Preference questions	108
A6.2 Timings	109
Appendix 7 – Experiment data forms	111
A7.1 Participant steps for Treatment pattern 1 t23 and 1 t32.....	111
A7.2 Participant steps for Treatment pattern 23 t1 and 32 t1	113

List of tables

Table 1 - Prototype features' weights.....	37
Table 2 - Prototype features' categorization	38
Table 3 - Prototype features' grouping to create three new main features.....	39
Table 4 - Main prototype features evaluation.....	39
Table 5 - Prototype iterations	55
Table 6 - Allocation of participants to models	69
Table 7 - Blocking of participants	70
Table 8 - All participants, answering speed	72
Table 9 - All participants, answering accuracy	73
Table 10 - Non-expert participants' answering speed.....	73
Table 11 - Non-expert participants' answering correctness	73
Table 12 - Expert participants' answering speed.....	74
Table 13 - Expert participants' answering accuracy.....	74
Table 14 - Hypotheses evaluation by participant groups	75
Table 15 – All participants' understanding times.....	75
Table 16 - Above-average participants' understanding times (all values in seconds).....	76
Table 17 - Below average participants' understanding times	76
Table 18 – Participants' over or under estimation of self-ability	77
Table 19 - Participants' preference data.....	79

Table 20 - Usability study, positive participant experiences (columns represent participants).....	80
Table 21 - Usability study, participant suggested improvements (columns indicate participants).....	81
Table 22 - Answers to preference questions.....	108
Table 23 - Performance results, timing 1	109
Table 24 - Performance results, timing 2	110

List of figures

Figure 1 - EASEL change sets [11]	11
Figure 2 - Eclipse history revisions view	12
Figure 3 - Eclipse CVS annotations view	13
Figure 4 - RSA model compare tree view.....	14
Figure 5 - RSA model compare visualization 1.....	14
Figure 6 - RSA model compare visualization 2.....	15
Figure 7 - RSA model compare visualization 3.....	15
Figure 8 – RSA UML note attached to a class	17
Figure 9 - StarTeam change request form, synopsis tab	17
Figure 10 - StarTeam change request form, solution tab	18
Figure 11 - RSA diagram elements popup menu.....	20
Figure 12 - RSA palette for class diagrams.....	20
Figure 13 - RSA popup menu for class diagrams.....	20
Figure 14 - RSA model explorer view	21
Figure 15 - RSA find and replace view.....	21
Figure 16 - Eclipse popup menu, team options	22
Figure 17 - Eclipse CVS options	23
Figure 18 - Eclipse CVS Resource History view	23
Figure 19 - BTA palette for class diagrams.....	24
Figure 20 - BTA popup menu.....	24
Figure 21 – BTA Model navigator view	24

Figure 22 - StarTeam Topic view	25
Figure 23 - StarTeam topic properties view	25
Figure 24 - StarTeam Audit view	26
Figure 25 - Details equalizer sketch.....	28
Figure 26 - Diagram equalizer sketch.....	40
Figure 27 - Diagram player sketch.....	41
Figure 28 - Diagram guide sketch.....	42
Figure 29 - Initial prototype, model screenshot with annotation.....	47
Figure 30 - Initial prototype, model screenshot with highlighting	48
Figure 31 - Initial prototype, model screenshot with annotations	48
Figure 32 – Diagram Player view (before using the term TME).....	53
Figure 33 - Final prototype screenshot.....	55
Figure 34 – TME prototype design, control package.....	61
Figure 35 – TME prototype design, integration package.....	62
Figure 36 – TME prototype design, UI functionality	63
Figure 37 – TME prototype design, information usage	64
Figure 38 - Elections system, design 1.....	93
Figure 39 - Elections system, design 2.....	94
Figure 40 - Investment system, design 1.....	96
Figure 41 - Investment system, design 2.....	97
Figure 42 - Airline system, design.....	98

Chapter 1: Introduction

The purpose of this research is to evaluate the benefits of designing application features based on Murray’s cognitive patterns [3]. Therefore, our plan is to develop a functional software prototype and evaluate its benefits to software developers. The idea behind the prototype is to add a new feature to modeling tools for better support of cognition, thus enhancing the user’s experience and performance. The research has been performed in collaboration with the IBM Centers for advanced studies, benefiting both the academic and industrial communities.

1.1 Main contribution

In this research, we have developed and evaluated a software prototype entitled ‘Temporal Model Explorer’ (TME) to help people explore, understand and manipulate the history of a software model.

The motivation for the research was the development of a set of *cognitive patterns* – descriptions of the ways people think and act when exploring or explaining software – developed by other researchers in the Knowledge-Based Reverse Engineering group at the University of Ottawa.

The main objective of our research is to determine to what extent software engineering tool features could be derived from the cognitive patterns. We specifically focused on patterns in the *temporal details* hierarchy (explained in Section 1.3).

As the first stage of our work, we studied the features in two major modeling tools: Rational Software Architect (RSA) and Borland Together Architect (BTA). This study analysed the extent to which the tools’ existing features relate to the cognitive patterns. Following this analysis, we developed, discussed and refined a list of potential new modeling tool features based on the cognitive patterns. Finally, we developed and evaluated a prototype of the feature that our study determined was the most promising.

The prototype we developed records fine-grained changes made to a UML model and allows a software engineer to review of the history of UML diagrams from their point of creation to their current state. The tool allows the author or reviewers of the diagram to edit and display *temporal annotations* associated with the state of a diagram at

a particular point in time (these are independent of UML notes and are not part of UML). The annotations could be used, for example, to provide design rationale. They would only appear when a software engineer reviews the diagram as it existed at the specific time; they then disappear.

We developed the prototype in the context of IBM's Rational suite of UML modeling products [16]. The final prototype is a plug-in for Rational Software Modeler, version 7; however, it is designed such that it should be able to work with any Eclipse-based tool that uses the Eclipse Graphical Modeling Framework [13].

We evaluated the prototype to capture the participants' preferences, experience and performance while exploring UML models. We conclude that the cognitive patterns are indeed a good basis for the development of software engineering tool features.

1.2 Background

The cognitive patterns were developed by Murray as a key element of his PhD research [1], under the direction of Lethbridge. The development of the patterns was based on extensive literature review and user studies in industrial settings [4]. The collection of patterns is divided into various categories including one called "*Temporal Details*" [3], which was our main focus in this research. Temporal Details is both a high level pattern, as well as a *pattern language* containing several sub-patterns.

It is well understood that while understanding a software system, a software engineer's mental model changes over time. The Temporal Details patterns describe the dynamics of the changes in someone's mental model [3]. The pattern can be used to describe the changes in how the mental model is expressed, e.g. using diagrams. One of the most important of the Temporal Details Patterns is called *Snapshot*. Murray put particular emphasis on developing this pattern, gathering a large amount of data and developing a comprehensive *snapshot theory*.

In Murray's research, the cognitive patterns and snapshot theory were developed with the hypothesis that they could help developers create better software engineering tools. The idea is to base tool feature development on the results of scientific studies. Resulting tools should better support aspects of human cognition, which is an important

factor in their evaluation [6]. In our research, we provide a practical implementation to test Murray’s hypothesis.

1.3 About cognitive patterns

“A cognitive pattern is a structured textual description to a recurring cognitive problem in a specific context” [3].

A cognitive pattern differs from the well-known software *design* patterns in the following manner: A design pattern captures a technique to solve a design problem, whereas a cognitive pattern captures a technique that is partly or wholly mental and that is employed potentially subconsciously by a person trying to perform any complex task. One example of a cognitive pattern is the ‘*Thinking Big*’ pattern. It describes how when the user is exploring one part of a system, he will tend to need to see the big picture in order to fully understand how the part he is studying relates to the rest of the system and how it affects the system.

Cognitive patterns are categorized in a hierarchy. Higher-level patterns may contain several related sub-patterns. Two examples of higher-level patterns [2] are *Baseline Landmark*, which describes how a person navigates his way to the understanding of a problem with constant reference to familiar parts of the system, and *Temporal Details*, which is our main focus in this research.

The *Temporal Details* pattern and its sub-patterns deal with the fact that humans cannot understand something complex instantly. Their understanding must evolve with time. In particular, aspects of initial understanding might need to be augmented or replaced. As a high level pattern, the temporal details pattern is broken down into the following sub-patterns: Snapshot, Long View, Multiple Approaches, Quick Start, Meaning, and Manipulate History¹. The following briefly explains what each pattern is about:

¹ Readers studying background literature will notice that the set of patterns evolved during its development. For example Thinking Big was removed as a Temporal Detail sub-pattern, and two other patterns were merged to form the Meaning pattern.

Snapshot: A snapshot is an instance of a representation² at a point in time during its evolution such that the most recent incremental changes to the representation is *conceptually complete* enough for people to discuss and understand it. The snapshot does not have to be an accurate or complete representation and it may contain inconsistencies. Snapshots can be seen during a time when someone is creating a diagram or model in a software engineering tool, or during an explanation someone presents on a whiteboard. The process of identifying snapshots is somewhat subjective, but in [1], Murray provides concrete guidelines for doing so, and also identifies a wide variety of types of snapshots. To illustrate the key concept of being conceptually complete: if the user added a class box in a UML diagram and then named the class, the snapshot would be considered to occur only after the class is named.

Long View: A Long View is a series of related snapshots; in other words, a set of representation-instances through a period of time as the representation is being developed to convey some concept. Showing the series of snapshots in a Long View is a way to tell a story visually. A user might use a Long View to explain a new aspect of a system.

Multiple Approaches: Sometimes a user has difficulty understanding a concept following a particular explanatory approach. A solution is to consider alternative approaches to gain more understanding. Moreover, there might be different valid alternatives to solve a particular problem.

Quick Start: People need simple starting places to begin understanding or exploring a system. They will often refer to something familiar and evolve their understanding from that point. Quick Starts can form the first snapshots in Long Views. For example, rather than explaining all aspects of a system's development, an explanation could start with a simple version that is well known.

Meaning: It is important for reviewers to understand the reasons behind design decisions or multiple approaches. The thoughts in the designer's mind are lost with time. It would be beneficial for the reviewer to be able to know what the designer was thinking and the reason behind his design. It is also important to capture the logic while moving on from one state of the system to another. It can also hold key information that explains

² The representations we will focus on are UML models, but the cognitive patterns have broader scope.

the changes made to a system. Meaning is essential in understanding how a system is built and how it evolved. The notion of temporal annotations, discussed earlier, is the most concrete manifestation used to explicitly record meaning, although the Meaning pattern covers the idea of implicit meaning too.

Manipulate History: This pattern builds on *Snapshot*, *Long View* and *Multiple Approaches* (those allow you to designate points, sequences and branches in the history of a model's evolution). *Manipulate History* allows you to adjust the history itself so you can revisit your understanding process.

1.4 Problem and main hypothesis

Software developers encounter difficulties when trying to understand or explain large software projects using current development tools. People have a difficult time understanding a complex artifact, such as a model or design, which has been developed over time.

The above problem can be broken down into several sub-problems:

- a) Humans are fundamentally unable to absorb a complex model when it is presented as a *single chunk*. Humans need assistance building a mental model of the model. The understanding process helps people to organize their mental model into chunks.
- b) People do not know what the *most important* aspects of a model are; in particular they have a hard time finding the parts of a complex model that they need to solve their own problem.
- c) People do not know the best place to *start understanding a model*. They do not automatically know a reasonable sequence to approach the understanding so that they can build on prior knowledge in a sensible way. They will therefore tend to start in an arbitrary place, and waste time understanding parts of a model which are not relevant to their needs, or which are not 'central' to the model.
- d) People are overwhelmed by the numbers of details present in a model and so become *frustrated*.
- e) People looking at a complete model tend to *miss important details* due to information overload.

- f) People are unaware of the *decisions and rationale* that led the model to be the way it is.
- g) Unawareness of aspects of a model leads to *incorrect decisions and repeated work* (such as re-analyzing the same issue someone has already analyzed).
- h) People are unaware of design alternatives that were considered but did not find their way into the final design – such lack of awareness can cause people to *choose a design alternative that should be rejected*.

To summarize: Software developers are not provided with enough features in their development environments that go side by side with cognition. This reduces the amount of understanding that developers are able to extract from software models therefore requiring more time to understand changes and design decisions.

We hypothesize that this problem could be solved to a limited extent by incorporating features based on the temporal details cognitive patterns.

1.5 Motivation

A prototype proposing a solution to the above problem could allow developers to understand software systems in a smaller amount of time, which would result in increased productivity. Such a feature may also improve understanding, resulting in better decisions, fewer defects, and higher quality.

The prototype could also lead to a commercial product delivered to customers.

The idea of basing tool features on cognitive patterns could influence the industry to base development of software features on scientific studies, and more specifically on studies of cognitive patterns.

1.6 Overview of the Temporal Model Explorer feature in the context of Cognitive Patterns

As discussed in Section 1.1, we created a feature in Rational Software Modeler that we call TME (Temporal Model Explorer). This feature records the complete history of development of a UML model at the level of the individual user-interface commands

the user employs (e.g. adds a class, renames a variable, or creates a relationship). The resulting record is a Long View.

The user can mark points in development history as Snapshots. People later trying to understand the model can use a scrollbar to slide each diagram “backwards and forwards in time”, and can jump from snapshot to snapshot. The set of snapshots can be edited at any time.

Finally, a user can create, edit and view temporal annotations, thus rendering the Meaning of changes explicit.

Incorporation of feature extensions related to Quick Start and Multiple Approaches is left to future work.

1.7 Key results

Participants expressed a very positive experience using our prototype. All the participants agreed that the TME prototype helped them understand class diagrams faster. Participants enjoyed the concept of snapshots and the majority wrote that temporal annotations are very useful when understanding models.

The majority of participants preferred a specific variant of our feature we call “final position.” In this variant, when viewing an earlier state of the system, the layout of the diagram appears with all classes in the positions to which they are eventually moved.

Participants agreed that the tool is user-friendly and that they would use it if it was available in their work environment if they were asked to understand a class diagram.

1.8 Outline of the remainder of the thesis

Chapter 2 includes a review of software development tools, with an analysis of their features and limitations, as well as how they support cognitive patterns. Chapter 3 outlines the procedure for choosing a new feature to prototype. Chapter 4 talks about the steps for building the prototype, and its functionality as well as the challenges faced during the process. Chapter 5 describes our evaluation strategy and presents the results of our user study. Finally we conclude this thesis in Chapter 6 by summarizing the work we did and the results that were achieved.

Chapter 2: Review of current software development tools

This chapter first introduces how development tool environments have evolved over time and outlines some of the remaining limitations in such environments. We will discuss current solutions and limitations illustrating these with examples from current software development environments including IBM Rational Software Architect 6.0 and Borland Together Architect 2006. We will relate current features to specific temporal details cognitive patterns.

2.1 Software development tools history

Software development tools and environments have advanced a lot starting with simple editors and compilers [23] to large-scale software visualization and development applications. With the advancement of computer hardware, software been able to progress in size and complexity to places never thought of before, with sizes of hundred of millions lines of code. Software exploration, search, analysis and visualization tools have become necessary, as have change management systems. New tools are often released, and studies of which tools are better have been performed [5]. Many tool evaluation frameworks have also been set up to help developers and designers create better tools.

Early environments were useful but they did not provide tools that were clearly integrated together [23]. It was the developer's job to connect the tools together: using pipes for example. The first tool integration efforts resulted in allowing a compiler to send the location of syntax errors to the editor which would handle the event [23]. Tools could register for events in other tools such that they would be notified when the registered events took place.

The main challenge in software development tools is still their integration [23]. While tools have advanced so much, in practise, their use has not advanced as much. The problem lies in the fact that the tools are still specific. They might force the user to write his program in a specific language or use a particular operating system. Some of the solutions to this challenge include the adoption of XML for saving and exchanging data

by a large number of commercial applications. Parsers have been developed to allow applications to read and save XML data easily.

Another important factor that is has often not been given enough attention in software applications is the problem of usability. While most developers know the basic graphical user interface guidelines, only a few of them are able to incorporate User-Centered Design in the software development lifecycle [20]. Developers should learn to appreciate a user-centered design approach and to evaluate the impact of choosing certain dialogue types and input/output devices on the user.

The above remarks were key motivators when building our functional prototype. We focused on the integration and usability factors: the prototype had to be well integrated and very easy to use. Our experiments in later stages confirmed that the participants found the prototype to be very user-friendly and they all agreed that they would use it if it was available to them.

2.2 Current solutions and limitations

We decided to explore the features of two modeling tools that are well known and well established in the software industry. The chosen tools were Rational Software Architect 6.0, which continues the series of the well known Rational Rose modeling products, and Borland Together Architect 2006.

IBM Rational Software Architect, RSA, is a software design and development tool that provides users with modeling capabilities and other features for creating well-architected applications and services [14]. Two powerful features of RSA are the browse and topic diagrams that allow users to explore a UML model based on a chosen central element from the model and looking through relationships of that element to the rest of the model. Filters can specify the depth and types of relationships to show. IBM Rational ClearCase, which is integrated with RSA, provides sophisticated version control [18]. Rational Software Modeler (RSM) [15] supports the same modeling features of RSA but lacks the enterprise features such as creating J2EE applications. Rational Systems Developer (RSD) supports modeling driven development for C/C++, Java 2 standard edition and CORBA based applications [16].

Borland released a new series of products in 2006 related to software modeling: Together Architect, Together Designer, and Together Developer [7]. Each tool provides specialized features related to the role of its intended user (software architect, designer, developer). However, they all provide the same modeling capabilities so we have chosen to evaluate Borland Together Architect 2006 (BTA) to learn more about the modeling features that Borland provides. The StarTeam product from Borland provides a complete range of change and configuration management solutions [8].

A variety of types of solutions have already been developed to address the problem described in the introduction (Section 1.4) – i.e. problem of people having a difficult time understanding a complex artifact, such as a model or design that has been developed over time.

The solutions can be broken down into several categories: physical division, temporal division, annotations, fine-grained change tracking, and persistent undo stacks. We will explain in the following the concepts in each category of solutions and the extent to which they solve the problem. We will also show screen shots and comment on how current products present features in certain solution categories. Additionally, we will relate the features to cognitive patterns.

2.2.1 Physical division

The most common known partial solution to the main problem we are addressing can best be described by the terms ‘divide and conquer’, ‘drilling down’ or ‘physical division’ of the artifact. A model is divided into multiple views or documents, typically arranged hierarchically. The understander starts by understanding a top-level overview that typically contains only a few details, and then ‘drills down’, expanding details as required.

Facilities for doing this kind of hierarchical exploration are found in a vast number of environments:

- Outline processors in a word processor allow you to see a table of contents to get an overview of a document, and then expand any section or subsection as needed
- Tools in modeling environments show a hierarchy of the artifacts available in a model

- ‘Grouping’ facilities in a spreadsheet allow you to hide and show groups of lines or columns. These can be nested.
- Facilities in a map viewer allow you to expand the types of details shown as you zoom in on a location.
- RSA browse diagrams allow you to browse a model by specifying a central object and the depth of the relationships from that object to the rest of the model. A user can increment the depth to learn incrementally about the model.
- EASEL [21] allows you to construct an architecture using several change sets (group of artifacts). Reviewers can apply or remove change sets to understand different features or versions of the represented system. Figure 1 shows EASEL’s user interface including the different layers (change sets) that the user can apply or remove.
- Physical division solutions relate to the Quick Start pattern discussed in Section 1.3.

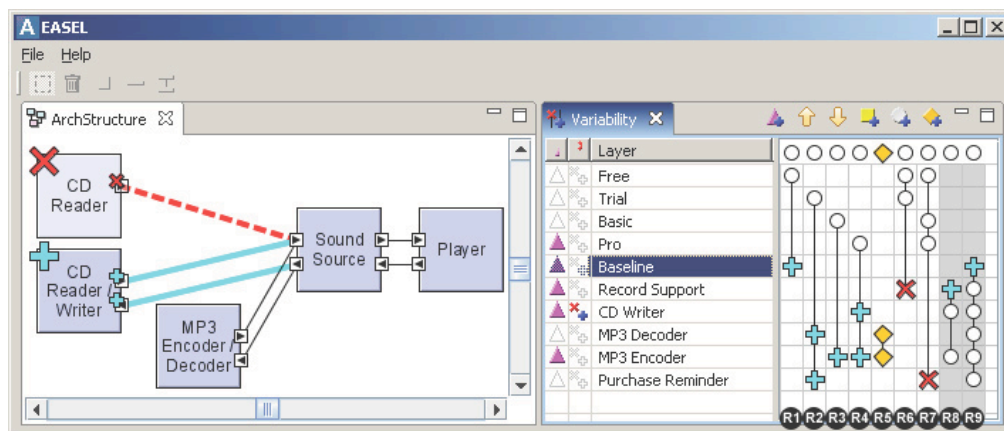


Figure 1 - EASEL change sets [11]

Extent to which the above solves the fundamental problem we are addressing

This first class of solutions, facilities for divide and conquer or drilling down, partially solve sub-problems a) to e) in Section 1.4, but they offer very limited assistance for sub-problems f) and h). In particular, the understander is always faced with understanding the model as it exists in its full final complexity.

2.2.2 Temporal division

The second major class of solutions is facilities that allow you to look at different versions of a model as they have existed at different points in time. For example, you can use a configuration management or change control tool (such as CVS, to be discussed in Section 2.2.3, or ClearCase [18]) to look at an earlier stage in a model’s development. Often the earlier stage is simpler and thus easier to understand. The understander can proceed by initially looking at the simpler model and then looking at subsequent versions one by one. This naturally solves sub-problem c) (in Section 1.4).

Temporal division solutions relate to the Snapshot and Longview patterns discussed in Section 1.3.

RSA and BTA support these solutions through the CVS features provided by Eclipse. The user has the option to use CVS repositories to maintain different versions of a system. The user is able to commit changes with comments that help understand the reason of the changes in the future. A table lists all the versions of a file including the time, date, author and comment related to the changes. The list of versions in the “CVS Resource History” can be considered as a Long View (series of Snapshots) as it shows the user the evolution of the system through each version. Figure 2 shows different versions of a file, each version is tagged with a date, author and comment.

Revisions of '/ModelSystem/Blank Model.emx'				
Revision	Tags	Date	Author	Comment
*1.3		5/9/05 4:32 PM	hfarah	added a Utility class
1.2		5/9/05 4:31 PM	hfarah	added generalizations and implementations relations
1.1		5/9/05 4:29 PM	hfarah	initial submission

Figure 2 - Eclipse history revisions view

The “CVS Annotate” feature allows the user to go through a file (text based) sequentially from the start until the end while seeing which part belongs to which version and the comments on that version. The number of lines and the author of the change are highlighted; the text inside the file is highlighted as well as the version number (as shown in Figure 3). The user can easily associate the highlighted areas together.

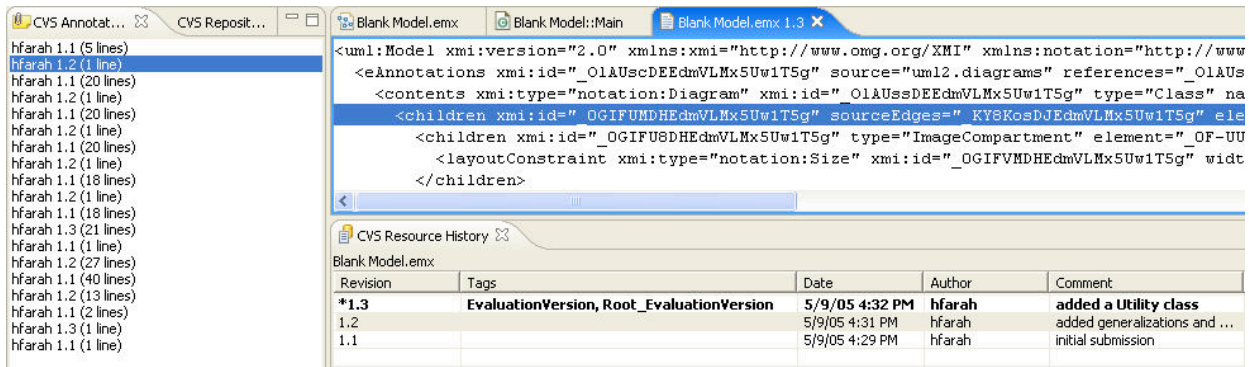


Figure 3 - Eclipse CVS annotations view

Some tools, such as Rational Software Modeler / Rational Software Architect have Compare-Merge facilities that allow you to see the difference between two versions to better understand the changes, and as a result, to better understand the overall model. RSA compare-merge functionality demonstrates the Snapshot pattern.

As discussed in Section 1.3, a snapshot is a view of a partial or entire system that can be discussed or contains relevant information.

RSA can show snapshots while comparing two versions of a system. The snapshots can be at different levels of granularity. The compare-merge feature automatically generates snapshots. Compare-merge produces snapshots at very low levels of granularity, and groups them in higher-level snapshots. The low-level snapshots are not meaningful from a user's perspective. For example, if we make an association between two classes, the snapshots shown are: 1) adding a reference in the source edges collection of the first class, 2) adding a reference in the target edge collection of the second class, 3) adding a reference in the edge collection of the diagram, and more, as shown in the tree figure. The higher-level snapshots groups all the snapshots related to the creation of the association. However, the user cannot have a customized-level of snapshots. The snapshots cannot be edited (added, merged or removed).

Snapshots could be part of a tree structure (shown in Figure 4) or visualized on side by side graphs (shown in Figures 5, 6, 7).

Tree:

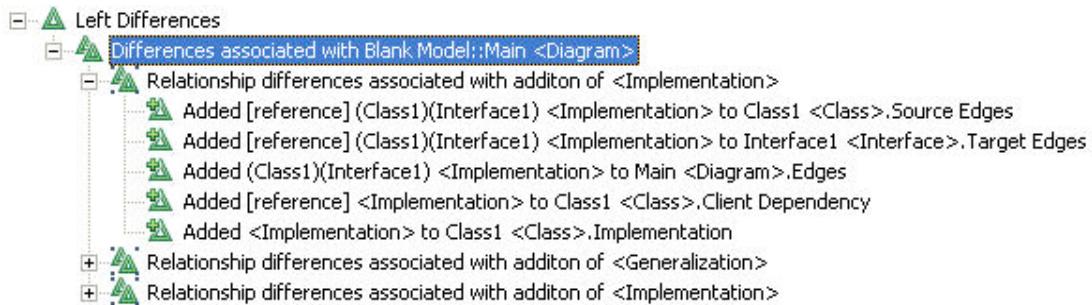


Figure 4 - RSA model compare tree view

At the higher level of granularity, only *Class1* and *Class2* would be highlighted since the added relationships concerned them most. But if we extend the tree node related to adding the implementation relationship between *Class1* and *Interface1*, we can visualize three different snapshots that highlight the process very well: *Class1* is highlighted (shown in Figure 5), *Interface1* is highlighted (shown in Figure 6), and the link is highlighted (shown in Figure 7).

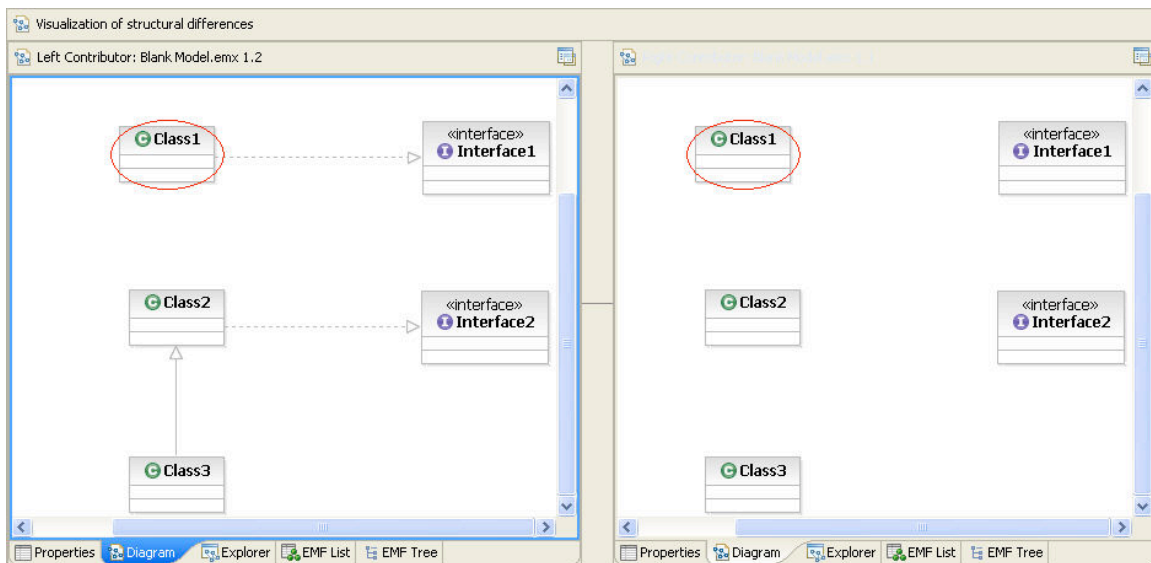


Figure 5 - RSA model compare visualization 1

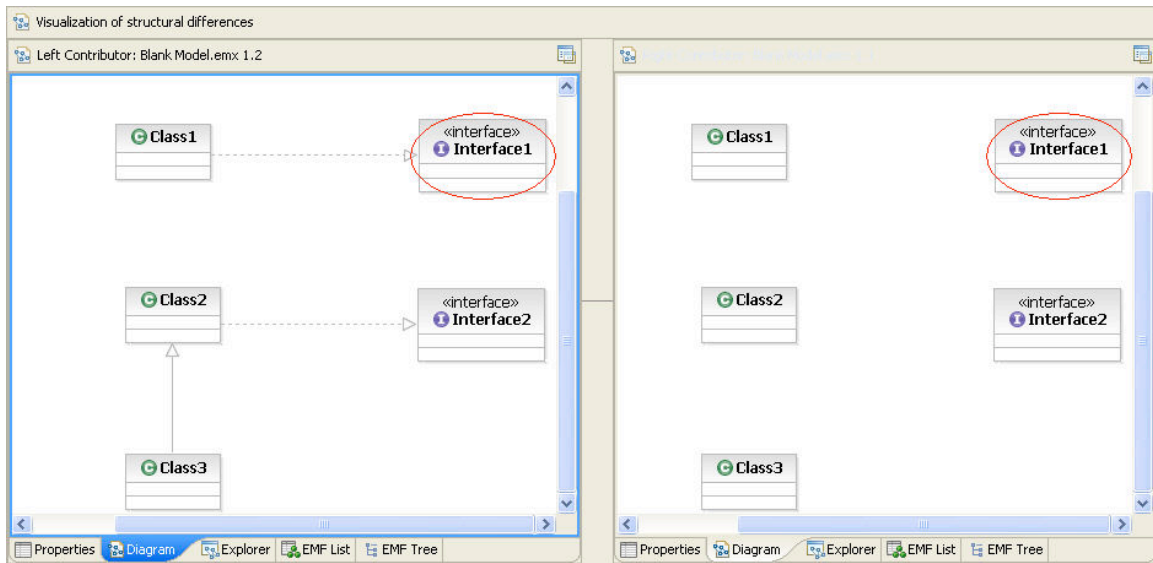


Figure 6 - RSA model compare visualization 2

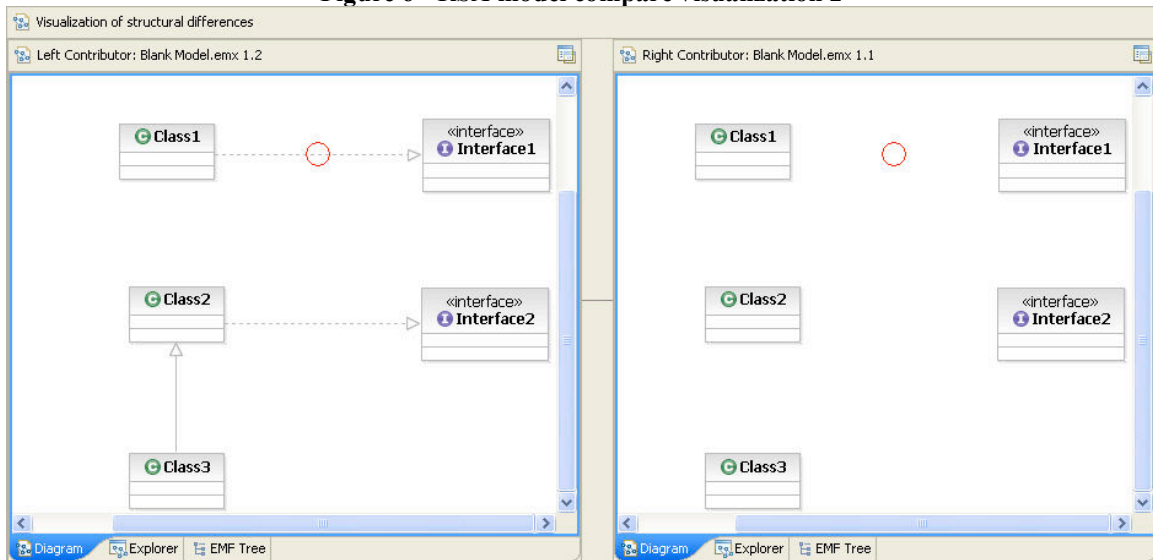


Figure 7 - RSA model compare visualization 3

The previous series of snapshots create a long-view (as discussed in Section 1.3, the Long View pattern is similar to telling a story) that can show the evolution of the system over time. While a snapshot consists of only 1 diagram, the long-view consists of successive diagrams that could be reviewed by clicking on consecutive items in the tree structure and visualizing the differences at each stage.

Our prototype builds on the general principle of temporal division, but does so in a novel and more effective way.

Extent to which the above solves the fundamental problem we are addressing

This second class of solutions, the ability to look at points in the history of a solution's development and compare such points, partially solves most sub-problems presented in Section 1.4.

The understander is able to see simpler versions of the model, and is also able to obtain some appreciation of the decision making process that went into the design, by observing the changes that were made. However, such solutions are somewhat awkward – the user has to explicitly load earlier versions and run compare-merge operations. Also the granularity of the deltas (differences between two versions) tends to be large (versions are normally saved only after a complete problem is solved) and unpredictable (people may do a large amount of work before saving a newer version).

2.2.3 Annotations, temporal annotations and design rationale documenting

The third class of solutions is facilities that allow you to add annotations. Annotations (often called 'comments' or 'notes') relate to the Meaning pattern discussed in Section 1.3. Such facilities are available in word processors, spreadsheets, CAD tools and software modeling tools. The modeler adds annotations to explain details that would not otherwise be obvious. Annotations can often help the understander make sense of some complex aspect of the model. However, UML notes should be added to a diagram in moderate numbers, since too many notes would complicate the diagram and hide its main design. Notes should be attached to existing elements only, if an element is deleted at a given stage in time, its note would not make much sense afterwards.

Annotations are also available in versioning systems (solution class 2.2.2 above). For example, when saving a version of an artifact in a tool like CVS, the saver will be prompted to document the reason for the change. (The reason might be *automatically* documented if the change is tied to a bug-tracking system). We call this type of annotation 'temporal annotations' since they document why something is being done at a particular point in time. Temporal annotations are particularly useful for helping people to understand the rationale for a particular change. In fact, there are tools explicitly designed to document the rationale for decisions.

Chapter 2 – Review of current software development tools

Hipikat [10] can save artifacts (change tasks, source file versions, messages posted on developer forums, and other project documents) during a project's development history. It can then recommend which artifacts are useful to complete a particular task. Depending on the type of artifact, it could contain design rationale or general information to help a developer better understand how to solve the task.

RSA and BTA support the following solutions related to annotations, allowing people to learn aspects of the rationale behind design decisions and alternatives.

UML diagrams support adding explanatory notes (shown in Figure 8) that give the user more information about the system (also available in BTA).

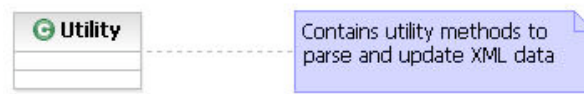


Figure 8 – RSA UML note attached to a class

Borland also presents additional features with its StarTeam product (RSA could support similar repository features using ClearCase [18]): we were required to set up the Borland StartTeam Server 2005 Release 2 [8] to enable the project sharing functionality.

Sharing a project using StartTeam gives the user more intuitive features allowing him to input more rationale when making changes as shown in Figure 9 below.

Change Request 82, Revision 1.1					
Synopsis	Description	Solution	Custom	Attachments	Comment
Status:	Priority:	Type:			
New	Yes	Defect			
Severity:	Platform:	Last build tested			
Medium	All	Web2			
External reference:			Addressed in build:		
Component:			By:		
User interface					
Category:			Responsibility		
UI			Isaac Newton		
Synopsis:			As of: 5/11/2006 1:		
Fix updating problem on close.			Entered by		
			Dawn Developer		
			On: 5/11/2006 1:43		

Figure 9 - StarTeam change request form, synopsis tab

The change request form allows the user to input all the details related to a change: status, priority, type, severity, platform, external references, component, category, synopsis, responsibility, description, solution, attachments, and comments.

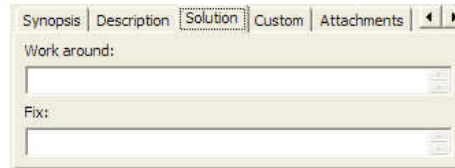
The image shows a screenshot of a software interface for a change request form. At the top, there are five tabs: 'Synopsis', 'Description', 'Solution', 'Custom', and 'Attachments'. The 'Solution' tab is currently selected and highlighted. Below the tabs, there are two text input fields. The first field is labeled 'Work around:' and the second is labeled 'Fix:'. Both fields have a small icon on the right side, likely for clearing or deleting the text.

Figure 10 - StarTeam change request form, solution tab

The solution tab shown in Figure 10 separates the types of solutions used to provide the user a better understanding of the context of the solution: work around or fix.

Extent to which the above solves the fundamental problem we are addressing

This class of solutions, annotations, and particularly temporal annotations, can work in conjunction with the other two classes of solutions to provide understanders with considerable guidance. However, the granularity of temporal annotations made in conjunction with a configuration management or version management system is dependent on the granularity with which versions are saved. An alternative, using a rationale-tracking tool [19] to explicitly document all design decisions is so cumbersome that such tools are rarely used in practice.

2.2.4 Fine-grained change tracking

A fourth approach is change tracking. In most word processors, and many other software tools, it is possible to track changes applied to a document by multiple authors. The understander therefore can glean information by looking at the types of information contributed by different people.

Extent to which the above solves the fundamental problem we are addressing

This approach does not solve the overall problem, but contributes to the solution to a limited degree.

2.2.5 Persistent undo stacks

Most software tools delete the stack of ‘undoable’ commands when the user quits

or saves a model. However, some tools have implemented *persistent* undo such that on reloading of a model, recent changes can be undone, perhaps all the way back to the beginning. This could be used by an understander trying to understand a model and forms the basis of a key aspect of this invention.

Extent to which the above solves the fundamental problem we are addressing

Persistent undo stacks have some potential to help with the understanding process, in that the understander could undo all changes and then replay them one by one. This has several drawbacks, including: 1) if a user undoes many changes and then starts editing, or saves the model, all subsequent undo/redo states would be lost; 2) the granularity of the undo stack is too fine; 3) persistent undo does not incorporate temporal annotations.

2.3 Support for other Temporal Details patterns

Rational Software Architect and Borland Together Architect also contain features related to temporal details patterns that were not discussed in the previous section. The analysis of these features helped us get better ideas for product new features as discussed in Section 3.

2.3.1 Rational Software Architect (RSA) 6.0

In the following, we will show how RSA to a certain extent supports the cognitive patterns, particularly Temporal Details. We will start by stating the pattern's name followed by explanations and screenshots from the tool:

Quick Start

The Quick Start pattern points out that people need a quick way to start a new task.

If we consider the task of building a class diagram: the user creates a new project, a blank model appears on the screen and the user has many alternatives to start building the system:

Chapter 2 – Review of current software development tools

1. If the user holds the cursor still for a few seconds, a simple menu appears with alternatives to start building the system including classes and interfaces (shown in Figure 11 below).



Figure 11 - RSA diagram elements popup menu

2. The user can choose from the items located on the palette (shown in Figure 12) by clicking on one item then clicking on the location in the diagram where he wants to place it.

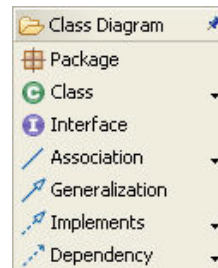


Figure 12 - RSA palette for class diagrams

3. Other alternatives include right-clicking on the blank diagram and selecting an option to add elements from the pop-up menu shown in Figure 13 below.

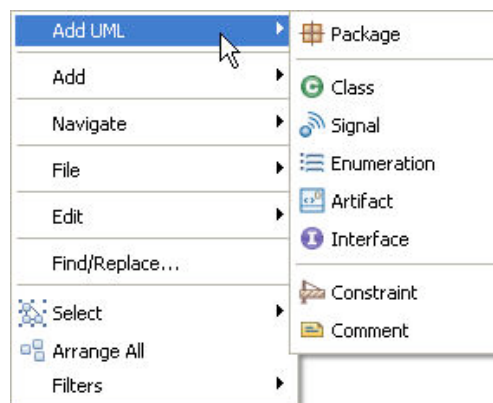


Figure 13 - RSA popup menu for class diagrams

If the user imported a project that already contained existing models, he has the option to drag and drop an element from the “Model Explorer” (shown in Figure 14) onto a diagram.

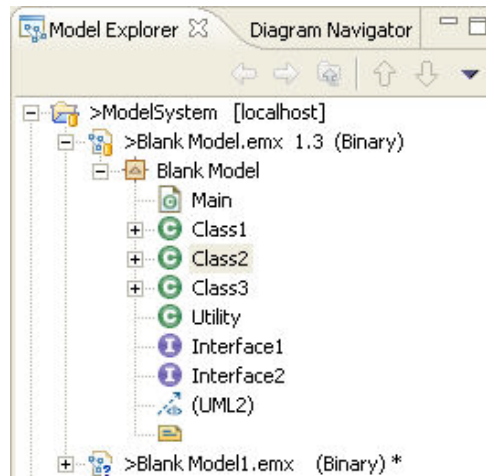


Figure 14 - RSA model explorer view

If the user is trying to understand a complex system, he can search a model for basic elements (*Baseline Landmarks* pattern) including a main method or a diagram to refer to his startup point. Double-clicking on the search results (shown in Figure 15 below) would open the diagram.



Figure 15 - RSA find and replace view

Multiple approaches

The following features are supported by Eclipse, hence they are available both in RSA and BTA.

CVS features allow a user multiple approaches for building the system: e.g. an evaluation version and a complete version.

- a. The CVS features offered in Eclipse allow the user to tag multiple file versions and to create different branches for files or projects. They can be accessed through the menu shown in Figure 16.

Chapter 2 – Review of current software development tools

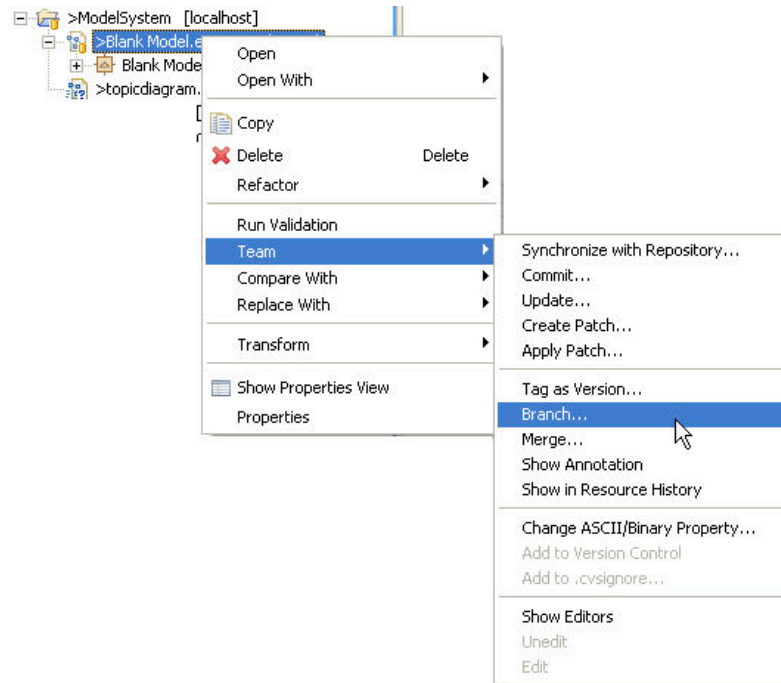


Figure 16 - Eclipse popup menu, team options

- b. To exclude some features from the evaluation version of his software product, a user creates another branch in CVS containing the files for the evaluation version which will not be affected by further updates to the files. The user will have the option to merge the branch with the other versions of the system in the future. The user can also switch between the development of multiple branches and versions as shown in Figure 17 or compare them in the “CVS Resource History” (shown in Figure 18) that also contains embedded rationale for the changes.

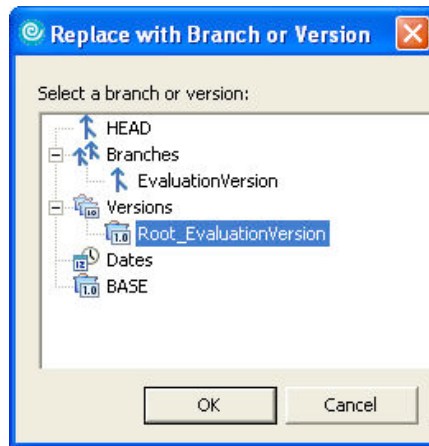


Figure 17 - Eclipse CVS options

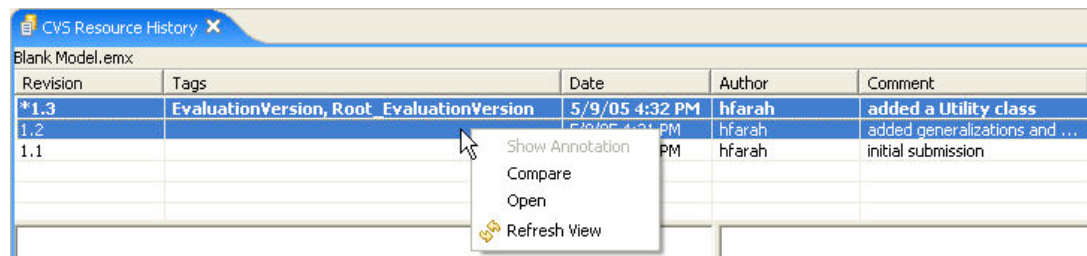


Figure 18 - Eclipse CVS Resource History view

2.3.2 Borland Together Architect (BTA) 2006 for Eclipse

The following are the features in Borland Together Architect 2006 related to cognitive patterns and particularly to the Temporal Details category. We will give each pattern's name followed by how it is demonstrated in BTA:

Quick Start

After creating a new modeling project, a blank diagram is displayed and the user has the following quick start alternatives to start building a design:

1. Click on an item to be selected from the palette (shown in Figure 19) then click on the diagram location for it to be placed:

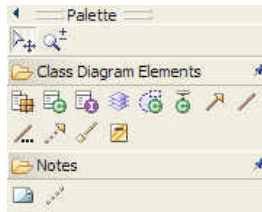


Figure 19 - BTA palette for class diagrams

2. Right-click on the empty diagram and choose an item from the context menu shown in Figure 20:

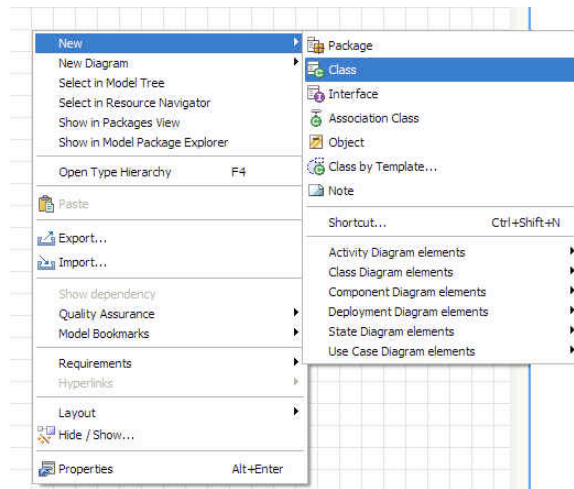


Figure 20 - BTA popup menu

3. If the user is working with an existing project, he could drag and drop existing model elements from the model navigator (shown in Figure 21) onto the diagram (given that they don't already exist in the diagram)

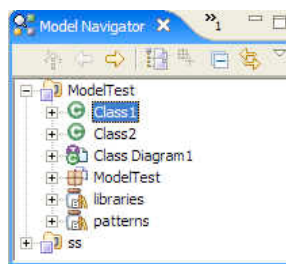


Figure 21 – BTA Model navigator view

Snapshot

A snapshot is a view of a partial or entire system that can be discussed or contains relevant information.

Chapter 2 – Review of current software development tools

The StartTeam environment allows a user to create and discuss a topic (as well as a change request, a requirement, or a task). Other users could reply to the topic forming a list of replies. Each topic or reply is a snapshot (related to the discussion, it's not a snapshot related to the artifact being discussed) since they contain a collection of information that can be discussed in a fair amount of details. Figure 22 shows a topic and multiple replies under it.

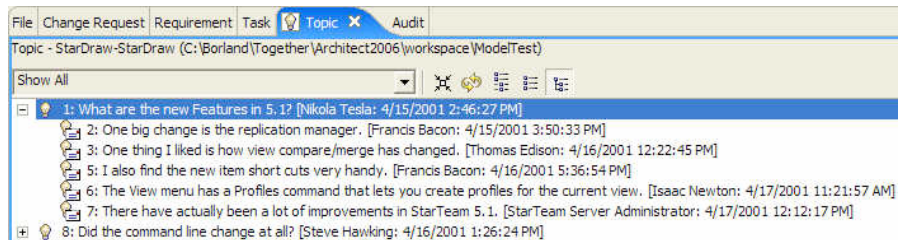


Figure 22 - StarTeam Topic view

Figure 23 shows information related to the selected topic in the Figure 22. This group of information is a snapshot.

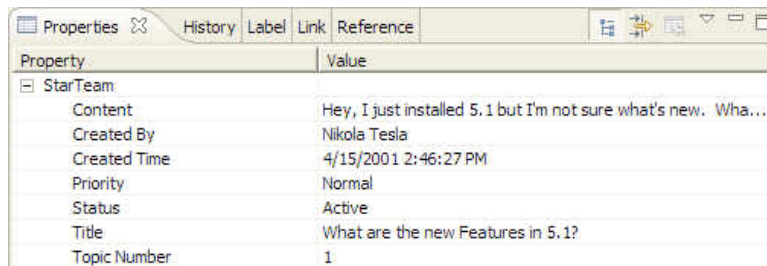


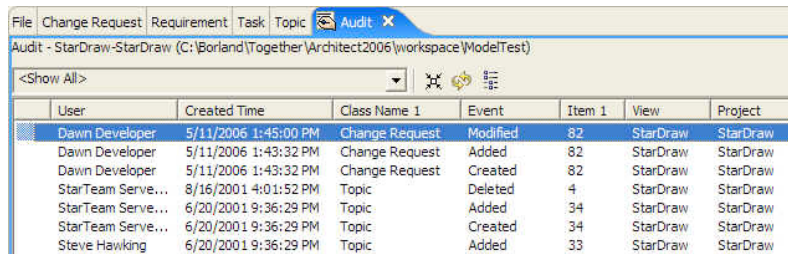
Figure 23 - StarTeam topic properties view

Long View

Figure 22 displays a series of snapshots forming a sub-tree having the first topic as the root. This sub-tree encapsulates the discussions on that topic from several users following a Long View pattern: a person would move from one snapshot to the other in order to fully understand the topic.

Another view featuring a Long View pattern is the audits view which displays all the changes being made to the shared project as well as additional information for each change as shown below in Figure 24.

Chapter 2 – Review of current software development tools



The screenshot shows the StarTeam Audit view interface. At the top, there is a menu bar with 'File', 'Change Request', 'Requirement', 'Task', 'Topic', and 'Audit'. Below the menu bar, the window title is 'Audit - StarDraw-StarDraw (C:\Borland\Together\Architect2006\workspace\ModelTest)'. There is a '<Show All>' dropdown menu and some icons. The main area contains a table with the following data:

User	Created Time	Class Name 1	Event	Item 1	View	Project
Dawn Developer	5/11/2006 1:45:00 PM	Change Request	Modified	82	StarDraw	StarDraw
Dawn Developer	5/11/2006 1:43:32 PM	Change Request	Added	82	StarDraw	StarDraw
Dawn Developer	5/11/2006 1:43:32 PM	Change Request	Created	82	StarDraw	StarDraw
StarTeam Serve...	8/16/2001 4:01:52 PM	Topic	Deleted	4	StarDraw	StarDraw
StarTeam Serve...	6/20/2001 9:36:29 PM	Topic	Added	34	StarDraw	StarDraw
StarTeam Serve...	6/20/2001 9:36:29 PM	Topic	Created	34	StarDraw	StarDraw
Steve Hawking	6/20/2001 9:36:29 PM	Topic	Added	33	StarDraw	StarDraw

Figure 24 - StarTeam Audit view

2.4 Concluding remarks

We learned that the previous tools support Temporal Details to a certain extent. We note that the level of granularity supported by CVS is not fine grained. Submissions to CVS cannot be changed: once a version has been submitted, it is persisted as a version. A user cannot delete versions from CVS or edit the comment on a submission. CVS branching is a powerful feature for managing multiple approaches.

Snapshots are supported but their level of granularity cannot be customized and they cannot be edited (added, merged, or removed).

In the next chapter, we will describe new tool features that address the above limitations.

Chapter 3: On generating new features

A major objective of the earlier research into cognitive patterns [1, 10, 12, 13] was that they should lead to ideas for better software engineering tool features – in particular, features that would better correspond to the way people think and work. This chapter describes the process we used to develop and refine a list of new features for RSx based on cognitive patterns.

The first step in our approach was to use existing UML tools extensively, in particular Rational Software Architect 6.0 and Borland Together Architect 2006, thinking about the features they provide from a cognitive patterns perspective. In addition, we attended one usability study for RSA in order to gain more information about how users use it and try to generate a list of new ideas for improving their experience. The study was performed by the user-centered design team at the IBM Ottawa Lab. We watched an experienced RSA user participate in the study. The study took place in two separate rooms in order not to bother the user. We monitored the study from the other room using NetMeeting to see everything the user did on the screen and by telephone to listen to the conversation.

In the next section we will describe a list of potential prototype features that we generated. In Section 3.2, we discuss how we evaluated those features in order to choose the most appropriate idea to prototype. Section 3.3 presents the three main features we generated and the chosen feature to prototype.

3.1 Initial list of features

We generated a list of potential new features while using Rational Software Architect, Borland Together Architect and reviewing the papers about cognitive patterns [1, 10, 13]. Our ideas were based on incremental learning, understanding how the system works, and learning why the system was designed the way it has been.

Our focus was on features for working with class diagrams because they tend to be complex, they are the most widely used UML diagram, and they capture important aspects of the internal design of the system. The user needs to spend time to analyze class diagrams and understand how the system works.

Please note that the ‘*Thinking Big*’ pattern (i.e. a user needs to see the bigger picture in order to better understand how a part fits in the system) was removed from the Temporal Details category at a later stage in this research. However, we have decided to keep the feature related to this pattern in this list since it was evaluated at that time as being part of the Temporal Details group.

The following outlines each of the features that were generated.

3.1.1. Details equalizer

Currently in RSA: filtering is applied to one object in the diagram at a time. RSA “Browse” and “Topic” diagrams ignore the filtering specified in the original diagram. This feature would filter attributes and operations from all the objects in any diagram based on a defined level of detail (public, protected, private, package). This approach would enhance incremental learning by showing the big picture then giving more details. The user would use a series of sliders (visualized in Figure 25) to quickly set the level of details on different criteria. (Relates to: *Quick Start*)

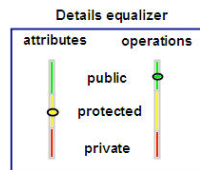


Figure 25 - Details equalizer sketch

3.1.2. Elements filter

RSA supports other types of filtering including relationship-based filtering (generalization, dependency, realization...) that is very well implemented as part of the ‘Browse’ or ‘Topic’ visualization diagrams. Using the Elements filter feature, a user would be able to choose to see elements in the diagram according to specific criteria, before looking at the full diagram. The user would be able to quickly understand some concepts and have a base for further understanding. The criteria could include elements that have notes attached to them, or the filter could be based on the most central elements (e.g. most attributes, associations, inheritance). (Relates to: *Quick Start*)

3.1.3. Diagram slider

In this feature, a user could view the entire history of a diagram's creation by using a slider control that would evolve the diagram from its initial state to its final state (similar to telling a story). This was part of the Temporal Model Explorer (TME) feature eventually chosen to implement. (Relates to: *Quick Start, Long View*)

3.1.4. Diagram version slider

This feature builds up on the previous one. Instead of having a very detailed slider, the user could specify at which points to tag a version of the diagram. Future users would review all the previous versions leading to the final diagram (an alternative would be to have multiple tabs allowing the user to randomly jump between versions). Aspects of this, i.e. the use of snapshots, are also found in the TME feature. (Relates to: *Quick Start, Long View, Snapshot*)

3.1.5. Package outline

The icon of a Java package should have the ability to show the classes and interfaces or other resources that belongs to it. This would show the big picture of the system to the user and might reduce search time by allowing increased visibility in the details of the system. However, if a package contains too many elements, an icon view would show limited details because of the space limitations. (Relates to: *Thinking Big*)

3.1.6. Easy access to versions of a diagram

Currently in RSA, if a user wants to see different versions of a diagram, he has to open the “CVS Resource History” view and double click on each version which will open the model followed by the diagram. By the time three different versions are displayed, the editor tabs are overwhelmed and it becomes confusing to switch between versions.

It would be much simpler if the diagram tab had ‘sub-tabs’ with version numbers, that would save adding all the unnecessary tabs to the main editor tabs. A drop-down menu could do the job as well. Right-clicking on the diagram and choosing which version to see is yet a third alternative to solve this problem. (Relates to: *Multiple Approaches*)

3.1.7. Search for feature

A user should have the option to view all areas related to a feature. This requires defining what parts of the software relate to the feature in the first place. This could save time when searching for textual matches or references that relate to the feature. An example of using this feature would be to search for all the places in the code or diagrams that relate to ‘printing’. (Relates to: *Quick Start*)

3.1.8. Explanation diagrams

This feature would provide a new type of diagram that would contain explanations and links to diagrams or code. This feature could guide the user where to start in understanding a problem. It lets the user go through the diagram step by step (e.g. tabs) or view it all (e.g. tree) to understand a concept while looking at other diagrams at the same time. The creator of a model could mark starting points for understanding each diagram.

One can argue that the user can achieve this with any diagram including notes but the advantage here would be the tree or tab view that allows the user to see all the steps in a compact way and go through them. (Relates to: *Quick Start, Meaning*)

3.1.9. Features list

This feature would list the features in the system with links to the diagrams where they are implemented.

A new user has no idea of a system he never saw before, he needs a starting point. Instead of looking at various diagrams and trying to figure out what they are for, a view could show a features list that contains multiple main features of the system with links to related diagrams that could serve as a starting point for understanding the feature or the system.

Each feature could include a sequence of diagrams to visit with some meaning of each diagram to the feature. (Relates to: *Quick Start, Meaning*).

3.1.10. Alternative diagrams display

Currently in RSA, if a user wanted to develop an alternative approach for a design shown in a particular diagram, he needs to create a new diagram, build the other approach into the new diagram and insert a link in the original diagram with a note informing the user of an alternative design.

We're proposing to have an icon in the original diagram which would represent the availability of an alternative design. When the user clicks on that icon, the window containing the original diagram would be divided in half and the two approaches would be displayed side-by-side in the split window. This will allow the user to examine the differences between the two approaches more effectively. (Relates to: *Multiple Approaches*)

3.1.11. Part swapping

Part swapping involves switching between design alternatives inside the same diagram. A user could select parts of a diagram and specify alternative files containing alternative design for the selected parts. An icon would specify that an alternative design exists and with a button click on that icon, the user would swap design alternatives in the diagram. (Relates to: *Multiple Approaches*)

3.1.12. Annotations

Annotations would highlight the reasons behind changes. Annotations could be included in the “Compare with local history” feature within Eclipse. This existing Eclipse feature saves a copy of the file that the user is working with each time the user saves the file in order to track versions of the file over time. Our feature would involve adding optional annotations to each point in time where a file version exists to explain the reason for a change.

Interesting questions come up: At what level could the user add these (perhaps after saving the file, without forcing him)? Another option is that a new shortcut key could pop-up an input dialog to enter a comment for the last saved file. This could potentially help developers in resolving bugs more efficiently by understanding changes quickly. This feature was adopted into TME. (Relates to: *Meaning, Snapshot*)

3.1.13. Version player

This feature would show the user different versions of a file or diagram in sequence. This could help the user understand how the file contents evolved.

Currently, the user could open all the versions of a diagram but it would result in too many editors and can become confusing. It would be useful if the user could see a sequence of diagrams, each one lasting for a specific time interval then move to the next. This could be done by selecting many versions in the “CVS Resource History” view and having a ‘play’ option. The version currently showing will be highlighted in the view so that the user can see the highlighted comment (entered when the user submits an update to the CVS repository). Aspects of this appeared eventually in TME. (Relates to: *Snapshot, Long View, Multiple Approaches*)

3.1.14. Diagram compare

When comparing diagrams in the RSA compare-merge view, highlight all the changes at the same time (currently not working), and have the option to see the change description (e.g. “added generalization class1-class2”) right on the diagram instead of having it as a separate tree (Figure 4), this would save the user having to click the item each time and switch to the diagram view. (Relates to: *Snapshot, Meaning*)

3.1.15. Save compare

Add the capability to save a compare-merge view (Figures 5, 6, 7) as one diagram to which the user can add notes explaining some changes. Currently, the user can compare two versions of a diagram side by side with highlighted changes but he cannot save that comparison. (Relates to: *Snapshot, Multiple Approaches*)

3.2 Evaluating and grouping the features

We held meetings with five senior IBM product managers and developers to evaluate the value of each of the ideas in our list. Product managers were closely linked to customers and they took in consideration the importance of a feature to the customers when evaluating its value. We did not question customers directly for confidentiality

purposes, this evaluation was restricted to IBM but in a way that ensured taking into consideration what the users of this tool desired.

3.2.1 Participants feedback

We interviewed participants with different qualifications: software developers, team managers, product managers and members from the IBM Centers for Advanced Studies (CAS) in Ottawa. We wanted to gather feedback from different points of views and capture a general perspective of what feature is mostly desired.

Participant 1

IBM Software Group, Rational

Model Driven Development Product Management

The discussion was brief, about 20 minutes. The participant knew RSA very well and quickly judged if a feature was beneficial for the overall product or not. He was able to point out the developer who would be the appropriate contact for every feature. I was able to hold meetings with these contacts to obtain better feedback on all the features.

Participant 2

IBM Software Group, Rational

IBM Ottawa Center for Advanced Studies

This participant gave us feedback on all the features, the meeting lasted about 40 minutes. His perspective focused on how to make the tool communicate the design between developers effectively. His judgment was also affected by how often or how likely the users are to use the feature. He gave us the following comments and suggestions:

- Add the ability to mark key objects (e.g. objects where the user would start understanding) for the element filtering feature (3.1.2).
- The diagram slider (3.1.3) would probably help the user see how the system evolved rather than to understand it.

- The search for a feature (3.1.7) would not be very effective because people will not trust it since some areas might be forgotten to be marked.
- Add a diagram player feature that records the order in which objects were created.
- Finally, he had doubts that the save compare annotations (3.1.14) would be used frequently.

Participant 3

IBM Software Group, Rational

Rational Modeling Platform Lead

The meeting lasted for about one hour. We discussed in depth the features related to understanding change in the compare-merge area. Prototyping opportunities were clear in that area. The following were some of his comments on the proposed features:

- The diagram player feature (3.1.3) could encounter limitations regarding maintaining the size of the history files but it could also be very instructive.
- The diagram version slider (3.1.4) could be improved by allowing the user to designate diagram snapshots (this was implemented in our prototype) and assigning them specific names.
- The search for feature (3.1.7) would be a great idea if users use it well: this required a user to link what he is currently working on to a feature of the system. If the user forgot to make a link to a feature any time he worked on something new, the search for feature would not return complete results.
- He suggested that a feature other than explanation diagrams (3.1.8) could be the use of hyperlinks (similar to the TODO concept: in Eclipse, while writing source code, a developer can add a comment using `//TODO`. This will automatically place his comment in a separate Eclipse view called ‘Tasks’. This feature helps the developer track all his remaining tasks in all the files in his development environments.).
- The feature list (3.1.9) was an improved way of reverse engineering the component architecture. It could be useful in cross model reference.

- Alternative diagrams display could be useful in the markup and review: The user could look at different alternatives and choose the main one.
- The diagram compare feature (3.1.14) could have one model (instead of two side-by-side displays) with add/change/delete annotations. When a user hovers over an annotation with the cursor, a tool tip could be displayed containing more information about the change.
- Saving the compare merge view was a feature taken into consideration by the team for future development; it takes a lot of time.

Participant 4

IBM Software Group, Rational

Senior Software Developer, Architect

This participant was related to the field of semantic concern: how to distinguish (e.g. using different colors) parts of the diagram based on some concerns. He was particularly interested in the first two filtering features (3.1.1 and 3.1.2) for browse and topic diagrams and suggested to merge them into one feature. He mentioned that the diagram player (3.1.3) is more of a demo feature and that it would have more value if the user could explain or add explanations later to playback (this was implemented in our prototype).

Participant 5

IBM Software Group, Rational

Aurora Shapes Management

This participant is involved in the field of visualization and diagram representation and filtering. He had the following comments:

- The diagram equalizer (3.1.1) feature would have more value if it was more general (not limited to browse and topic diagrams).

- Modify to the second suggestion (3.1.2) by linking it to a query. The challenge would be the way it is presented to the user. However, something was already being done in that area.
- The diagram slider (3.1.3) was cool but he questioned how useful it would be.
- The search for feature (3.1.7) has something similar already available: Rational RequisitePro is a requirements management tool that facilitates the communication of project goals between a group of people. It provides detailed traceability to show how requirements may be affected by changes [17].
- For part swapping (3.1.11), how would we handle printing hidden sections?
- The model annotation (3.1.12) was very interesting, an idea would be commenting as well (MS Word example: users could choose text and include a comment that would appear on the side of the screen for other users to read and possibly reply to).
- The diagram player (3.1.13) would have scalability issues.
- An alternative to the save compare screen (3.1.15) feature: a user could take a snapshot by a third-party program, annotate it then save it as an image.

3.2.2 Feature scores

Table 1 shows the features and their evaluation scores given by the different participants indicated as P1, P2, P3, P4, and P5. We asked each participant to rank each feature between 1 and 3: 1 meant that the feature was needed and 3 meant that this feature was not required.

ID	Feature	Weight					
		<i>P1</i>	<i>P2</i>	<i>P3</i>	<i>P4</i>	<i>P5</i>	<i>Avg</i>
1	Details equalizer	1	1	1	1	2	1.2
2	Elements filter	2	3	2	1	3	2.2
3	Diagram slider	1	2	2	2	3	2
4	Diagram version slider	2	2	2	2	2	2
5	Package outline	3	2	3	3	2	2.6
6	Easy access to versions	3	2	1	2	2	2
7	Search for feature	1	3	1	1	1	1.4
8	Explanation diagram	2	2	2	2	2	2
9	Feature list	3	2	1	2	1	1.8
10	Alternative diagrams display	3	2	2	3	3	2.6
11	Part swapping	3	2	3	3	2	2.6
12	Annotations	2	3	1	2	1	1.8
13	Version player	2	3	3	2	3	2.6
14	Diagram compare	3	2	1	2	2	2
15	Save compare	1	3	1	3	3	2.2

Table 1 - Prototype features' weights

Participants were asked about the features in the same order but each time they were asked to rank each feature independently from the others. This was necessary because each team manager might rank the features depending on whether they were related to his work or not, and we wanted to get feedback independent of that to gain a better idea of the overall importance of each feature.

The package outline (3.1.5), alternatives diagram displays (3.1.11), part swapping (3.1.12), and version player (3.3.13) features averaged a score of 2.6. They were considered as possible future enhancements. There was a lack of interest in implementing such features and they were removed from our list.

A grouping strategy that we used was to determine if the feature was related to static incremental learning, system evolution, or could be an enhancement.

The following are the codes used in Table 2:

Chapter 3 – On generating new features

[inc] Allows incremental understanding on static system by managing views

[dev] Shows system evolution dynamically

[enh] Facilitates and enhances existing usability/new feature

ID	Feature
[inc]	Details equalizer
[inc]	Elements filter
[dev]	Diagram slider
[dev]	Diagram version slider
[enh]	Easy access to versions
[inc]	Search for feature
[inc]	Explanation diagram
[inc]	Feature list
[enh]	Annotations
[enh]	Diagram compare
[enh]	Save compare

Table 2 - Prototype features' categorization

We removed most of the features related to enhancements and merged similar features in order to group them in one common feature. Table 3 shows how the remaining features were grouped:

Feature
[inc] Details equalizer [inc] Elements filter
[dev] Diagram slider [dev] Diagram version slider [enh] Annotations
[inc] Search for feature [inc] Explanation diagram [inc] Feature list

Table 3 - Prototype features' grouping to create three new main features

Finally, we gave names to the three larger ‘main’ features resulting from this grouping: Diagram equalizer, Diagram player and Diagram guide. Table 4 shows these three features with the highest and lowest rankings that they received (highest and lowest rankings relate to the combined rankings of the features that were merged):

ID	Feature	Weight (max/min)					
		<i>P1</i>	<i>P2</i>	<i>P3</i>	<i>P4</i>	<i>P5</i>	<i>Avg</i>
A)	Diagram equalizer	1/2	1/3	1/2	1/1	2/3	1.2/2.2
B)	Diagram player	1/2	1/2	1/2	2/2	1/3	1.2/2.2
C)	Diagram guide	1/3	2/3	1/2	1/2	1/2	1.2/2.4

Table 4 - Main prototype features evaluation

3.3 Descriptions and analysis of the three main features

The three main features are described in the following:

3.3.1 Diagram equalizer

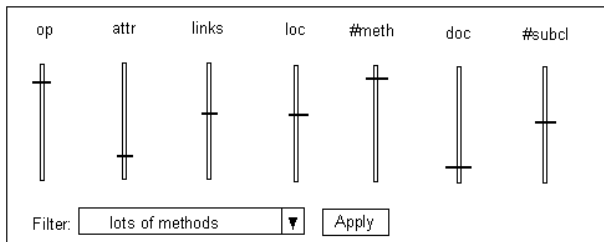


Figure 26 - Diagram equalizer sketch

The diagram equalizer shown in Figure 26 uses several sliders to control the level of details in a diagram. Each combination of slider positions applied by the user can be saved for reuse.

It could help the user by reducing the amount of details in the diagram and so reducing the size of the diagram making initial understanding easier. It also removes uninteresting details from the diagram and leaves the user with only what he wants to see making understanding easier by only having related material that flows with the user's thought.

One of the problems to be addressed is the diagram layout after filtering is applied: The “Arrange all” option in RSA can sometimes produce unpredictable results which are not optimal, we shouldn't rearrange the diagram, and we should rather shrink the size of the boxes and shorten the length of the connectors (compacting the diagram as needed).

We used the following three criteria to analyze the feasibility of implementing this feature:

1. *Possible to implement*
 - a. There is something already under development for next version of RSA that can change the level of details (attributes and operations) for a group of elements.
2. *How important is it for the teams*
 - a. Very important to have such filtering capabilities
3. *How relative is it to temporal details*

- a. The diagram representation is still static but the different views obtained after applying different filtering patterns could allow the user to learn incrementally about the diagram.

Currently in RSA, relationship filtering can be applied for the entire diagram. Details filtering can be applied for one element at a time but there is work already being done to apply filtering to multiple items at once

3.3.2 Diagram player

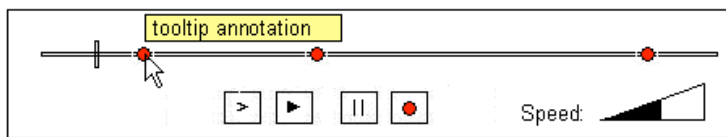


Figure 27 - Diagram player sketch

The diagram shown in Figure 27 player plays back the construction of the diagram and allows tagging playback locations as snapshots with annotations. The playback could possibly keep elements fixed at their final position in the diagram. New additions and deletions in the diagram could be highlighted to help the user visualize the changes better.

It could help the user by replacing the final static representation of the diagram with a dynamic approach. We hypothesize that seeing how the diagram was constructed can help the user better understand the diagram (e.g. the most important element might have been added first to the diagram).

One of the problems to be addressed is scrolling a diagram not fitting on the same page.

The following were the results of our three analysis questions for this feature:

1. *Possible to implement*
 - a. Depends on how hard it is to capture, save, and replay the undo stack (we implemented this feature for our prototype)
2. *How important is it for the teams*
 - a. It is important from product management perspective
3. *How relative is it to temporal details*

- a. Directly related to Temporal Details as it shows how the diagram was built from start to finish with annotation capability to add embedded rationale.

Currently in RSA, the history maintains a limited number of versions of the diagram each time it is saved. We cannot add annotations to versions in the history. We cannot play sequential changes. We have to open each saved diagram separately.

3.3.3 Diagram guide

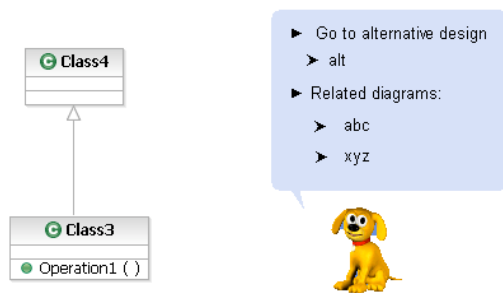


Figure 28 - Diagram guide sketch

When the user is looking at a diagram, the diagram guide shown in Figure 28 would tell him of related diagrams to look at and for what purposes (e.g. alternative designs), it allows the user to go to the diagrams with a single click. It could also list or open diagrams related to a feature (e.g. printing). It could as well indicate the relationship of the diagram to the feature (e.g. controls access to the printer).

This feature would help the user by reducing the search pain and the time analyzing search results to find a logical link between diagrams containing the implementation of a feature.

One of the problems to be addressed is how to teach the guide initially and relate diagrams to features or between each other.

Answers to the three analysis questions are as follows:

1. *Possible to implement*

- a. Might be easier as a view with dynamic behavior instead of a character

2. *How important is it for the teams*

- a. Relatively important, however less than the two previous ideas.

3. *How relative is it to temporal details*

- a. Users working on several diagrams understand how they relate. Being able to communicate relationships between diagrams to others might bring a lot of help in understanding big systems. This information is usually lost.

Currently in RSA, we can put links to other diagrams and notes explaining how they relate, however, they are part of the diagram itself and not separate.

3.3.4 Chosen feature

The diagram filtering feature was encouraged by everyone. It was a main concern that needed to be addressed. However, it is not very close to the Temporal Details category of the cognitive patterns. It can be argued that this feature allows you to understand better over time by limiting the amount of details shown in the diagram. A user could start by applying a filter that shows only the high level components without their internal details. The level of details can then be augmented step by step allowing incremental understanding of the system. But there is a difference between understanding incrementally over time and showing the evolution of the system over time.

The average features scores were similar with a small decrease for the diagram guide feature. However, the diagram player had the least current tool support and relates very well to Temporal Details. The diagram player had four 1s and one 3 in its evaluation while the other options had four 1s and two 3s.

The diagram player (later named Temporal Model Explorer) was chosen to be prototyped. The next steps discuss how we built the prototype in an iterative process, and how we evaluated our hypothesis with user studies.

Chapter 4: Building the TME prototype

In this chapter, we begin by giving a general overview of our prototype. Following the brief description, we explain how we developed an initial proof of concept before deciding to implement a functional prototype. In order to explain how we built our functional prototype, we will discuss several steps including: our decision to adopt RSx as our target application, the challenges in setting up our development environment, the iterations, functionality and challenges we encountered. We will also discuss our design alternatives, core ideas behind our prototype, its architecture and integration within the target application.

4.1 Prototype description

The TME prototype is a plug-in for Rational Software Architect/Modeler 7.0 enabling users to record changes on a UML model and its diagrams and to replay those changes at any time using features such as temporal annotations and snapshots.

Temporal annotation support enables the user to add a note after making any change to the model or a diagram. This annotation will be shown to any user later viewing the diagram's historical state corresponding to the time that change was applied.

Snapshot marking enables the user to group a set of changes that will be applied together providing a higher level of granularity for moving between changes.

Recording is done automatically once the TME plug-in is loaded (except if the user disabled the TME functionality from its preference page). The changes are recorded at the EMF level allowing us to record changes on different kinds of diagrams.

While reviewing the history, a user can go through the changes step by step, or jump from snapshot to snapshot. The user can use the keyboard or a slider control to go through the history.

For the purpose of this research, we have studied the use of the tool with class diagrams only, given the time limitations and the hypotheses we wanted to validate.

4.2 Prototyping with informal tools

We developed the early iterations of the TME feature using an informal tool, since studies have shown that informal tools are better than both pencil and paper as well as high fidelity tools for building early prototypes [9].

Low fidelity tools or pencil and paper are very easy to use: they don't force the prototyper to include unnecessary details. The prototyper has the freedom to represent his ideas as he likes, and this tends to encourage him to propose more ideas. However, the downside of such approaches is that they limit the prototyper's ability to communicate behavioural design ideas (temporal and interactive parts of the application).

High-fidelity tools, on the other hand, allow the user to express and show the application's full behaviour. However, they require the user to spend much more time and effort as compared to using low-fidelity tools.

In order to obtain the advantages of both pencil and paper and high-fidelity tools, an informal design tool tries to transform a designer's early design representations into a functional prototype. It can allow the user to edit the representations, include annotations to capture the rationale behind the design, and facilitate group work.

We used MS PowerPoint and screen capturing tools to design our early prototype of the TME feature. This allowed us to demonstrate its projected functionality and get a good feel of the user's response if he was presented with a functional prototype. We received positive feedback about our proposed feature and started the development of a functional prototype.

4.3 Initial prototype

We performed an initial study with a colleague at the IBM Center for Advanced Studies in Ottawa. The study involved asking her to model a given system using a class diagram. Afterwards, the system description was modified and she was asked to modify her class diagram to reflect the changes. A screen snapshot was taken after each of her modifications using a screen capturing tool. After the modeling was completed, the person was allowed to add annotations to screen snapshots she selected as being relevant.

The following is the initial system description that the user was asked to model: "A TIVO-type recording device that can be programmed to record TV shows from

various channels at various times. It can also record any show of a certain type or a certain series. It downloads the schedule so it knows what is playing. It has to manage the total amount of storage available and erases shows that the user did not explicitly ask to have recorded.” [22]

After the participant completed modeling the system, the requirements were modified as follows: the recording device has to contact a database to authenticate storage, and storage is not local anymore, it is on the network and needs to be accessed through a particular storing device. Commercials can be recorded as well as shows but a user can decide to view a show with or without commercials. The recording device has playback functionality now to play shows or commercials.

While the participant was modifying the system, screen snapshots were also captured. After the participant completed modifying the system, we showed her the snapshots and asked where she felt that she could add annotations to improve the user’s understanding of the evolving system. The participant decided to add the following two annotations: the first was that the *playCommercial()* method was removed from the TIVO class because of the re-interpretation of the requirements (commercials cannot be played individually) and the second was that the association between the storage and the authentication database was replaced by an association between the authentication database and the network access device to reduce unnecessary access to the storage. Figure 29 is an example of the snapshots being captured (after the annotation was added):

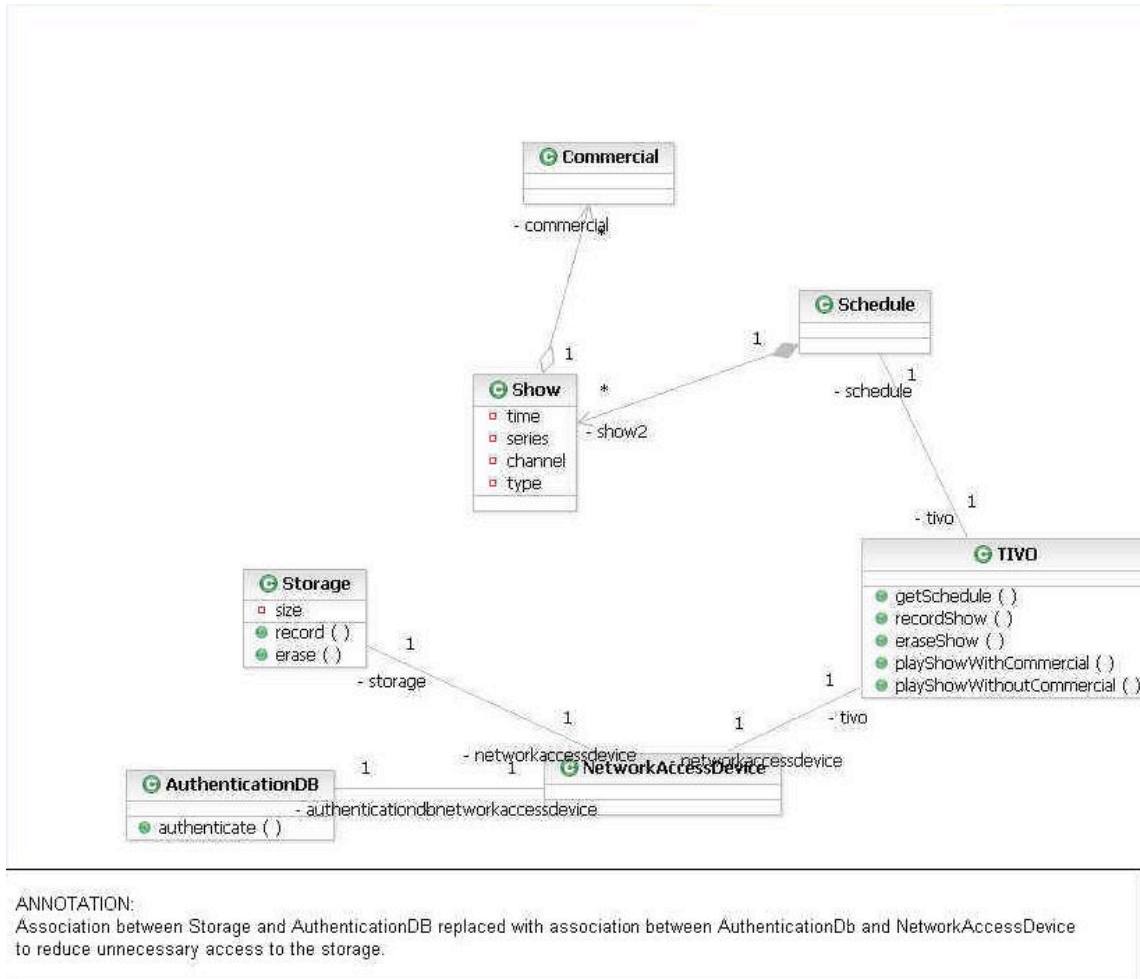


Figure 29 - Initial prototype, model screenshot with annotation

Many interesting questions were discussed after we captured the first set of snapshots: Do we play back the changes as they were performed by the user? Should we ignore changes related to movements and keep elements in their final positions (less distraction)? Should we highlight new changes or annotate them (making them easily identifiable)? We created a set of snapshots with elements showing always in their final position in the diagram and discussed two highlighting techniques: the first would be to mark new changes with a circle as shown in Figure 30, the second would be to annotate additions with a '+' sign and items that were to be deleted in the next step with a '-' sign as shown in Figure 31.

Chapter 4 – Building the TME prototype

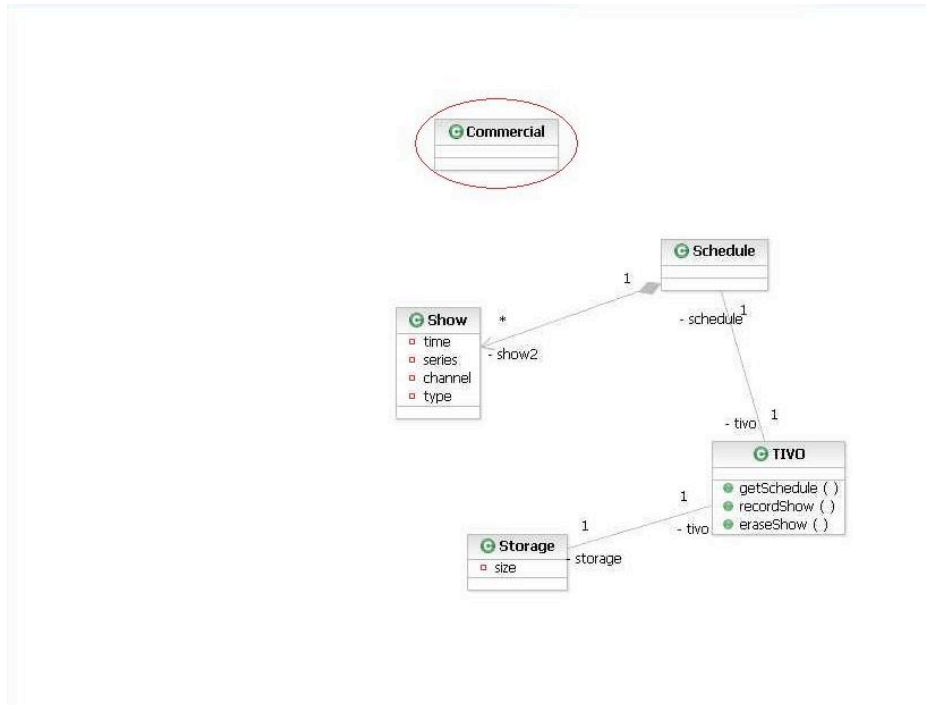


Figure 30 - Initial prototype, model screenshot with highlighting

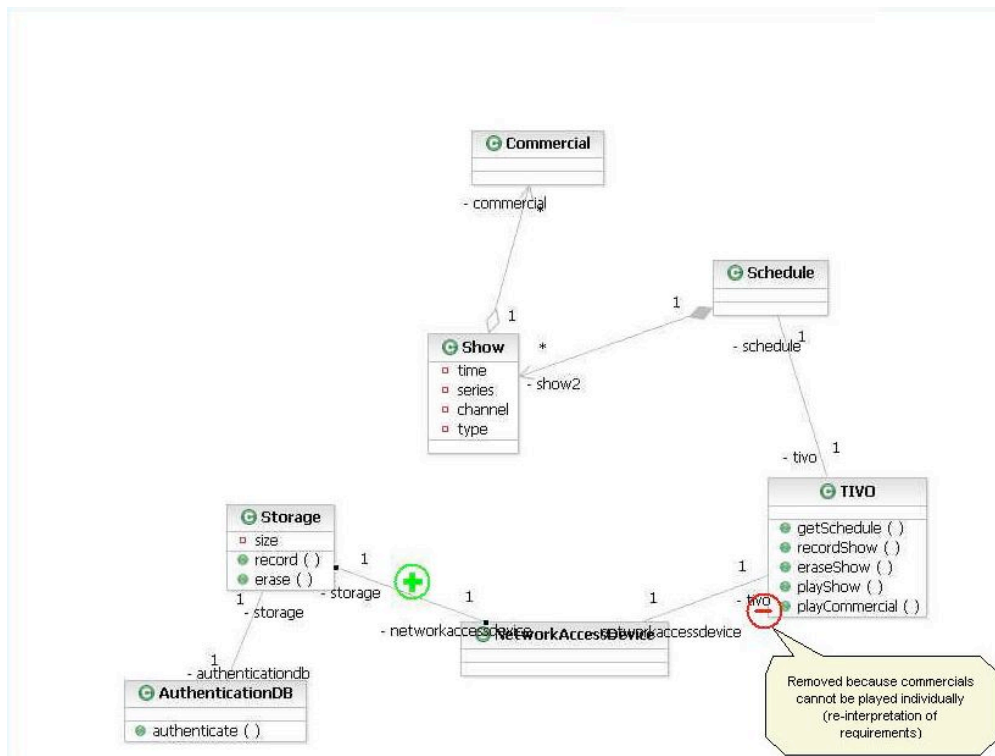


Figure 31 - Initial prototype, model screenshot with annotations

We demonstrated the initial prototype as a slide show for interested IBM staff and it was judged to be a very good idea for a new feature. After the positive feedback, we decided to start a functional prototype.

4.4 Functional prototype

4.4.1 For which tool, why?

We have decided to choose Rational Software Architect/Modeler (RSx) as our base application. The alternative was to implement a prototype for Eclipse. However, RSx is built on top of Eclipse so we can use the same Eclipse plug-in architecture including the components of RSx related to modeling. RSx has functionality already implemented that we could re-use for our prototype including the EMF change-recording functionality that allows us to capture the changes in a diagram between two different instances of time. This saves us a lot of implementation difficulties regarding comparing two instances of a diagram. The challenge here was to learn how to use the EMF change recording functionality, which we succeeded in doing. All of the functionality relies only on GMF, so it would work in any Eclipse tool using GMF.

4.4.2 Setting up the correct development environment

It was challenging to find out how to set up the needed components for the development environment.

We started by using a special version of Eclipse to build the source code for Rational Software Architect 7.0. Our initial decision was to target the prototype to plug into RSA 7, however, this version of RSA was in a build phase and wasn't stable. We encountered various exceptions when building diagrams because of missing classes or bugs (for example, a class diagram always placed classes at the top left of the diagram overlapping each other but the problem did not exist for interfaces). Another issue was compile time: the code base would take about an hour to compile and this operation was performed each time the development environment was closed and re-opened. We could not leave the development environment running all the time because it would consume too much memory and would crash. We must mention that this was annoying and wasted precious time. Even though our development machine had one gigabyte of random access

memory, we encountered a lot of out of memory problems since RSA is a huge application. We tried to reduce the number of plug-ins loaded into RSA but this operation was very confusing because of plug-in dependencies that would cause certain needed product features to stop working. Moreover, the underlying API for the Rational Modeling Platform was also undergoing changes so there was a risk that the functionality we used in our prototype might also have to undergo changes.

We consulted developers from the Rational division and we were recommended to use the Rational Software Architect 6.0.1.1 itself as our development tool because it was proven to be the most stable release for that product (as opposed to building RSA using Eclipse and the RSA source code, we could use RSA as the development environment since it has all the required binaries to support our plug-in). We followed that recommendation which required us to refactor our code for the prototype because there were major API changes between version 7.0 and version 6.0.1.1.

In addition to deciding on the target environment for our feature, we also had to decide on the development environment we would use. Again we chose RSA 6.0.1.1 as it proved to be the most stable, until we reached a stage where we encountered blocker bugs existing in 6.0.1.1 and were obligated to switch to RSA 7.0 even though it was under development. Furthermore, we decided to use RSM instead of RSA because it contained all the required modeling features less the extra features available in RSA that use more memory and CPU power. It was not until June 2006 that we were able to work with a stable version of RSM 7.0 with the debugging capability.

4.4.3 Iterations, functionality, and challenges

We used an iterative development process while implementing the prototype. This process allowed us to gather feedback and merge suggestions in the development process. It was also a successful risk management strategy as we always knew that we were developing a prototype that will please the interested parties.

Iteration 1

The first step was to create a new project and set up its dependencies on other components. We created a plug-in project to integrate our prototype within RSA. We

were required to get access rights to separate repositories to download the required base projects: this step lasted several weeks to go through the process and figure out the right components to download (about four hundred projects).

We had no prior experience with the technologies used within RSA including the Eclipse Modeling Framework (EMF) and the Graphical Editing Framework (GEF). We therefore had to discover the functionalities provided by these two technologies and evaluate how to take advantage of their offerings to build our prototype. We held meetings with development team members in order to build up knowledge of available APIs and techniques currently being used. We learned multiple approaches that could become possible solutions to be used in our tool. We had also researched and debugged the source code for locations that handle the undo and redo requests because they could hold promising functionality that could be reused in our tool. Finally, we decided to use the EMF change recorder (an alternative is discussed in the design decisions, Section 4.4.4) that provided the core of our functionality to detect and serialize changes.

The first prototype had four buttons: record, stop, move forward, and move backward. The user had to press the stop button each time a change was made in order to capture it then press the record button to track the next change. The changes were kept only in memory: this meant that the user could not review the history of a model after closing the main application since the changes were not persisted. Also, the user could not add annotations to the changes to explain the rationale behind a change. The user could, however, use the ‘move forward’ and ‘move backwards’ buttons to review the change history.

Iteration 2

In this iteration, we used the Java API’s timer functionality to schedule the change-capturing tasks. We automatically captured changes every 500 milliseconds. This time interval was chosen to be small enough to capture each user operation on the model as a separate change. The user was only required to press record once. After each capture, the EMF change recorder returned a change description that our code then checked for change content. If changes were detected, we added the change description to a list

containing all the change descriptions that were then serialized when the diagram was saved.

Serialization support enabled us to persist the change descriptions and load them if they were available inside the resource containing the diagram (a .emx file in RSA contains the UML model and its diagrams). One of the challenges was to find the location of the resource file. We had two choices to get the file path: the first choice was to use the Eclipse API to find the file associated with the editor showing the diagram; the second choice was to use the Rational Modeling Platform (RMP) API to find which file the diagram belongs to. We decided to use the RMP API because of the various editor types and the differences in determining which file belongs to a particular editor: Different editors in RSA related to different diagram types do not share the same API to discover which file is represented by the editor, however, the RMP approach was common to all types of diagrams.

The changes became persistent and saved within the model: this allowed the user to review the changes at any time and to share change history with other users (this could not be achieved if the changes are saved in memory only).

Annotation support was added: we enabled the user to add annotations while recording the changes (the user could write inside a text area and click the annotate button). We showed the annotations at the correct time while the user is going through the diagram's history (the annotation would pop-up as a message box on playback).

A slider control was introduced to enhance the tool's usability: the user could now move the slider back and forth to review the change history instead of having to press the buttons.

Iteration 3

Annotation display was improved for this iteration: the message box used to block playback requiring the user to press 'ok' in order to continue. The annotation now showed in a small modeless window and automatically disappeared after three seconds or when the user moved to the next change.

We added the ability to continue recording changes after loading changes from a file. In the previous iteration, pressing the record button would start recording a new set

of changes. We added the functionality of loading all existing changes and continuing to record future changes from that point.

Shortcut keys were added: When the user clicked on the TME view, the focus was given to the slider: the left and right arrow keys then allowed the user to go step by step through the changes. The up arrow would advance the user smoothly until an annotation shows then it stopped. The down arrow would just play all the changes. The user could press the down arrow again to pause playback (playback showed a new change every half second).

Figure 32 shows our plug-in's view after the third iteration. All the functionality of the prototype is inside the same view.



Figure 32 – Diagram Player view (before using the term TME)

Iteration 4

This iteration witnessed major functional changes. Earlier iterations allowed recording and playback for a single diagram only, and the user needed to press the record button to load and record changes.

For iteration 4, we supported automatically tracking changes on multiple diagrams. The record and stop buttons were removed; their functionality is now automated (when switching from one diagram to another, we would stop recording on the first and start recording on the second). Details of this functionality can be found in the discussion of architecture, Section 4.4.5.

We also modified our technique for capturing changes from being timer-based to being event-based as discussed in the design alternatives, Section 4.4.4.

We realized in earlier iterations that tracking changes related to the diagram only is not sufficient (discussed in the design alternatives, Section 4.4.4). In this iteration, tracking was performed on the entire model, not on the diagram.

Iteration 5

This was the final iteration. Its main enhancements were better integration, more functionality and bug fixing. We had built the base of our prototype in earlier iterations: mechanisms to capture changes, saving and loading. We now took the step of managing changes at a higher conceptual level: what do we do with the changes?

We changed the functionalities of the shortcut keys to support generally adopted concepts in other tools: ‘page-up’ and ‘page-down’ replaced the ‘up’ and ‘down’ keys. They allowed the user to jump between snapshots or temporal annotations in the history of a model. We removed the concept of the annotation disappearing after three seconds because we realized the user might not get enough time to read it. The annotation box would now disappear when the user presses any key.

We added the ability to edit or remove a temporal annotation by adding another one at the same place which would overwrite it (an empty text would mean that the annotation is to be deleted).

A palette item was added to allow adding a temporal note. We aimed to give the user a behavior that is common to his modeling routines. Since the user would use the palette to drop elements on a diagram, we thought it would be easier for him to choose to add a temporal annotation from the palette which would pop up a box and allow him to enter a message to explain a design rationale for example.

We added the capability to filter the changes of certain types. For example, we had the ability to filter out all the changes that related to moving something in the diagram. This was necessary for our final positioning feature.

We added ability to add or remove snapshots while exploring the history of a model. Another user could jump between those marked positions, therefore applying a group of change at the same time. This allows for increasing the level of granularity at which a user can explore the history of a model.

We added another console window that was initially used for debugging our prototype and later on to give confirmations to the user when loading changes, saving changes, adding or removing snapshot positions.

Figure 33 is a screenshot of the latest version of the TME prototype:

Chapter 4 – Building the TME prototype

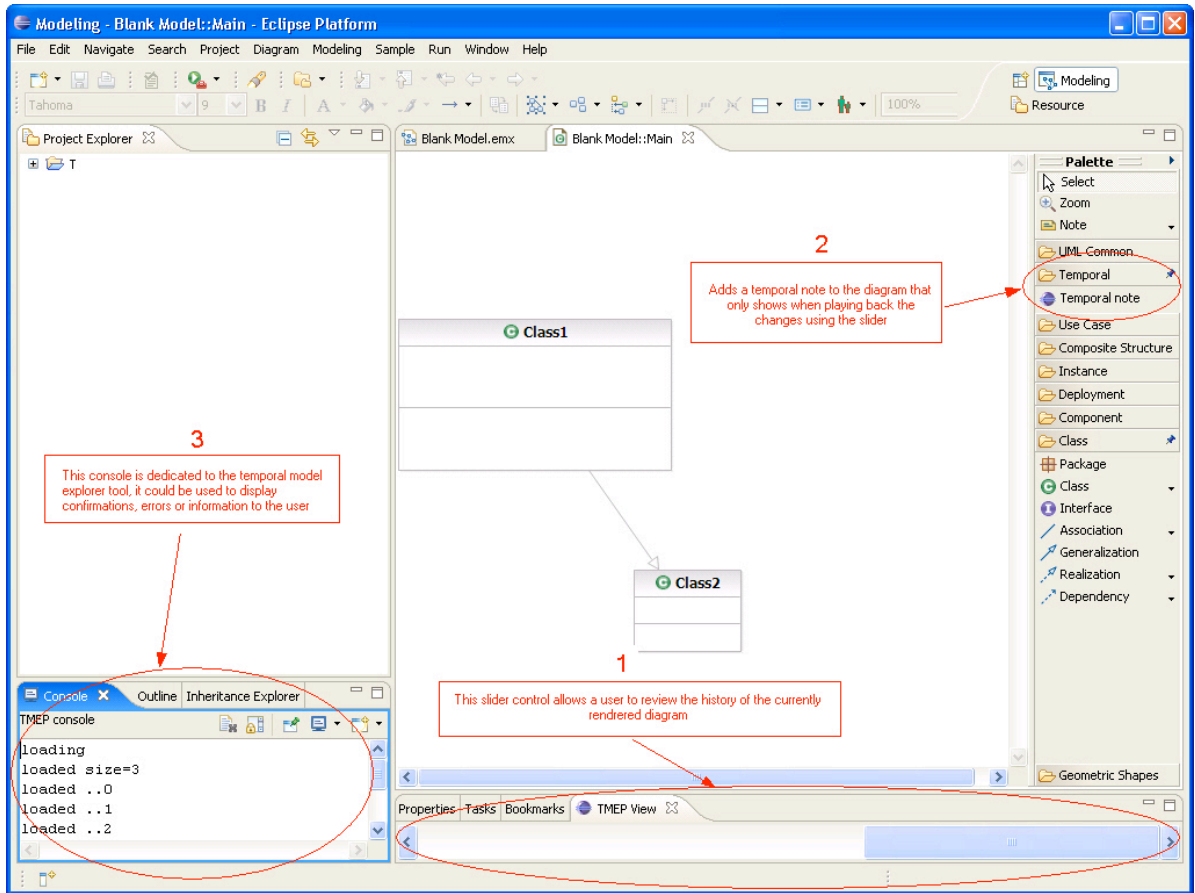


Figure 33 – Final TME prototype screenshot

Table 5 summarizes the idea behind each iteration:

Iteration #	Idea
Iteration 1	Basic UI
Iteration 2	Basic automation
Iteration 3	Basic prototype
Iteration 4	Enhanced automation
Iteration 5	More functionality, better integration

Table 5 - Prototype iterations

4.4.4 Design alternatives

As our prototype evolved, we discovered limitations and came up with various alternatives for improving our design and providing better functionality. We discuss in

the following some of the interesting issues we encountered. The objective of this section is to provide insight for future tool developers.

Event-driven versus timer-based approaches for capturing changes

We developed four different techniques for capturing changes. We learned from the limitations of the first three techniques to solve the problem of capturing changes of appropriate size in the final technique.

- a) Event-driven approach: extending the EMF change recorder
 - The first technique was to subclass the change recorder class and to extend its *notifyChanged()* method to add the functionality of capturing and saving a change description on every call. This technique provided a very fine level of granularity. For example, if a user added a new class to a diagram, calls to *notifyChanged()* would be performed after each of the following: adding the class to the top left of the diagram, moving the class to the correct *x* position, moving the class to the correct *y* position, sizing the class width appropriately, and sizing the class height appropriately.
 - The advantage of this technique was that it had support for serialization which would facilitate saving and loading changes to/from disk. The drawback of that method was that the level of granularity of the changes was too fine-grained.
 - The level of granularity provided by this alternative was not desired since it does not reflect the real actions of the user and it would be confusing for the user to see that much detail when reviewing the changes individually. Moreover, we did not want to change the user's original visual experience (when building the diagram).
- b) Event-driven approach: using *MUndoInterval*
 - The second technique was to use the *MUndoInterval* class that captures changes on the undo stack. The advantage of this method was that it captured the changes at the level the user performs them (each change is similar to what the user can undo and redo in the environment).

- However, a major drawback of this technique was that the changes cannot be serialized. We could not persist the list of changes, which was a blocker for using this method.
- c) Timer-based approach: In this approach, we tried to solve the problem in a) (fine level of granularity)
- We needed a technique to group related changes together in order to show a number of changes at the same time that made sense to the user (refer to the example on the snapshot pattern in RSA, Section 2.2.2, Figure 4: fine-grained details are grouped at a higher level, we needed to do something similar but that was very challenging, specially at the beginning of our research).
 - We tried using the Java API to provide timer functionality to perform tasks at specific intervals (ex: 500 milliseconds). At the end of each interval, we would check for changes using the EMF change recorder to compare the state of the diagram between the two points in time. If there were changes, we would add the change description to a list.
 - The approach worked, but theoretically it could provoke some undesired behaviour such as capturing only a part of a change if the interval was not small enough. Moreover, this approach was not very elegant.
- d) Event-driven: Combining the undo stack listener with the EMF change recorder
- The better and more elegant approach was to use an event based technique that would only require capturing a change in the case it actually exists. The solution that was adopted was to track the changes at the same level as the undo stack (by adding a listener that would notify us of changes on the undo stack). We would play back the changes to the user in the same way he can undo or redo them.
 - We took the best of the previous methods: we listened for events on the undo stack and captured the change using the EMF change recorder. This allowed us to control the level of granularity and to serialize the changes.

Recording on diagram or model

The EMF change recorder tracks changes on a target. Our initial design decision was to track changes on a diagram. This choice was not appropriate because adding attributes or operations to a class was not being detected. If a user followed the following steps when building a diagram: adds class A, adds another class B, adds an attribute to the class A. When playing back the history the previous sequence, the first time class A showed, it included its attribute and did not follow the same sequence of events since class B was to show before the attribute was added to class A. The reason behind this was that the diagram did not contain the semantic information of the model. This information was saved as part of the model. We distinguish between semantic information (part of the UML model) and notational information (visual to the user).

To address this limitation, we chose to record changes on the model itself. However, change information will be saved for the diagram they belong to (but they affect the entire model not only the diagram).

Recording on the diagram only would enable us to easily manipulate recording in multiple diagrams. Recording on the model complicates things because a change in the model can affect many diagrams at the same time. This would require us to manage properly which change recorders are enabled on which diagrams at a specific time in order not to capture the same change twice.

How to save changes

The diagrams are saved in an XML format; the EMF change recorder supports serializing changes as XML data. To persist a change, the TME prototype creates a new XML node to hold the change information. In the early versions of the TME prototype, it was added to the resource containing the diagram: when the user saved the diagram, the resource was saved and so the change information was saved as well.

Adding change descriptions to the resource limited it to contain a single diagram's change information (while the resource can contain multiple diagrams). To solve this problem, we know that XML nodes are nested so a more general solution was to include the change information node as a child node of the diagram node. However, this means that the editor of the diagram would be confused since it expects nodes of known types

that are part of the diagram. To address this issue, we take advantage of the fact that EMF support nodes of type *EAnnotation* that are used to add extra information to the diagram (those nodes were not rendered as diagram elements by an editor). Our prototype understands that those nodes are related to change descriptions. The *EAnnotation* has a type so we can detect which *EAnnotations* relate to change information inside each diagram.

At a later stage in our design, we decided to separate the change information from the resource file. We decided to save changes related to a diagram in separate file named after the diagram and with the extension .chg. This approach has many benefits:

- Separation the concerns: resource and change. If a user wanted to share a resource without including its history, the changes needed to be in a separate file.
- If a user wanted to delete the change history, he could simply delete the change file instead of having to go through complicated XML data and deleting individual *EAnnotation* nodes that relate to changes.
- Allowing *Multiple Approaches*: Several users could work on the same base model and produce separate change files that could be used to demonstrate multiple approaches in solving a problem.

The one disadvantage is that the change file can become disassociated from the resource file.

Model View Controller approach for extendibility

The initial approach for designing our system was to have a controller that used the EMF change recorder and listened on the undo stack to capture and record changes. The controller would send the Temporal Model Explorer view update calls in order to extend the slider with each new change description. However, we thought that this would introduce an inconvenience in the future for extending the system with other functionality since the controller would have to hold an extra variable for each class that wishes to handle the event of diagram changes; the name of the method to call in that class is also required.

Instead of the limited approach above, there is a very well-known design pattern called Observable/Observer [12] which we applied in our case. We created two new classes. The first named *ChangeInfo* to encapsulate the data related to a change description including the time stamp at which it occurred, a temporal annotation if it exists for that change and a Boolean to indicate if that change position represents a snapshot. The other class named *DiagramInfo* contains and manages all the *ChangeInfo* objects for a diagram (refer to the architecture Section 4.4.5). The *DiagramInfo* class extends the Observable superclass.

Each class that wishes to handle change events needs to implement the Observer interface and its *update* method. The *update* method is called each time the diagram change information is changed. This makes extending the system easier by implementing an *Observer* class and adding it to the list of observers of the *DiagramInfo* class.

4.4.5 Architecture and tool integration

As discussed earlier, we developed the Temporal Modeler Explorer feature prototype in Rational Software Modeler as an Eclipse plug-in. It uses three APIs: Eclipse API to provide views and detect switching between diagrams, GEF API to integrate with the GEF palette in RSM, and EMF API to capture and serialize changes.

In the following, we will refer to the various components using the following letters:

- a) A diagram tracker, which includes a listener for user interface events.
- b) A listener for commands in the diagram commands stack (i.e. the undo stack)
- c) A component of EMF that generates a delta between two versions of a model
- d) A change file corresponding to each diagram with the same name as the diagram but with the extension *.chg.xml*.
- e) A change recorder that loads and records changes in the file.
- f) A preferences page
- g) An item in the palette marked ‘Temporal annotation’
- h) A dialog to input a temporal annotation
- i) A slider (scrollbar)
- j) A key listener on the slider (h).

- k) The Temporal Model Explorer view, which contains the slider (h).
- l) A box to display a temporal annotation
- m) A tracker for the TME view

The following class diagrams and explanations demonstrate how our prototype is implemented. Note that we created all the classes shown in the diagrams. Also note the notation used to visualize attributes: public (circle), protected (triangle) and private (square).

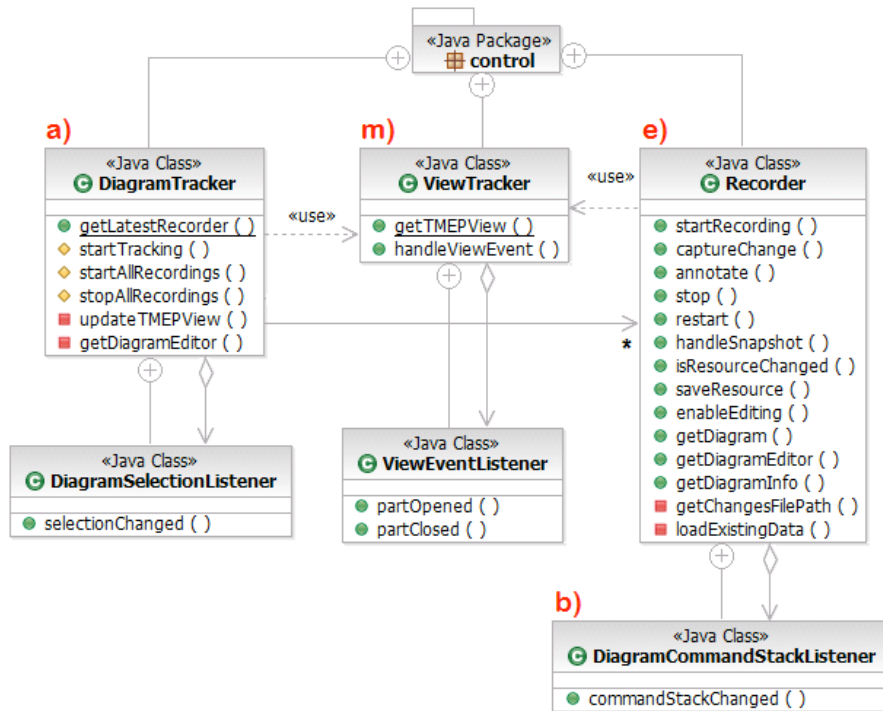


Figure 34 – TME prototype design, control package

The control package, described in Figure 34, is responsible for tracking, recording, saving and loading change data. It uses the Eclipse API to track the selection of views and editors. It can detect the selection of a diagram editor and ensure recording can be performed to capture its changes:

- The DiagramTracker (a) detects when the user switches among different views.
- If the newly displayed view shows a diagram, the DiagramTracker (a) ensures there is a change recorder (e) for that diagram, in order to track changes.
- If the change file (d) already exists for the diagram, the change recorder (e) first

loads the existing change descriptions, and continues recording at the end. Otherwise a new change file (d) is created.

- The change recorder (e) uses the command listener (b) and the EMF component (c) to obtain a change description corresponding to each change made to the diagram.
- The change recorder (e) writes out serialized change descriptions to the change file (d)

The information about each change is saved as follows:

An XML node is created with four children at most:

1. The EMF change description
2. The user's temporal annotation if included for that change
3. A time stamp of when the change was captured (can be analyzed to detect snapshots). The time stamp should be relative to the time the diagram was opened or created (this addresses the issue of sharing between multiple computers with different times or in different time zones).
4. An indication if the change is a snapshot.

Figure 35 shows the *integration* package containing the classes to integrate some features of our prototype with RSM:

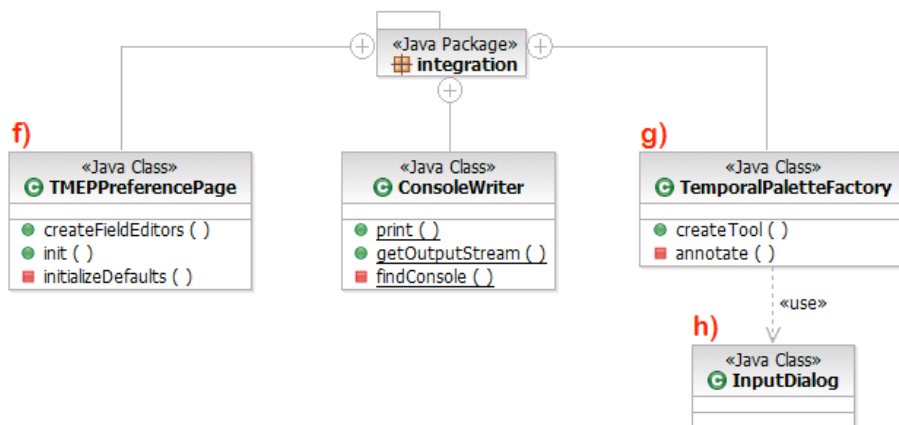


Figure 35 – TME prototype design, integration package

- When a user selects the 'Temporal annotation' palette item (g), code in the

TemporalPaletteFactory shows a dialog (h) allowing the user to enter a temporal annotation. The temporal annotation is saved along with the change description. A new temporal annotation overwrites any existing temporal annotation. If a user adds an empty temporal annotation, this has effect of deleting any existing temporal annotation.

- The *ConsoleWriter* can display messages for the user on the Eclipse console. It can be used to confirm loading or saving changes, or marking/unmarking a position as a snapshot.
- The preference page (f) can be used to enable or disable the recording functionality of the TME prototype. The user does not need to restart the application in order for the changes to take effect, the behavior is dynamic.

Figure 36 shows some of the user interface functionality:

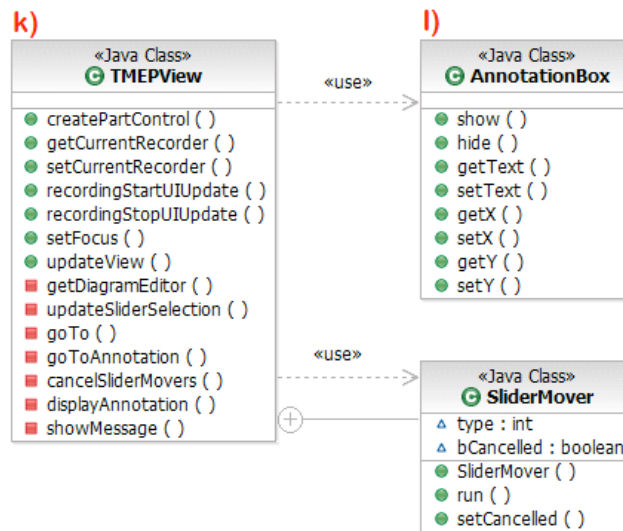


Figure 36 – TME prototype design, UI functionality

- When a user moves the slider (i) or presses certain keys (j), the Temporal Model Explorer (k) applies a set of changes (forward or in reverse) through EMF functionality. The changes are reflected on the diagram by adding or removing elements. The *SliderMover* class is a thread that handles moving several positions at a time. The goal could be to hit a snapshot or a temporal annotation location.
- If the change being applied has a temporal annotation the Temporal Model

Explorer (k) displays the temporal annotation box (l) implemented by the *AnnotationBox* class.

- If the user presses the ‘s’ key the Temporal Model Explorer (k) will ask the change recorder (e) to mark the most recently displayed change in the change file (d) as a snapshot. If it is already a snapshot, then it is no longer marked as a snapshot.
- Marking snapshots in an existing change file can be automated by sequentially examining each change in the change file (d) and applying defined snapshot rules.

Figure 37 shows how the information can be used and processed in our system:

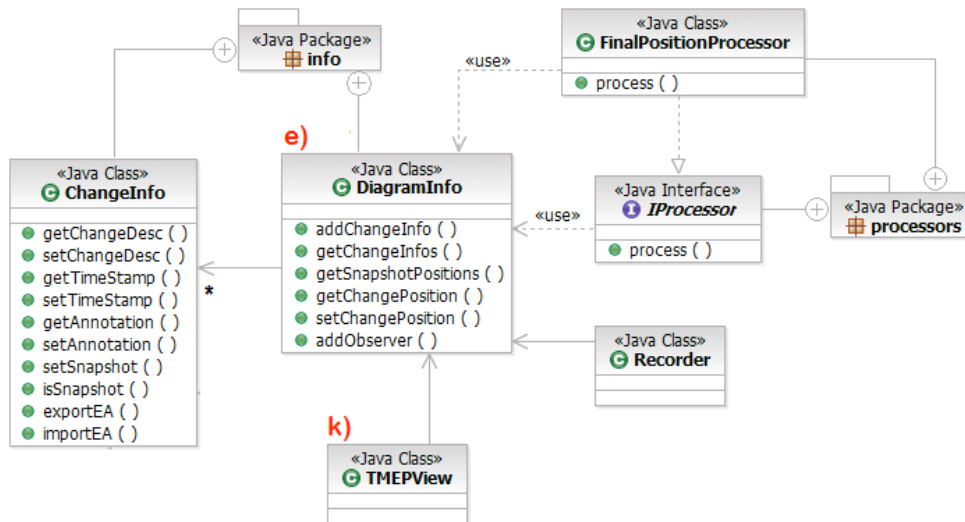


Figure 37 – TME prototype design, information usage

- The change file contents (d) can be filtered to remove change descriptions related to moving elements of the display. This implements the **final positions** feature. The *DiagramInfo* class contains a list of objects of type *ChangeInfo* that capture each change description and its related information (time stamp, annotation, snapshot). *DiagramInfo* could be processed in order to use the change in a more intelligent way, giving the user more ways to improve their understanding as we do with filtering changes related to movement.

Chapter 5: Prototype evaluation

We evaluated the Temporal Model Explorer prototype using twelve participants. We involved participants with different backgrounds to reduce potential bias.

We gathered data of two kinds: 1) The personal opinions the participants had of the tool, and 2) Performance comparisons. The latter included tracking the amount of time taken to understand a diagram and answer questions about it, as well as the quality of answers given by each participant. Performance results are discussed in Section 5.3, while opinion data are discussed in Section 5.4 and 5.5.

Overall, we aimed to validate the following hypotheses:

H1: When given a complex UML class diagram, and asked various questions about it to show he understands it, a software engineer will be able to answer the questions more quickly (H1a), and accurately (H1b) if using TME. Furthermore the software engineer will prefer to have TME as opposed to not having it (H1c).

H2: Software engineers using TME to explore an earlier stage in diagram history will benefit from having modeling elements in their final position, as opposed to where the elements were actually located at earlier times. Showing the final position should improve speed (H2a) and accuracy (H2b). Furthermore, software engineers will prefer it (H2c)

The rationale for this hypothesis is as follows: Showing elements always at their final position in the diagram should reduce confusion: the understander will not see changes related to moving elements in the diagram so he will be able to focus more on how the diagram was constructed.

5.1 Summary of the procedure

We followed the steps below to evaluate the TME prototype. It should be noted that the whole process was first approved by the Research Ethics committee of the University of Ottawa (see recruitment script and consent forms in the Appendices 2, 3, and 4).

Step 1

Two problems were selected (See Appendix 1). The author of this thesis, his supervisor (the project's 'principal investigator', Dr. Lethbridge), a CAS student, and a software engineer within IBM then developed UML class diagrams for these problems using a version of RSM in which the recording functions of the TME prototype had been activated.

Then the researchers reviewed the class diagram with the person who created it, so that additional changes and improvements could be made to it.

The result of this step was 8 models with their development history recorded. The researchers chose the best 4 models to include in the study, two of an investments system and two of an elections system. An additional model, of an airline scheduling system (See Appendix 1), was developed by the principal investigator in order to have a common model that all the experiments would use for tracking.

Given that all the participants were knowledgeable about UML, the total time required per participant was about 90 minutes (45 minutes per model).

Step 2

We conducted some formal experiments to investigate the hypotheses. Details of the experiments setup are discussed in the next section (5.2).

Step 3

We administered a short questionnaire (after each experiment in step 2) containing preference questions with fixed answers ranging from 'strongly agree' to "strongly disagree" and two questions for the user to write his positive experience and suggestions for improvements. Details of the questionnaire are found in Appendix 5. The results of the preferences questions are discussed in Section 5.4. The user suggestions for improvements are grouped into categories in Section 5.5.

5.2 Details of the experiment setup and procedure for Steps 2 and 3

5.2.1 Participants

Twelve participants were selected to answer questions about UML class diagrams. The participants all had to have some experience creating UML class diagrams. Participants were coded A to L to maintain anonymity.

5.2.2 Independent Variable

The independent variable we were concerned with manipulating in this study was one of the following three ‘treatments’ consisting of particular setups of Rational Software Modeler (RSM):

T1

‘Only final diagram’: Standard RSM without the Temporal Model Explorer feature. The participant studied the class diagram without any ability to explore its history.

T2

‘TMEP original pos’: RSM with the TMEP view at the bottom of the screen, and the model set up with the slider at the far left. By using page down (or page up to go back, or arrow keys to see smaller increments), the participant can step through the editing history of the class diagram. The participant can see the final class diagram as in T1 by pressing page down enough times, or pressing ‘End’.

T3

‘TMEP final pos’: Same as T2 except that the model elements always appear in their final position in the diagram while exploring its editing history, i.e. not in the positions they were originally placed.

5.2.3 Variables controlled, and blocking

The following variables were controlled to the greatest extent possible to minimize their influence and improve generalization of the results. Tables 6 and 7 show how the blocking was done.

Problem

Three different problems were used to create the original models, an Airline system, an Investments system, and an Elections system. Each participant saw one model for each problem. Using three models ensured the result generalizes to multiple models.

Problem sequence

All possible combinations of model sequences were used with various participants to ensure there was no transfer of learning.

Person creating original model

Models from three different people were used

Model

Five different models were used, one from the Airline system, two from the Investments system, and two from the Elections system.

Treatment pattern

Four treatment patterns were used. Each treatment pattern involved the participant doing work with three models. The treatment patterns were named using a three-number code with the numbers corresponding to the treatment. The lower-case 't' was inserted to indicate the point in the sequence at which the TMEP training would be performed. This training is discussed in Section 5.2.5.

The first two treatment patterns involved T1 (only final diagram) first, followed by training in TMEP, then the two possible sequences of T2 (TMEP original pos) and T3 (TMEP final pos).

- 1 t32 (i.e. T1, followed by training, followed by T3, followed by T2)
- 1 t23

The second two treatment patterns involved training in TMEP first, then the two possible sequences of T2 and T3, then finally T1

- t32 1
- t23 1

Participant ability

The experiments were careful to ensure that participants with higher experience were distributed over the other variables (such as treatment pattern) as were participants

Chapter 5 – Prototype evaluation

with lower experience. This was to avoid biasing a treatment pattern or treatment with people with a particular experience level.

Those users who had industrial experience creating class diagrams for real problems, or who had worked on the class diagram features of software engineering tools were classed as high experience. Those who only had classroom experience creating class diagrams were classed as low experience.

Table 6 shows the allocation of participants A-K to five models (horizontal axis) and treatment (vertical). The number after the participant letter indicates whether this model (row) and treatment (column) was first, second or third in the sequence for that participant.

	T1: Only final diagram	T2: TMEP original pos	T3: TMEP final pos
Investments-A	A1	G1	C2
	E3	K2	I2
Investments-B	B1	H1	D2
	F3	J2	L2
Elections-A	C1	F2	A3
	H3	I3	J3
Elections-B	D1	E2	B3
	G3	L1	K1
Airline	I1	A2	E1
	K3	C3	G2
	J1	B2	F1
	L3	D3	H2

Table 6 - Allocation of participants to models

To further reduce bias, we allocated the twelve users making sure to have at least one user with higher UML experience allocated to each of the four possible problem sequences. This is illustrated in Table 7.

Participant	Experience	Treatment pattern	Problem sequence
A	Higher	1 t23	IAE
B	Lower	1 t23	IAE
C	Higher	1 t32	EIA
D	Lower	1 t32	EIA
E	Higher	t32 1	AEI
F	Lower	t32 1	AEI
G	Higher	t23 1	IAE
H	Lower	t23 1	IAE
I	Not applicable	1 t32	AIE
J	Not applicable	1 t23	AIE
K	Not applicable	t23 1	EIA
L	Not applicable	t32 1	EIA

Table 7 - Blocking of participants

5.2.4 Setup of the equipment

We created detailed documents describing what was to be said and done in each experiment session, in order to ensure that nothing would be omitted, and all sessions would be consistent. These forms had places in which to record such information as timings (refer to Appendix 7).

Prior to each session we set up a computer with each required diagram pre-loaded and set to the correct initial state.

We also set up a spreadsheet to analyze the data. Following each session we entered the data into the computer and sanity-checked it, e.g. to make sure that the timings made sense.

We had planned to consider the first couple of participants to be part of a pilot study, however, their sessions went without a hitch, so we dispensed with the need to ‘start again’.

5.2.5 Conduct of the experimental sessions

Each session of the experiment was conducted in the following manner.

Participants were first given the consent form and asked to read and sign it. One participant backed out very shortly into the study over personal concerns; this person's data was not counted.

Prior to working on the first model using T2 or T3, the participants were given a short training session in the TME feature. The training session consisted of showing the participant a class diagram of a University system, and showing the participant how the home key would rewind the history of the diagram back to its 'empty' starting point. The experimenter then demonstrated how pressing the arrow and page-up/page-down keys would allow navigation of the history. The experimenter ensured that a temporal annotation appeared in this process. Finally, the participant was given the chance to play with the feature using the University system, and was asked to let the experimenter know when he or she understood the user interface well enough to proceed.

For each model in the treatment pattern, the experimenter first revealed on the computer screen the model in its initial state. Participants were asked to take some time to understand it. When doing T1, they could simply look at the class diagram on the screen. When doing T2 and T3, they could use the facilities of TME – the model was presented in initial (empty) state, so participants were forced to use TME in some way.

Both experimenters (the author of this thesis and his supervisor) started timers to time the period the participant took to understand the model. The reason for having two separate timings was twofold: Firstly, if one researcher forgot to start timing for a given activity, then data would not be lost. Secondly, the exact end of an activity is slightly subjective, so having two people detecting it helps reduce systematic bias.

Participants were asked to let the experimenter know when they felt they understood the model well enough to start answering some questions about it.

Participants were asked to answer three specific questions about each model. Participants were passed the first two questions on a paper and asked to write down their answer. The third question was answered orally and the experimenters took notes.

We recorded the time that a participant took to arrive to each answer. Following the experiment, each answer was evaluated for correctness.

Again the author and his supervisor recorded timing separately. Upon studying the data after a few participants had completed their work, we noticed that both people were using two slightly different timing criteria: Timing 1 took in consideration everything the participant said before moving to the next question while Timing 2 stopped as soon as the participant wrote his final answer on paper. Timing 2 would account for additional time if the participant decided to change his answer. In the end both timings showed statistically the same or very close results. Overall, each session took between 45 minutes and one hour.

We used a correctness scale between 0 and 5, 5 representing a correct answer. The answers were evaluated by the principal investigator as he wrote the problems and has the necessary skills and teaching experience to do this evaluation.

5.3 Results of performance improvements tests

5.3.1 Time and accuracy answering questions

First we analyzed the results including the complete set of twelve participants. The numbers in the speed and accuracy tables are normalized values by model and by participant. The average for all timings within the timing method (1 or 2) is set to 1. So for example, a mean of 1.06 for ‘-TME’ means that not using TME it took 6% longer than the average. Normalization by model was necessary so overall performance differences on the five models considered will not bias the results (to account for models that were easier or harder than other models). Normalization by participant was necessary so participants who are overall better or worse won't bias the results.

Table 8 and 9 show the results for twelve participants (‘+/- TME’ columns refer to users using/not using the TME prototype and ‘Orig.P./Final P.’ columns refer to the absence/presence of the final positioning feature while using the TME prorotype):

	Timing 1				Timing 2			
	- TME	+ TME	Orig. P.	Final P.	- TME	+ TME	Orig. P.	Final P.
Mean	0.99	1.00	1.04	0.97	1.01	0.99	1.04	0.94
95% c.i.	0.14	0.07	0.14	0.08	0.18	0.09	0.19	0.12

Table 8 - All participants, answering speed

	- TME	+ TME	Orig. P.	Final P.
Mean	1.00	1.00	1.01	0.98
95% c.i.	0.08	0.04	0.08	0.05

Table 9 - All participants, answering accuracy

The overall accuracy results using the TME prototype were equal to the results without using TMEP, not validating hypothesis H1b. Hypothesis H2b was not validated either, in fact, it was slightly reversed but without any statistical significance. If we consider the speed of answering questions, hypothesis H1a was evaluated to be neutral: overall, participants took the same amount of time to answer questions with or without the TME prototype. However, participants took less time to answer questions when they used the final positioning feature, validating H2a.

Next we analysed the twelve results excluding the four expert participants (those who ranked themselves as highly knowledgeable or expert in UML). Tables 10 and 11 summarize the results of the non-expert participants. We show the results for both timing strategies that we applied. Both results show the same conclusions.

	Timing 1				Timing 2			
	- TME	+ TME	Orig. P.	Final P.	- TME	+ TME	Orig. P.	Final P.
Mean	1.02	0.99	1.05	0.93	1.06	0.97	1.07	0.86
95% c.i.	0.14	0.07	0.16	0.09	0.18	0.09	0.20	0.15

Table 10 - Non-expert participants' answering speed

	- TME	+ TME	Orig. P.	Final P.
Mean	1.05	0.98	0.98	0.97
95% c.i.	0.10	0.05	0.08	0.07

Table 11 - Non-expert participants' answering correctness

We notice that the non-expert participants were able to answer questions faster using the TME prototype compared to understanding a final static model (hypothesis H1a

valid). We also notice average participants taking less time answering the questions using the final positioning feature (hypothesis H2a valid). However, results for better accuracy were not validated using the TME prototype (H1b invalid), in fact the result was reversed but without statistical significance. Using the final positioning showed to have a neutral effect on accuracy (H2b neutral).

On the other hand, if we consider only the four expert participants, the conclusions regarding our hypotheses are different. Expert participants answered questions more accurately using the TME prototype (H1b) than without. The speed factor was neutral using TMEP (H1a). The hypotheses that playing the changes using the final positioning feature would increase the speed and accuracy for answering questions were not validated for expert participants.

Tables 12 and 13 show the results for expert participants.

	Timing 1				Timing 2			
	- TME	+ TME	Orig. P.	Final P.	- TME	+ TME	Orig. P.	Final P.
Mean	1.01	1.00	0.91	1.09	1.02	0.99	0.84	1.14
95% c.i.	0.15	0.07	0.06	0.10	0.17	0.08	0.08	0.11

Table 12 - Expert participants' answering speed

	- TME	+ TME	Orig. P.	Final P.
Mean	0.94	1.03	1.05	1.01
95% c.i.	0.15	0.08	0.16	0.02

Table 13 - Expert participants' answering accuracy

Table 14 summarizes our finding for the various participant categories, including which hypotheses were validated and which were not. Note that statistical significance was not achieved in these results, so when we say ‘positive’ we are merely saying there is good suggestive evidence in favor of the hypothesis, and when we say ‘negative’, we are merely saying there is suggestive evidence for the opposite of the hypothesis.

	H1a (TME enables answering questions more quickly)	H1b (TME enables answering questions more accurately)	H2a (Final position enables answering questions more quickly)	H2b (Final position enables answering questions more accurately)
All participants	Neutral	Neutral	Positive	Negative
Experts	Positive	Negative	Positive	Neutral
Non-expert	Neutral	Positive	Negative	Negative

Table 14 - Hypotheses evaluation by participant groups

5.3.2 Initial understanding time for participants

The above section related to answering questions following a period of understanding. In this section we analyse how the TME prototype and the final positioning feature affected the initial understanding time for the participants.

Table 15 shows the understanding time taken for all twelve participants.

	- TME	+ TME	Orig. Pos.	Final Pos
Min	56	106	74	137
Max	225	317	385	425
Mean	118.8	234.5	234.3	234.7
95% c.i.	32.0	42.7	53.0	49.2

Table 15 – All participants’ understanding times

Overall, we notice that the participants took more time to understand the diagrams using the TME prototype and that the final positioning feature did not make a difference. However, you will note in the next section (5.4) that all the participants agreed that the prototype helped them understand faster. We hypothesize that the quality and the amount of understanding the participant was able to achieve using TMEP is greater than trying to understand a static final diagram. Future studies could try to validate this hypothesis by asking the participant a larger number of questions related to the diagram.

We separately evaluated the times taken by those six participants who ended up scoring better than the average when they later answered the questions, then the ones who scored below the average.

Chapter 5 – Prototype evaluation

Table 16 compares the minimum, maximum and mean understanding times using TME prototype or not and using the final positioning feature or not:

	- TME	+ TME	Orig. Pos.	Final Pos.
Min	56	158	164	145
Max	225	317	385	334
Mean	138.2	255.3	266.2	244.3
95% c.i.	72.7	68.1	85.3	71.8

Table 16 - Above-average participants' understanding times (all values in seconds)

This group of participants saved understanding time using the final positioning feature. All values (min, max, and mean) indicate this. However, the confidence interval shows clearly that there was not enough data for statistical significance.

Another interesting result is that the participants who scored less than the average for answering the questions took less time to understand the models. The final positioning feature did not help those participants to understand the diagrams faster. Table 17 contains the times for six below-average participants:

	- TME	+ TME	Orig. Pos.	Final Pos
Min	61	106	74	137
Max	147	309	349	425
Mean	99.3	213.8	202.5	225.0
95% c.i.	28.8	83.0	97.8	106.1

Table 17 - Below average participants' understanding times

Table 18 shows the results for each of the participants' average correctness and over or under self-estimation of ability. The actual ability column is measured depending on the correctness with which the participant answered our questions; the data is normalized so that the average equals 1. The declared expertise column was chosen by the participant in one of the preferences questions. The over- or under-self-estimation is calculated by computing a normalized value of declared expertise (setting the mean to 1),

and then computing how much this normalized compared value exceeds (or is below) the measured expertise.

Participant ↓	Actual ability (mean = 1)	Declared expertise	Over (>1) or under (<1) self-estimation
A	1.17	4	0.88
B	0.91	3	0.96
C	0.96	5	1.23
D	0.73	3	1.20
E	1.02	5	1.15
F	1.08	2	0.68
G	1.06	2	0.69
H	0.81	3	1.08
I	1.15	3	0.77
J	1.11	3	0.79
K	1.01	2	0.73
L	0.95	2	0.77

Table 18 – Participants’ over or under estimation of self-ability

The correlation coefficient between measured ability and declared expertise was 0.02, which indicates almost no relationship. We conclude from these results that the participants declared self-ability is questionable.

Refer to Appendix 6 for complete user study data.

5.4 Participant preference

All twelve participants were all very satisfied from the benefits of using our prototype. The ratings they gave (on a scale between 1 and 5, respectively ranging from strong disagree to strongly agree) to the preference questions clearly show how their experience of understanding a model was enhanced by using our prototype. Preference questions details can be found in Appendix 5.

Chapter 5 – Prototype evaluation

Question 1 addressed time taken for the participant to gain an understanding of a class diagram. All the participants agreed that the TME prototype helped them understand class diagrams more quickly. The mean value for this question was 4.2/5 with a very strong 95% confidence interval of only 0.22. This means that we can be 95% confident that the population mean would be at least 3.98 out of 5, where any value above 3 would constitute a positive response.

Note that, as discussed earlier, in practice the time taken to understand diagrams was actually longer using TME, so the results of Question 1 disagree with participants' actual performance.

Question 2 addressed the concept of snapshots, in particular it asked whether the grouping of steps in the development of the model was useful to the participant. Most participants agreed that the increments while using page up and page down were of an appropriate size. The mean value for this question was 4.1/5, and the 95% confidence interval is 0.45.

Question 3 asked whether the participants preferred that classes appeared in their final positions in the diagram and did not move when exploring history. Overall, this question received a mean of 3.8/5 and a 95% confidence interval of 0.8 which means participants preferred final positioning, and we can be statistically confident that the population mean would be above the neutral value of 3.

However, two participants (managers) strongly disagreed with this, they thought that seeing classes move around in the diagram might present some design logic. Excluding those two of twelve participants would raise the mean value to 4.3/5 with a confidence interval of 0.42. Note that this is an optional feature that our prototype features for playing back the history of a model, so we do satisfy all of our users – those who don't like it can turn it off.

Question 4 asked whether the participant would actually use our prototype if it was available in his work environment and they were asked to understand a class diagram. The results were positive. The mean value is 3.9/5 with a 95% confidence interval of 0.45. This is encouraging and proves that we are helping developers get more out of software tools.

Chapter 5 – Prototype evaluation

Question 5 evaluated whether important classes appeared at earlier stages in the history of a model and less important classes came at later stages. In general participants agreed with a mean of 3.4/5 and a 95% confidence interval of 0.56. This question was of general interest, and did not serve to evaluate the prototype itself. The confidence interval is low enough, such that for this question we cannot be certain the population mean would be above neutral 3.

Question 6 took another approach: we reversed the direction of the question and asked about the participant’s negative experience instead of the positive experience. We asked the participants if using the TME prototype resulted in them taking longer to answer the questions. Most participants disagreed, although some were neutral about this. The mean value for this question was 2.4/5 and the 95% confidence interval is 0.29, indicating that this is statistically significant. Asking questions in a negative way is common practice when using a Likert scale – having questions with both polarities serves to double-check the results.

Similar to question 6, the final question (Question 7) about the participant’s experience addressed a usability issue: was the prototype awkward to use? We calculated a mean value of 1.7/5 and a 95% confidence interval of 0.37 clearly showing that the participants found this prototype to be easy to use. This is positive because users would naturally be more willing to use a simple tool.

Table 19 groups the rankings of participant preferences including minimum value, maximum value, mean value, standard deviation and 95% confidence interval:

	Q1 Faster understanding	Q2 Useful snapshots	Q3 Prefer final pos.	Q4 Would use it	Q5 Important class first	Q6 More time to answer	Q7 Awkward- ness
Max	5	5	5	5	5	3	3
Min	4	2	1	3	1	2	1
Mean	4.2	4.1	3.8	3.9	3.4	2.4	1.7
StDev	0.4	0.8	1.4	0.8	1.0	0.5	0.7
.95 cfd	0.22	0.45	0.80	0.45	0.56	0.29	0.37

Table 19 - Participants’ preference data

All of our results calculated from participant preferences are positive. This is an excellent sign that this prototype could become a successful tool feature.

5.5 Additional participant feedback

We compiled Tables 20 and 21 to show the positive experiences of participants and their suggestions for improvements of the prototype. The participants wrote these in their answers to open ended questions. We mark a box in the tables with an ‘x’ if the participant mentioned the described positive aspect or suggestions in his answers to Questions 11 and 12.

Positive participant experience	A	B	C	D	E	F	G	H	I	J	K	L
Ease of use							x		x	x		
Ability to go back and forth between steps					x			x				
Stepping between snapshot	x		x					x			x	
Intuitive/Enhanced learning/Not overwhelming						x	x		x	x		x
Useful temporal annotations	x	x	x	x	x	x		x	x		x	x

Table 20 - Usability study, positive participant experiences (columns represent participants)

It is remarkable that almost every participant mentioned the usefulness of the temporal annotations. This shows that capturing temporal design decision information in the model is considered extremely important for understanding. However, this requires that the creator of the model makes the decision of including a temporal annotation, although not necessarily at the moment that the change is made. On our part, we have integrated this functionality inside the palette that the model creator uses to create the model. We aimed to make this as visible and easy to use as possible in order for this feature to be adopted.

The participants also suggested many useful extensions to the prototype. These are listed in Table 21 (columns indicate participants).

	Suggested Improvements	A	B	C	D	E	F	G	H	I	J	K	L
1	Filter changes related to moving elements							x					
2	Maintain final variable names throughout the history							x					
3	Highlight new changes		x	x	x		x	x					
4	Handle multiple diagrams on the same model	x											
5	Show comparison of snapshots on the same surface			x									
6	Check points that we can jump to instead of sequentially moving between snapshots			x		x							
7	Stepping buttons need to work when the diagram is selected (when TMEP view does not have focus)					x							
8	Faster operation					x							
9	Separate semantic and notational changes					x							
10	Scalability issues								x				
11	Label each iteration in the timeline				x							x	
12	Displaying the annotation box should not block the diagram view										x		
13	Show the number of steps left to reach the end										x		
14	Annotation box should be displayed even when going backwards in the history										x		x
15	Ability to choose what types of changes to show or not												x

Table 21 - Usability study, participant suggested improvements (columns indicate participants)

We will discuss in the following the suggested improvements grouped in the following categories: change management, visualization, operation, and navigation.

5.5.1 Change management

Suggestions (1), (2), (9), and (15) are related to filtering. We already support filtering changes related to movements (1) by keeping the elements in the final position in the diagram. A participant suggested that we also maintain the final name given to an element (2). This means that we would filter out changes related to renaming elements. We could hypothesis that this would give the user a better perspective on what the final design is going to be. Another participant suggested (9) separating changes that affect the

user interface (notational changes) and semantic changes that only affect the semantic UML model without being reflected in the UI. A generalization of (2) and (9) is (15), we support filtering in our prototype's architecture by creating a class implementing the *IProcessor* interface. Each filtering class could manage the changes following a particular strategy. Further user studies could determine which strategies are useful to the users. Specific types of users might need specific strategies: a system architect perspective might be different from a junior programmer.

5.5.2 Visualization

Suggestions (3), (5), (12), and (14) discuss visualization enhancements. We already discussed the idea of highlighting new changes (3) but did not achieve it in the current prototype because of time constraints. Techniques for highlighting new changes are described in Chapter 4. We have investigated how highlighting could be performed within GMF and we will incorporate it in future versions of the prototype.

Another visualization technique is to superimpose two representations of the diagram using shading to separate their elements (5): the previous representation could be faded out so the new additions would stand out clearly to the person understanding the changes.

A small bug was mentioned in (12): the annotation box displaying the temporal annotation could perhaps hide elements of the diagram. We will need to find a strategy to display this box in an empty location in the diagram, let this box have some degree of transparency or dedicate a particular view in the application to display annotations (a challenge here would be to make sure the user notices that an annotation has been displayed in the view).

We had decided to display an annotation if the person understanding the changes was going forward in time, but not backward. We made this decision because we wanted to show how the original creator of the diagram was thinking. However, some participants also wanted to see that temporal annotation when navigating backwards in time (14). The reason behind this was that it is confusing that they don't see the same elements being shown and hidden (if the annotation should show whether its change was being applied or reversed). We could include this option as a preference in our prototype.

5.5.3 Operation

Suggestions (4), (8) and (10) relate to the operation of the prototype.

Tracking changes on multiple diagrams (4) within a model is part of the architecture of our prototype. Currently, there are some bugs when recording changes on multiple diagrams, this functionality works well for playing back changes. The bugs will be addressed in the future.

One participant wanted to instantly jump between the start and end of a diagrams history. Currently, we need to apply or reverse changes sequentially in order to move between two points in time. This is the architecture of the EMF change recorder that our prototype depends on. One participant did not like the fact in a particularly complex diagram it took a few seconds to jump between the start and the end of the set of changes. A suggestion to improve performance was to disable refreshing the user interface (diagram) until all the changes have been applied.

Another issue was the scalability (10) of our prototype. Currently, change files captured for our models were between 166 KB and 385 KB. We noted that the change file is larger than the model itself, since model size was between 68KB and 96KB. We do not have control over the size of the change descriptions as they are serialized by the EMF change recorder. However, they are XML data with many repetitive textual elements, they could be compressed significantly. Further studies could determine how a change files grows over time.

5.5.4 Navigation

Suggestions (6), (7), (11), and (13) addressed issues related to navigating changes.

Currently, we have a slider control that shows the user the where he is in the changes timeline. A participant wanted a more accurate location description (13) by showing the number of steps left to reach the end.

There is no mechanism to quickly move between points in the timeline. Some participants wanted to jump among a list of checkpoints (6). Labeling snapshots (11) in the timeline is also desired: we could have an indicator in the timeline with a tip that is displayed when the user hovers over a snapshot with the mouse.

Chapter 5 – Prototype evaluation

The controls to navigate the history are tied to the slider. The user exploring the history of a model needs to select a particular view in RSx (Eclipse view), in order to use the keys to navigate. Sometimes, an element is out of the scope of the visible diagram area and the user needs to use the scrollbar in order to view it. A couple of participants forgot to click back on the TME view and wondered why the keys would not work. A more convenient way to navigate the changes would be to tie the keys to the diagram (7): a user would be able to press a navigation key while having the diagram view selected so he doesn't have to switch between views.

Chapter 6: Conclusion

6.1 Problem statement

People face limitations in quickly understanding a complex artefact such as a UML model. As the artefact has been developed other time, many temporal aspects are not embedded in its final static representation. These temporal details are important for understanding. Current software development environments present features with limited support to the Temporal Details patterns. Users do not know what the most important elements in a model are. They are overwhelmed by a great number of details. People encounter difficulties understanding design decisions in UML class diagrams since they are unaware of the rationale that led the design to be the way it is.

6.2 Proposed solutions and their effectiveness

We proposed a tool that captures model and diagram changes and allows users to add annotation associated with any change. The tool allows playing back the changes and viewing and editing the annotations. Snapshot marking allows the user to navigate the changes at various levels of granularity. The ‘final positioning’ feature can filter out particular types of changes (related to movement) allowing the user to only focus on how the diagram was constructed.

The tool idea is based on the cognitive patterns category Temporal Details. The main idea of the tool is to show a software representation dynamically and incrementally, this is the idea of capturing temporal details that are usually lost in the final static representation of the system. We directly support the Snapshot and Meaning patterns by allowing the user to mark snapshot positions in the history of a model and to attach a temporal annotation to any change at any position in the history of the model. The Long View pattern is supported by jumping between snapshots while reviewing the history. The Quick Start pattern is supported by letting the user understand a diagram incrementally starting at the point the diagram was created. The Multiple Approaches pattern is future work.

The following represent some of the advantages to our approach compared with the other known solutions:

a) The understander can step through history and add annotations at different levels of granularity, unlike other approaches such as persistent undo and configuration/version management.

b) Control of what aspects of history can be explored are controllable by the understander, whereas configuration/version management approaches put that control largely in the hands of the modeler alone.

c) Movement through history is in real-time, unlike in configuration/version management approaches which require discrete interactions.

d) Temporal annotations can be added and manipulated at any time, unlike in version management tools.

e) Unlike change tracking as in a word processor, all changes back in time were tracked, not just the last set.

f) Unlike persistent undo, the final model is preserved when the understander looks back in time.

Our empirical study with twelve industrial participants showed that practitioners overwhelmingly approve of the feature and would use it if it were installed when they have to understand class diagrams.

We attempted to obtain evidence that performance (in terms of time savings and better answers) of practitioners would be improved when using our prototype. However, results were mixed and generally not statistically significant. There was significant evidence (for expert users), however suggesting that displaying the ‘final positions’ of model elements is better than showing their original positions, when viewing the earlier state of a diagram.

Our overall conclusion is that the TME feature would be useful and should be deployed. It could attract customers to IBM’s product line (in a small incremental manner), and would help modelers feel they can better perform their work.

6.3 Threats to validity

In our user study we attempted to control a wide variety of variables, therefore reducing threats to validity. However, some of the remaining threats to validity are the following:

- Low number of participants: Participant time is expensive, so we limited ourselves to twelve people. It is possible that with very significantly larger numbers of people, our results might have been different: In particular, we would have eventually obtained statistically significant results on our performance tests.
- Questionable expertise level of participants: We observed that participants tended to have lower expertise at modeling than we expected, and were generally poor at self-assessing their levels of expertise.
- Self-evaluation of whether participants would use the TME feature: Although the participants were enthusiastic about the prototype, and said they would use it, they may have been over-optimistic. In an extended study it would be necessary to install the tool and observe their use of it over time.
- Limited population from which the sample was drawn: The participants were primarily UML tool developers or their managers, not people doing large-scale modeling. They may be biased in favour of tool features in some way.

6.4 Future work

The Temporal Model Explorer prototype opens doors for other features based on cognitive patterns including Multiple Approaches: while playing back the diagram history, a user might decide to stop at a certain point and continue the design from a different approach. The TME prototype could be extended to provide support for this functionality allowing the users to create and view multiple design approaches.

Multiple levels of snapshots could be incorporated: we currently support only one level of snapshots that groups a set of changes. An additional feature would be supporting a snapshot that groups a set of other snapshots. This would be particularly useful if the set of changes is very large. It would allow users to quickly navigate through the entire history and then review in more detail the evolution between two selected higher-level snapshots.

Highlighting changes between two consecutive steps in the history would increase the user-friendliness of our prototype. We already attempted to implement this functionality and it will become available in future versions of the TME prototype.

Chapter 6 – Conclusion

Other interesting features that would increase the user's performance using our prototype is to allow users to search temporal annotations, having the capability of placing the user at a position in the history where a certain artefact was created, and allowing the user to re-order snapshots in order to provide alternative explanations.

Further user studies could include a larger number of participants, more complex models, and more questions per model in order to attempt to get more statistically significant results.

References

- [1] Adam Murray, *Discourse Structure of Software Explanation: Snapshot Theory, Cognitive Patterns and Grounded Theory Methods*, Doctoral thesis, Computer Science, University of Ottawa, 2006.
<http://www.site.uottawa.ca/~tcl/gradtheses/amurray/>
- [2] Adam Murray, Timothy C. Lethbridge, “A Brief Summary of Cognitive Patterns for Program Comprehension”, *Working Conference on Reverse Engineering*, Delft, Netherlands, IEEE Computer Society, pp. 304-305.
- [3] Adam Murray, Timothy C. Lethbridge. “Cognitive Patterns for Program Comprehension: Temporal Details”, *Pattern Languages of Programs (PloP) 2005*, Allerton Park, IL, USA, <http://hillside.net/plop/2005/proceedings.html>
- [4] Adam Murray, Timothy C. Lethbridge. “On Generating Cognitive Patterns of Software Engineering” CASCON 2005, Toronto, October, IBM, in ACM Digital Library, pp. 129-139.
- [5] Ahmed Seffah and Alina Andreevskaia. “Empowering Software Engineers in Human-Centered Design”, *Proc. 25th International Conference on Software Engineering (ICSE)*. IEEE Computer Society. 2003, pp. 653-659.
- [6] Andrew Walenstein. “Theory-based Analysis of Cognitive Support in Software Comprehension Tools”. *10th International Workshop on Program Comprehension (IWPC'02)*. IEEE. 2002, pp 75-84.
- [7] Borland’s website, <http://www.borland.com/ca/products/together/index.html>, visited on May 8, 2006.

References

- [8] Borland StarTeam website
<http://www.borland.com/us/products/starteam/index.html>, visited on September 10, 2006
- [9] Brian P. Bailey, Joseph A. Konstan. “Are Informal Tools Better? Comparing DEMAIS, Pencil and Paper, and Authorware for Early Multimedia Design”, *Proc. SIGCHI conference on Human Factors in Computing Systems*. ACM Press. 2003, pp. 313-320.
- [10] Davor Cubrani, Gail C. Murphy, Janice Singer, Kellogg S. Booth, “Learning from Project History: A Case Study for Software Development”. *Proc. 2004 ACM conference on Computer-supported cooperative work*. ACM Press. 2004, pp. 82-91.
- [11] EASEL website, <http://www.isr.uci.edu/projects/easel/index.html>, visited on September 16, 2006
- [12] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley Professional; 1st edition (January 15, 1995)
- [13] Graphical Modeling Framework website, <http://www.eclipse.org/gmf/>, visited on September 11, 2006
- [14] IBM, Rational Software Architect website, <http://www-306.ibm.com/software/awdtools/architect/swarchitect/>, visited on September 09 2006.
- [15] IBM’s website, <http://www-306.ibm.com/software/awdtools/modeler/swmodeler/index.html>, visited on May 8, 2006.

References

- [16] IBM Rational Software products website, <http://www-306.ibm.com/software/rational/offerings/design.html>, visited on September 11, 2006
- [17] IBM Rational RequisitePro website, <http://www-306.ibm.com/software/awdtools/reqpro/>, visited on September 12, 2006
- [18] IBM Software, Rational ClearCase website, <http://www-306.ibm.com/software/awdtools/clearcase/>, visited on September 11, 2006
- [19] Jintae Lee, “SIBYL: a tool for managing group design rationale”. *Proc. 1990 ACM conference on Computer-supported cooperative work*. ACM Press. 1990, pp. 79-92.
- [20] Mary Beth Rosson, Susanne Maass, and Wendy A. Kellogg. “The Designer as User: Building Requirements for Design Tools from Design Practise.” *Communications of the ACM*. 31, 11, ACM Press. 1988, pp. 1288-1298.
- [21] Scott A. Hendrickson, André van der Hoek, “Modeling Product Line Architectures through Change Sets”. Technical report.
- [22] Timothy C. Lethbridge, Robert Laganière, *Object Oriented Software Engineering: Practical Software Development Using UML and Java*, 2nd Edition, 2004, McGraw Hill, London.
- [23] William Harrison, Harold Ossher and Peri Tarr. “Software Engineering Tools and Environments: A Roadmap”, *ICSE 2000, Proc. Conference on The Future of Software Engineering*. ACM Press. 2000, pp. 261-277.

Appendix 1 – Software systems descriptions and designs

A1.1 Elections Management System

The Ootumlia Elections Commission is designing a system to manage elections. The system will manage elections for a variety of different elected bodies (e.g. school boards, city councils etc.). Each elected body can have various positions (also called seats, e.g. mayor, councilor etc.). Elections are scheduled for a specific date, and usually several (or all) positions are voted on together; however, sometimes there may be the need for a by-election (e.g. to elect a particular councilor because the incumbent - the previous person who held the position - has resigned).

The system will keep track of candidates for each seat. The system will also record who is the incumbent for a seat, since newspaper reporters are interested in reporting whether incumbents have won again or lost. The system records the name and address of each candidate and incumbent.

The system will also keep track of the list of eligible voters. Each voter can only vote for certain positions (e.g. a particular council seat that represents their area). Each voter is also assigned to vote at a specific poll - each poll has a number and is located in a polling station. The system records the name and address of each voter.

Finally, the system will keep track of the number of votes for each candidate at each poll. However, under no circumstance will it record which voter voted for which candidate, nor whether a voter voted at all.

Figures 38 and 39 show the two design solutions for the problem above.

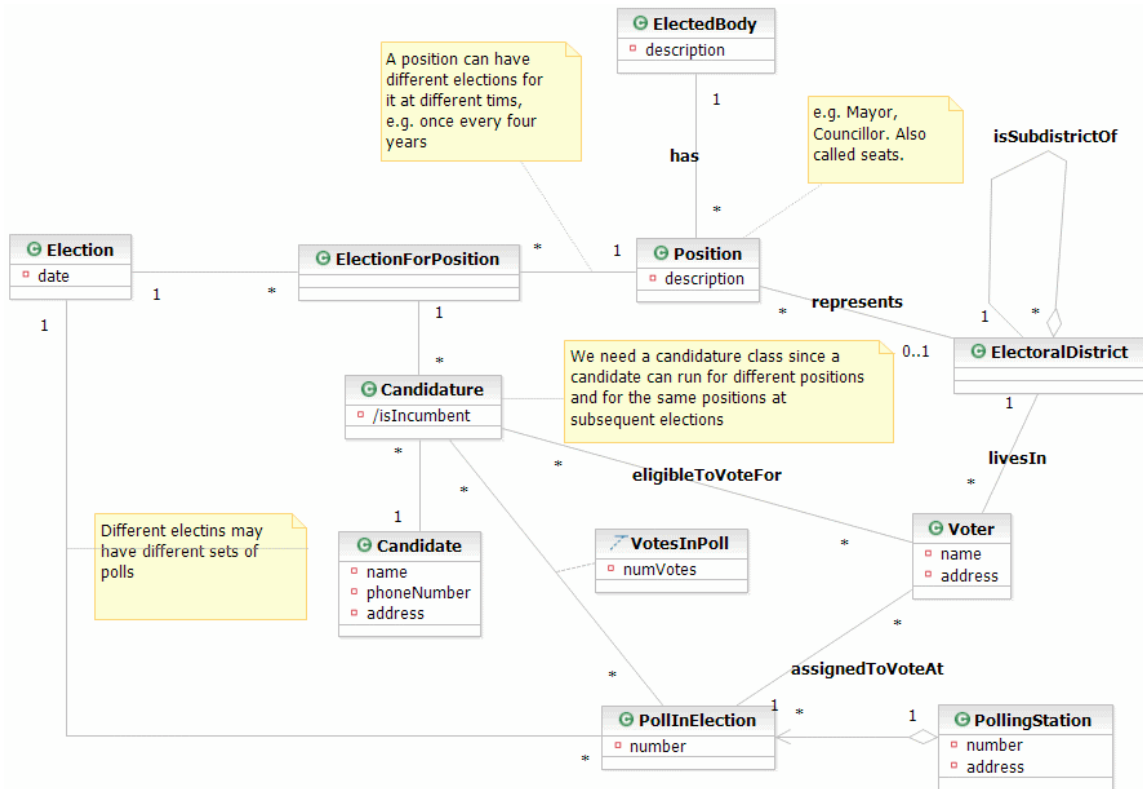


Figure 38 - Elections system, design 1

Participants who were given the Elections system diagram in Figure 38 were asked to answer the following questions:

1. Each voter needs a voter number. This number will be printed on a registration card that they take to the polling station. This should be added as an attribute in which class?
2. We need to add a derived attribute showing the total number of votes a candidate got in the election. Which class should this go in?
3. We want to change the model to record whether a voter has voted, but without recording who the voter voted for. The idea is to stop the voter voting twice. Explain to the experimenter how you would edit the model to incorporate this feature.

Appendix 1 – Software systems descriptions and designs

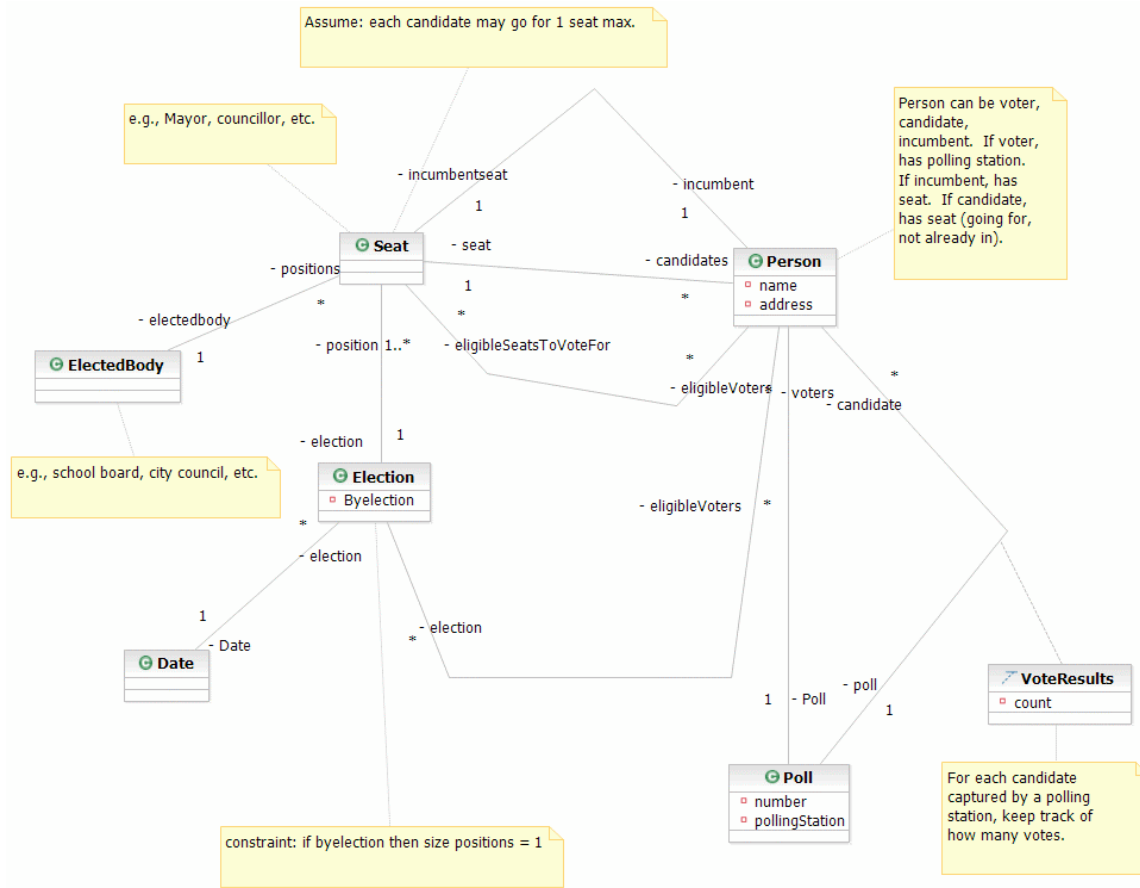


Figure 39 - Elections system, design 2

Participants who were given the Elections system diagram in Figure 39 were asked to answer the following questions:

1. Each voter needs a voter number. This number will be printed on a registration card that they take to the polling station. This should be added as an attribute in which class?

2. In which class would I put a derived attribute 'totalVotes' to indicate the total number of votes a candidate received in the election?

3. Class Person does double duty representing a Voter and a Candidate. However, associations in the model currently state that a voter is

- a) always an incumbent in a seat,
- b) always a candidate in a seat, and
- c) always has results in a poll.

Describe to the experimenter what you would do to fix these three problems. Hint: The solution to each problem is similar.

A1.2 Investments System for OOBank

OOBank has a separate investment division. This division manages various mutual funds in which investors may invest and also looks after the investment portfolios of investors.

An investor may at any point in time have several investment advisors. These help the investor decide in what to invest. Different investment advisors specialize in different types of investments.

Investors make a series of transactions and may have to pay a commission on each transaction. The commission is paid to the investment advisor that arranged the transaction.

For each investment the system must keep track of the number of shares (also called units) in addition to the amount the investment is worth today and the amount originally invested.

Each mutual fund invests its money in various securities. The securities can be stocks, bonds or other mutual funds. We must be able to calculate the original amount invested in each security as well as how much that investment is worth today. Each mutual fund may have several investment advisors that help the fund decide what securities in which to invest.

The mutual funds in which investors invest may be managed by OOBank or by some other company. Each mutual fund company may manage several mutual funds.

Figures 40 and 41 show the two design solutions for the problem above.

Appendix 1 – Software systems descriptions and designs

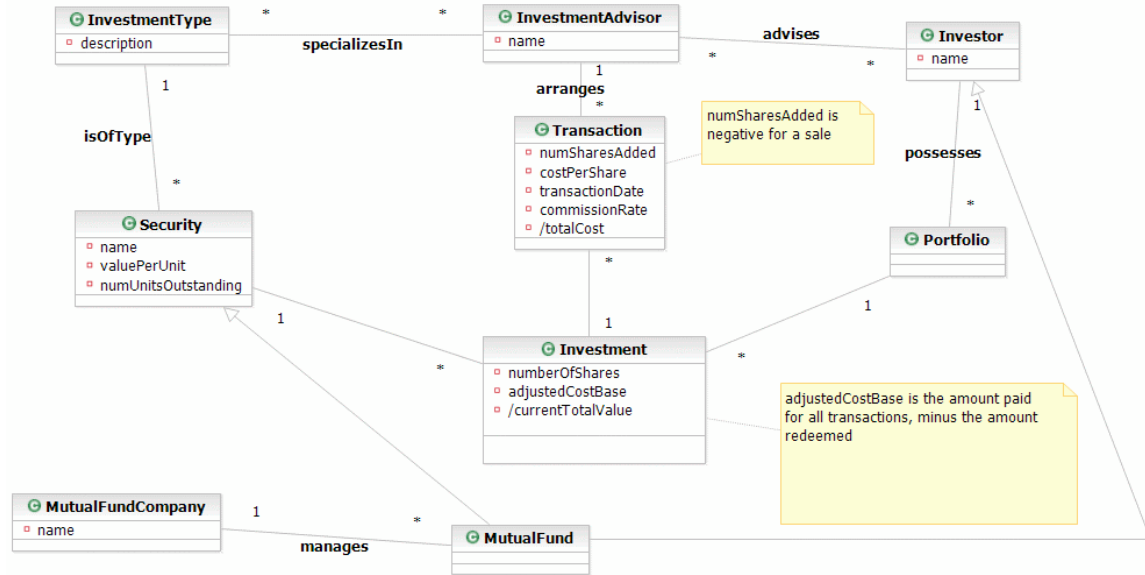


Figure 40 - Investment system, design 1

Participants who were given the Investment system diagram in Figure 40 were asked to answer the following questions:

1. Mutual funds have an alphanumeric code used to uniquely identify them. In which class should I add an attribute fundCode?

2. Some investment transactions, such as payment of a dividend, do not involve an investment advisor at all. Currently, however the diagram requires that all transactions do involve an advisor. What change to the diagram should you make to fix this?

3. Explain to the experimenter how you would modify the diagram to allow an investor to invest in stocks. The new Stock class should have attribute 'dividendRate'.

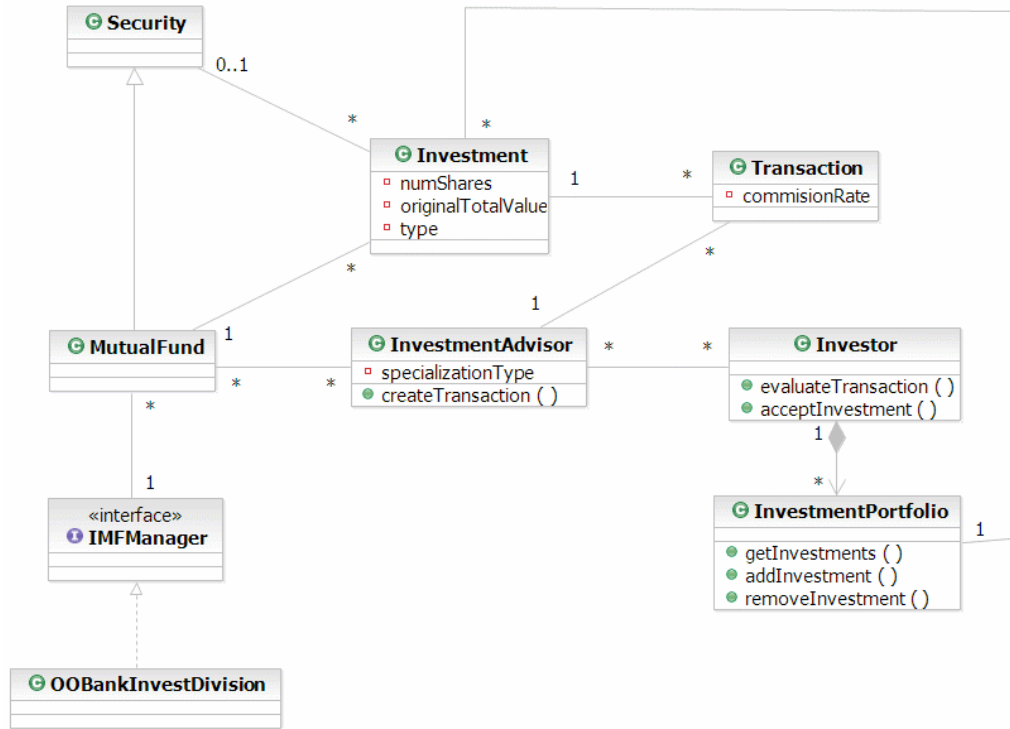


Figure 41 - Investment system, design 2

Participants who were given the Investment system diagram in Figure 41 were asked to answer the following questions:

1. Mutual funds have an alphanumeric code used to uniquely identify them. In which class should I add an attribute fundCode?
2. A transaction must specify an amount of shares bought or sold. What change or changes to the diagram should you make to fix this?
3. Currently the single instance of OOBankInvestDivision can take on the role of IMFManager to manage a mutual fund. Imagine an InvestmentAdvisor can also take on this role. Explain to the experimenter how you would modify the diagram to this.

A1.3 Airline system

The Basic Airline Reservation System will enable a new airline to quickly configure its flights and start taking bookings. The airline will fly a number of routes, each broken down into legs, where a leg involves flying from one airport to another. Airports are identified by three-letter codes, such as LAX for Los Angeles International Airport, or YOW for Ottawa International Airport.

Appendix 1 – Software systems descriptions and designs

Once a set of routes are established, the airline schedules flights on those routes. There may be more than one flight a day on the same route. Each scheduled flight is given a flight number, which is reused from day to day. When defining a scheduled flight on a given route, the departure and arrival time must be defined for each leg. Finally each daily departure of a given scheduled flight must be set up. The actual and expected departure and arrival times for these are changed in real time as data becomes available.

Crew members must be defined for each flight leg. The system tracks the job title, employee number, name and address of each crew member, as well as who is their supervisor.

Finally, passengers are added to the system. A passenger has a name, and may have a frequent flier number, emergency contact number, home address and passport number. A passenger can be booked on a set of flight legs. Each booking has a class (e.g. economy, business), a fare, and an assigned seat.

Figure 42 shows the design solution for the problem above.

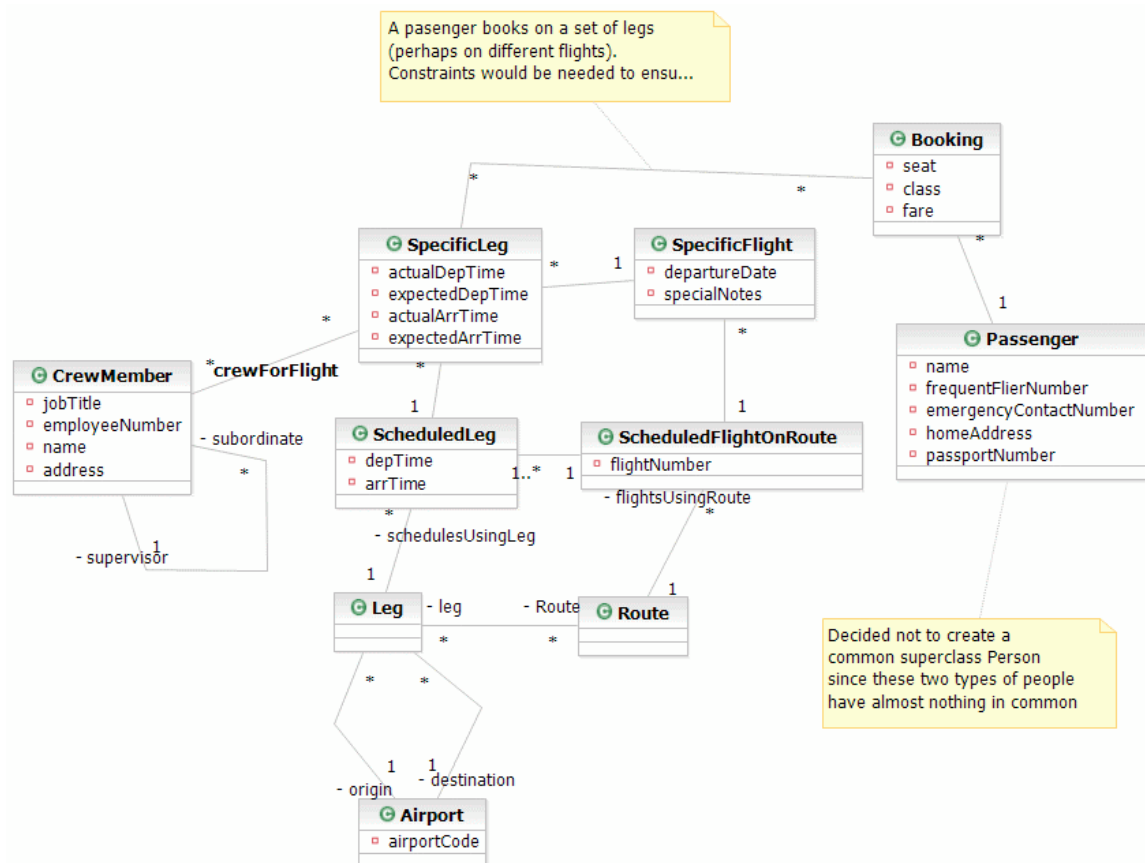


Figure 42 - Airline system, design

Appendix 1 – Software systems descriptions and designs

Participants who were given the Airline system diagram in Figure 42 were asked to answer the following questions:

1. The model doesn't currently keep track of the name of the cities where flights go. In which class should I add an attribute cityName?

2. Imagine the airline regularly flies from Ottawa to Toronto, but wants to create an extra charter flight tomorrow from Ottawa to Toronto. To enable this, the system will have to create instances of one or more classes. Which classes will the system have to make instances of? No need to worry about booking any passengers or assigning crew.

3. Explain to the experimenter how you would modify the diagram to allow several bookings to be grouped (e.g. when family members fly together).

Appendix 2 – Recruitment text

The following email was sent to prospective participants for Step 1. When sent to students, the wording about managers was not used.

Hi,

My name is Hanna Farah and I am an IBM CAS Student. I am doing my masters degree at the University of Ottawa under the supervision of Timothy Lethbridge.

I am looking for volunteers to participate in a research project. I need a couple of people to build UML models using a special version of Rational Software Modeler. The models you create would then, anonymously be used in another step of my research, in which we will ask participants to try to understand them.

If you agree to participate, the total time requirement would be about an hour and a half, at a time convenient for you. It could also be broken into two 45-minute sessions.

If you are willing to participate, would you simply reply to this email (or drop by my desk at IBM). Participation is strictly voluntary. Your name was one of the names suggested by IBM managers, but they do not require you to participate.

Thanks

Hanna

Appendix 2 – Recruitment text

The following email was sent to prospective participants for Steps 3 and 4

Hi,

My name is Hanna Farah and I am an IBM CAS Student. I am doing my masters degree at the University of Ottawa under the supervision of Timothy Lethbridge.

I am looking for volunteers to participate in a research project. I need several people to participate in an experiment in which you would try to answer some questions about a class diagram. This would be done using a version of Rational Software Modeler in which certain special features may have been activated. The purpose of the experiment is to evaluate the features.

If you agree to participate, the total time requirement would be about 55 minutes, at a time convenient for you.

If you are willing to participate, would you simply reply to this email (or drop by my desk at IBM). Participation is strictly voluntary. Your name was one of the names suggested by IBM managers, but they do not require you to participate.

Thanks

Hanna

Appendix 3 – Informed consent, step 1

I, (name of research subject) _____, **accept to participate in a University of Ottawa research project entitled “Applying Cognitive Patterns to Software Tool Development”**. The student performing the research, Hanna Farah, will use the results as part of his Master’s thesis. The research supervisor is Dr. Timothy C. Lethbridge, of the School of Information Technology and Engineering.

The purpose of the research is to help improve certain features of software modeling tools. Specifically, we are interested in a feature that allows a user to look at the history of development of a UML class diagram in order to understand it.

In the step of the work in which I am participating involves performing two modeling tasks. The models I create, including the steps I used (the order in which I added classes and associations, for example) will then be used in later experiments with other users. However, nobody will know that it was I that created the models.

More specifically, my participation will consist of the following:

- To use a special version of Rational Software Modeler (RSM) to create two class diagrams. I will be given some simple requirements on which to base the diagrams. The version of RSM has been instrumented to record the steps I use as I create the model.
- After completing each model, I will review it with the researcher and possibly make some changes to the model.

It is anticipated that developing each model will take about 45 minutes, resulting in a total of about an hour and a half of participation.

There are no known risks to this activity, however I understand that **I have the right to stop participating at any time**. In such a case, a partial models I have built will not be used.

Appendix 3 – Informed consent, step 1

I understand that participation is strictly voluntary. If I am a student, my grades and academic standing will in no way be affected by my participation as a research subject, or my choice not to participate.

I have received assurance from the researchers that **the models I create during the session will remain strictly anonymous.** Anonymity will be assured because my name will not be recorded anywhere.

Any information requests or complaints about the ethical conduct of the project may be addressed to the Protocol Officer for Ethics in Research, +1 613 562-5800 ext. 1787.

There are **two copies** of this consent form, one of which I may keep.

If I have any questions, I may contact the student at +1 612 262-4567 (email drhif@yahoo.com) or Dr. Lethbridge at +1 613 562-5800 ext. 6685 (email tcl@site.uottawa.ca).

Research Subject's signature _____ Date _____

Researcher signature _____ Date _____

I wish to receive a summary of findings of this research when available:

Yes ___ No ___

If I wish to receive a summary of findings of this research, then I can be reached at

Appendix 4 – Informed consent step 3,4

I, (name of research subject) _____, **accept to participate in a University of Ottawa research project entitled “Applying Cognitive Patterns to Software Tool Development”**. The student performing the research, Hanna Farah, will use the results as part of his Master’s thesis. The research supervisor is Dr. Timothy C. Lethbridge, of the School of Information Technology and Engineering.

The purpose of the research is to help improve certain features of software modeling tools. Specifically, we are interested in a feature that allows a user to look at the history of development of a UML class diagram in order to understand it.

In the step of the work in which I am participating involves attempting to answer some questions about UML class diagrams. The class diagrams have been created anonymously by other participants.

More specifically, my participation will consist of the following:

- To use a special version of Rational Software Modeler (RSM) to study three class diagrams and answer five questions about each of them. Various features will be available to me, and the researcher will explain these features.
- After completing the above, I will be asked some general questions about my experiences.

It is anticipated that my participation will take about 55 minutes in total.

The time it takes me to answer the questions about the class diagrams will be measured. However, **I understand that it is not me that is being evaluated**; instead it is the software tool.

Appendix 4 – Informed consent, step 3,4

There are no known risks to this activity, however I understand that **I have the right to stop participating at any time**. In such a case, the data gathered from my participation will not be used.

I understand that participation is strictly voluntary. If I am a student, my grades and academic standing will in no way be affected by my participation as a research subject, or my choice not to participate.

I have received assurance from the researchers that **the data arising from my participation will remain strictly confidential**. Anonymity will be assured because my name will not be recorded anywhere.

Any information requests or complaints about the ethical conduct of the project may be addressed to the Protocol Officer for Ethics in Research, +1 613 562-5800 ext. 1787.

There are **two copies** of this consent form, one of which I may keep.

If I have any questions, I may contact the student at +1 612 262-4567 (email drhif@yahoo.com) or Dr. Lethbridge at +1 613 562-5800 ext. 6685 (email tcl@site.uottawa.ca).

Research Subject's signature _____ Date _____

Researcher signature _____ Date _____

I wish to receive a summary of findings of this research when available:

Yes ___ No ___

If I wish to receive a summary of findings of this research, then I can be reached at

Appendix 5 – Preference questionnaire

For Q1-Q5 of the following, *please circle* whether you strongly agree, agree, are neutral, disagree or strongly disagree with the statement:

Q1: *I found that the TMEP feature (the ability to explore the history of development of a diagram) helped me understand class diagrams more quickly.*

Strongly agree agree neutral disagree strongly disagree

Q2: *When exploring the history of a model using the page-up and page-down keys, I found that a **useful set of steps** in the development of the model (snapshots) were presented. In other words the increments with which the development of the model was revealed were neither too small nor too large.*

Strongly agree agree neutral disagree strongly disagree

Q3: *When exploring the history of a model using TMEP, I preferred when the classes did not move. In other words, they were shown in their final position, even though the modeler may have moved them.*

Strongly agree agree neutral disagree strongly disagree

Q4: *I would use the TMEP feature **if it was available to me in my work environment and I was asked to understand a class diagram.***

Strongly agree agree neutral disagree strongly disagree

Q5: *When looking back at the earliest stages of a model's development with the TMEP feature, the most important classes appeared first, and the less important classes appeared later.*

Strongly agree agree neutral disagree strongly disagree

Q6: *Overall, I found that using the TMEP feature to step through the changes **resulted in me taking a longer time to answer the questions** presented, than if I had just looked at the final diagram. In other words, TMEP didn't save me time.*

Strongly agree agree neutral disagree strongly disagree

Q7: *The TMEP feature was **awkward to use.***

Strongly agree agree neutral disagree strongly disagree

Q8: *My expertise in UML is:*

Very high high medium low very low

Q9: *I create class diagrams:*

Appendix 6 – Raw and normalized data from user study

Every day being educated every week every month occasionally only when I was being educated

Q10: *I have to try to understand class diagrams:*

Every day being educated every week every month occasionally only when I was being educated

Q11: *What aspects of the TME feature did you **most like**?*

Q12: *What aspects of the TME feature could be **improved**?*

Appendix 6 – Raw and normalized data from user study

A6.1 Preference questions

Preference questions are ranked between 1 and 5 (1:strongly disagree – 5:strongly agree), refer to Appendix 5 for more details about the questions.

Participant ↓	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10
A	4	4	4	3	4	2	1	4	2	2
B	4	4	4	3	4	3	2	3	2	3
C	4	2	3	3	3	3	2	5	5	4
D	4	5	5	4	3	2	2	3	3	2
E	4	5	5	4	5	2	2	5	4	2
F	4	4	4	4	3	3	1	2	2	2
G	4	5	1	5	1	3	1	2	2	2
H	4	4	1	4	3	2	2	3	4	4
I	5	4	4	5	3	2	1	3	2	2
J	5	4	4	5	4	2	1	3	2	2
K	4	4	5	3	4	3	2	2	2	2
L	4	4	5	4	4	2	3	2	2	2
Max	5	5	5	5	5	3	3	5	5	4
Min	4	2	1	3	1	2	1	2	2	2
Median	4	4	4	4	3.5	2	2	3	2	2
Mean	4.2	4.1	3.8	3.9	3.4	2.4	1.7	3.1	2.7	2.4
StDev	0.4	0.8	1.4	0.8	1.0	0.5	0.7	1.1	1.1	0.8
.95 Confidence interval +/-	0.22	0.45	0.80	0.45	0.56	0.29	0.37	0.61	0.61	0.45

Table 22 - Answers to preference questions

A6.2 Timings

Table 23 shows the normalized (by model and by participant) performance results for speed and accuracy for the twelve participants in our study according to the first timing strategy. The letters A to L represent the participants.

	Accuracy				Speed			
	Only T1	T2 & T3	Only T2	Only T3	Only T1	T1 & T2	Only T2	Only T3
	no TMEP	TMEP	TMEP orig	TMPE final	no TMEP	TMEP	TMEP orig	TMEP final
A	0.88	1.06	1.15	0.96	0.96	1.02	0.86	1.18
B	1.12	0.94	1.09	0.79	1.03	0.98	1.12	0.85
C	1.07	0.97	0.85	1.08	1.20	0.90	0.91	0.88
D	1.21	0.90	0.80	1.00	0.90	1.05	1.20	0.91
E	1.01	1.00	1.02	0.97	0.57	1.21	1.34	1.08
F	0.94	1.03	0.97	1.08	1.13	0.94	0.95	0.92
G	1.14	0.93	0.84	1.02	0.73	1.13	1.51	0.76
H	0.74	1.13	1.26	1.00	1.06	0.97	1.01	0.93
I	1.18	0.91	0.98	0.84	1.16	0.92	0.92	0.92
J	1.06	0.97	0.99	0.95	1.45	0.77	0.58	0.97
K	0.85	1.07	1.12	1.03	0.99	1.01	0.80	1.21
L	0.85	1.07	1.09	1.06	0.73	1.14	1.23	1.05
Max	1.21	1.13	1.26	1.08	1.45	1.21	1.51	1.21
Min	0.74	0.90	0.80	0.79	0.57	0.77	0.58	0.76
Median	1.03	0.98	1.01	1.00	1.01	0.99	0.98	0.93
Mean	1.00	1.00	1.01	0.98	0.99	1.00	1.04	0.97
StDev	0.15	0.07	0.14	0.09	0.24	0.12	0.26	0.14
.95 Confidence interval +/-	0.08	0.04	0.08	0.05	0.14	0.07	0.14	0.08
.90 Confidence interval	0.07	0.04	0.07	0.04	0.11	0.06	0.12	0.06

Table 23 - Performance results, timing 1

Appendix 6 – Raw and normalized data from user study

Table 24 shows the same results according to the second timing strategy:

	Accuracy				Speed			
	Only T1	T2 & T3	Only T2	Only T3	Only T1	T1 & T2	Only T2	Only T3
	no TMEP	TMEP	TMEP orig	TMPE final	no TMEP	TMEP	TMEP orig	TMEP final
A	0.88	1.06	1.15	0.96	0.86	1.07	0.84	1.30
B	1.12	0.94	1.09	0.79	1.20	0.90	1.12	0.68
C	1.07	0.97	0.85	1.08	1.35	0.83	0.81	0.84
D	1.21	0.90	0.80	1.00	0.85	1.07	1.24	0.91
E	1.01	1.00	1.02	0.97	0.47	1.26	1.39	1.14
F	0.94	1.03	0.97	1.08	1.15	0.92	0.98	0.87
G	1.14	0.93	0.84	1.02	0.61	1.20	1.75	0.64
H	0.74	1.13	1.26	1.00	1.06	0.97	0.97	0.96
I	1.18	0.91	0.98	0.84	1.28	0.86	1.00	0.73
J	1.06	0.97	0.99	0.95	1.54	0.73	0.45	1.01
K	0.85	1.07	1.12	1.03	1.06	0.97	0.79	1.15
L	0.85	1.07	1.09	1.06	0.71	1.14	1.20	1.09
Max	1.21	1.13	1.26	1.08	1.54	1.26	1.75	1.30
Min	0.74	0.90	0.80	0.79	0.47	0.73	0.45	0.64
Median	1.03	0.98	1.01	1.00	1.06	0.97	0.99	0.93
Mean	1.00	1.00	1.01	0.98	1.01	0.99	1.04	0.94
StDev	0.15	0.07	0.14	0.09	0.32	0.16	0.33	0.20
.95 Confidence interval +/-	0.08	0.04	0.08	0.05	0.18	0.09	0.19	0.12
.90 Confidence interval	0.07	0.04	0.07	0.04	0.15	0.08	0.16	0.10

Table 24 - Performance results, timing 2

Appendix 7 – Experiment data forms

A7.1 Participant steps for Treatment pattern 1 t23 and 1 t32

Subject letter and initials _____ Date _____ Treatment pattern _____

0. Make sure the experiment is set up properly well before the participant arrives.

1. Welcoming the participant: Explain general purpose of the experiment and have the participant sign the informed consent form.

2. First diagram (No TMEP). Show them their first diagram _____.

Record the start time _____

Ask them to generally try to understand the model for 2-3 minutes, and to tell you when done.

Notes about interesting things he/she did

Record the time after basic understanding _____

Give them the problem sheet for that diagram. Ask them to answer the questions by looking at the diagram.

Time done Q1 _____

Time done Q2 _____

Time done Q3 _____ Evaluation of correctness _____

Record their general comments about the diagram.

3. Training: Show them TMEP in the University system. Show them the operation of page down, page up, home and end, and have them *walk through the system* to understand how TMEP operates.

Ask them if they understand how TMEP operates. Continue explaining if they seem unsure.

Continued on next page

Appendix 7 – Experiment data forms

4. **Second diagram (TMEP):** Show them their correct second diagram _____ that should be blank since it is TMEP in home position.

Record the start time _____

Ask them to generally try to understand the model by stepping through the time sequence, and looking at the final model. Ask them to tell you when done.

Notes about interesting things he/she did

Record the time after basic understanding _____

Give them the problem sheet for that diagram. Ask them to answer the questions by looking at the diagram, and using TMEP to go back if they find it helpful.

Time done Q1 _____

Time done Q2 _____

Time done Q3 _____ Evaluation of correctness _____

Record their general comments about the diagram

5. **Third diagram.** Repeat of 4 for the correct third diagram _____.

Start time _____

Notes about interesting things he/she did

Time after basic understanding _____

Time done Q1 _____

Time done Q2 _____

Time done Q3 _____ Evaluation of correctness _____

Record their general comments about the diagram:

6. **Preferences:** Ask the participant to complete the preference questions, and thank them.

A7.2 Participant steps for Treatment pattern 23 t1 and 32 t1

Subject letter and initials _____ **Date** _____ **Treatment pattern** _____

0. Make sure the experiment is set up properly well before the participant arrives.

1. Welcoming the participant: Explain general purpose of the experiment and have the participant sign the informed consent form.

2. Training: Show them TMEP in the University system. Show them the operation of page down, page up, home and end, and have them *walk through the system* to understand how TMEP operates.

Ask them if they understand how TMEP operates. Continue explaining if they seem unsure.

3. First diagram: Show them their correct first diagram _____ that should be blank since it is TMEP in home position.

Record the start time _____

Ask them to generally try to understand the model by stepping through the time sequence, and looking at the final model. Ask them to tell you when done.

Notes about interesting things he/she did

Record the time after basic understanding _____

Give them the problem sheet for that diagram. Ask them to answer the questions by looking at the diagram, and using TMEP to go back if they find it helpful.

Time done Q1 _____

Time done Q2 _____

Time done Q3 _____ Evaluation of correctness _____

Record their general comments about the diagram:

Continued on next page

Appendix 7 – Experiment data forms

4. **Second diagram.** Repeat of 3 for the correct second diagram _____.

Start time _____

Notes about interesting things he/she did

Time after basic understanding _____

Time done Q1 _____

Time done Q2 _____

Time done Q3 _____ Evaluation of correctness _____

Record their general comments about the diagram:

5. **Third diagram.** Show them their third diagram _____. Explain that TMEP will now **not** be available.

Record the start time _____

Ask them to generally try to understand the model for 2-3 minutes, and to tell you when done.

Notes about interesting things he/she did

Record the time after basic understanding _____

Give them the problem sheet for that diagram. Ask them to answer the questions by looking at the diagram.

Time done Q1 _____

Time done Q2 _____

Time done Q3 _____ Evaluation of correctness _____

Record their general comments about the diagram:

6. **Preferences:** Ask the participant to complete the preference questions, and thank them.