A MASTER THESIS ON PORTING THE ENTERPRISE ARCHITECTURE ANALYSIS TOOL TO ECLIPSE MODELING PROJECT

Stanislav Ivanov

Degree project in
ICS
Masterthesis,
Stockholm, Sweden 2011

XR-EE-ICS 2011:018
A MASTER THESIS ON PORTING THE ENTERPRISE ARCHITECTURE ANALYSIS TOOL TO ECLIPSE MODELING PROJECT

Stanislav Ivanov

A Master Thesis Report written in collaboration with Department of Industrial Information and Control Systems Royal Institute of Technology Stockholm, Sweden

August, 2011
## List of abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EAT</td>
<td>Enterprise Architecture Analysis Tool</td>
</tr>
<tr>
<td>AM</td>
<td>Abstract Modeler</td>
</tr>
<tr>
<td>CM</td>
<td>Concrete Modeler</td>
</tr>
<tr>
<td>EAM</td>
<td>Eclipse Abstract Modeler</td>
</tr>
<tr>
<td>ECM</td>
<td>Eclipse Concrete Modeler</td>
</tr>
<tr>
<td>EMP</td>
<td>Eclipse Modeling Project</td>
</tr>
<tr>
<td>IDE</td>
<td>Integrated Development Environment</td>
</tr>
<tr>
<td>EMF</td>
<td>Eclipse Modeling Project</td>
</tr>
<tr>
<td>DSL</td>
<td>Domain Specific Language</td>
</tr>
<tr>
<td>PRM</td>
<td>Probabilistic Relationship Model</td>
</tr>
<tr>
<td>XML</td>
<td>Extensible Markup Language</td>
</tr>
<tr>
<td>UML</td>
<td>Unified Modeling Language</td>
</tr>
<tr>
<td>GUI</td>
<td>Graphical User Interface</td>
</tr>
<tr>
<td>MVC</td>
<td>Model View Controller</td>
</tr>
<tr>
<td>XMI</td>
<td>XML Metadata Interchange</td>
</tr>
<tr>
<td>GMF</td>
<td>Graphical Modeling Framework</td>
</tr>
<tr>
<td>EEF</td>
<td>Extended Editing Framework</td>
</tr>
<tr>
<td>UI</td>
<td>User Interface</td>
</tr>
<tr>
<td>RCP</td>
<td>Rich Client Platform</td>
</tr>
</tbody>
</table>
**Abstract.** This master thesis is a part of the ongoing research on EAT development project. Its main goal is to research whether Eclipse Modeling Project can be used as an alternative platform to using NetBeans in implementing EAT tool. In order to fulfill this goal, it contains analysis of the current EAT tool version and design research of a new version using EMP. The design addresses most of the issues related to building a new version and eventually recommends porting EAT to EMP.

**Keywords.** EAT. EMP. NetBeans. modeling.
1. Introduction

1.1. Foreword

This master thesis is a part of the ongoing research on Enterprise Architecture Assessment tool (EAT) development project. More specifically, it deals with an alternative implementation of that tool using Eclipse Modeling Project (EMP).

1.2. Abbreviations in this report

This master thesis report contains a lot of abbreviations related to EAT tool's current version, the version under research, as well as technologies' and frameworks' names. They are organized in the "List of abbreviations" and it should be used as a reference throughout this document. The first five abbreviations are related to the EAT tool in its current version and the version under research. The rest are standard abbreviations for technologies' and frameworks' names.

1.3. EAT tool

Enterprise Architecture Assessment tool (abbreviated as “EAT tool”, “EA2T tool” or just EAT) is an ongoing development project at the Department of Industrial Information and Control Systems (ICS) at the Royal Institute of Technology (KTH), Sweden. EAT tool is inspired by graphical decision-theoretic methods, such as Bayesian networks and influence diagrams. EAT is a versatile and user-friendly environment for modeling and analysis of enterprises and their information systems. The modules, developed at ICS at KTH in Sweden, are publicly available since September 2008.

The EAT tool consists of two parts, the Abstract Modeler(AM) for defining the underlying theory for the assessment and the Concrete Modeler(CM) for modeling enterprises and performing assessments.

1.4. Development platform of the current version of EAT tool

Currently, EAT tool is implemented using NetBeans IDE [2] as its platform. Prior to 2010, NetBeans was a product owned and developed by Sun Microsystems. On 20 April 2009, it was announced that Oracle Corporation plans to acquire Sun Microsystems [3]. This immediately reacted in speculations about the future of NetBeans as Oracle already has its own IDE for developing Java applications namely JDeveloper and in the same time is contributing to Eclipse. John O'Conner, a software architect, consultant, author, and speaker who worked many years at Sun Microsystems expressed his speculations about the future in his article [4] on the next day after the announcement of the acquisition: “Like many open source projects, NetBeans gets a lot of support from corporate interest. In this case, it's no secret that Sun pours cash into NetBeans. However, Oracle already supports an IDE. Let's make that two IDEs: JDeveloper and Eclipse. Both receive financial backing from Oracle already, and...well, my first thought is that Oracle simply won't need a third IDE.” Similar opinions can be seen as comments in another article [5] discussing the future of Sun products.

On 27 January 2010, Oracle finalized the acquisition of NetBeans' owner Sun Microsystems [6]. At first, Oracle announced that they would continue to support and enhance NetBeans [7] together with the other two IDEs – JDeveloper and Eclipse, this way targeting to fulfill the needs of different developers. However, article [8] makes an independent and extensive analysis of the future of the technology. It concludes that “whatever happens with NetBeans in the future, one thing can be sure: only time will tell whether Oracle will be able to fulfill its promise of developing three IDEs and two community web portals simultaneously, or whether NetBeans – arguably at the bottom of Oracles priority list – will be left behind in Oracle's three-way IDE race.” Similar independent opinions are seen in the community's blogs [9]: “Smells like they[Oracle] want to have three IDE’s in order to take everybody to them, no matter what you like: NetBeans, Eclipse or JDeveloper. So far, idea is good, as long as they will not move all the good parts from NetBeans to JDeveloper, then lock on pay and then abandon NetBeans entirely, making it unusable.” As the integration of parts of NetBeans into JDeveloper has already been announced [7, 8], such scenario should be taken seriously into consideration when using NetBeans.

The future of NetBeans is questionable in another aspect – how the IDE will evolve. So far, the plans are general and vague. For example, Thomas Kurian, Executive Vice President at Oracle, said that "Oracle plans to focus NetBeans on PHP, Python and other dynamic scripting languages, while pushing JDeveloper IDE as its strategic development environment". [8] That brings a question about the reason behind Oracle's decision to “give” NetBeans these particular languages. Dana Gardner, principal analyst at Interarbor Solutions thinks so: "There really is no need to do scripting in a Java IDE. This might just be a way of sun-setting NetBeans." [8]

Current EAT implementation is already heavily dependent on NetBeans as it makes extensive use of its Visual library. On one hand, the future evolution and existence of NetBeans is questionable and unclear.
On the other hand, the research on EAT is continuing. Consequently, researching potential alternative implementations using other IDEs and platforms is recommendable.

1.5. Eclipse

Four critical factors prove together the importance of Eclipse (and EMP) in this case when an alternative implementation of EAT tool is researched.

1. Article [10] makes a brief overview of currently available Java IDEs with their advantages and disadvantages. As we can see from it, Eclipse is placed first, NetBeans coming second on the list. Eclipse has much more advantages over NetBeans.

2. For years now Eclipse has been considered NetBeans’ main rival. This claim is supported by the numerous citations [11, 12, etc.] and articles and discussions about that [13, 14, etc.].

3. When an alternative implementation using other IDE is considered, it is crucial that the alternative should provide if not better, at least similar functionality as the one used so far. Eclipse provides Eclipse Modeling Project (EMP), which “focuses on the evolution and promotion of model-based development technologies within the Eclipse community by providing a unified set of modeling frameworks, tooling, and standards implementations.”[15] We already mentioned that EAT is a “versatile and user-friendly environment for modeling”[1]. The similarity of the functionality provided by EAT and EMP should thus be further considered and researched.

4. EMP is used already, specifically one part of it - Eclipse Modeling Framework (EMF), by the researchers of the EAT tool. [16]

All of the factors above show the necessity of research about the modeling capabilities of Eclipse Modeling Project. Consequently, Eclipse (and EMP respectively) will be used as a platform for the next version of EAT tool.
2. Scope, objective and delimitations

2.1. Objective
This master thesis investigates whether Eclipse Modeling Project (incl. EMF, ECore, DSL, etc.) can be used as an alternative platform to using NetBeans in implementing EAT tool. Either if it can or it cannot, this must be proved. If it can be used, best case scenario would be to develop an Eclipse-based plug-in version of EAT tool. This new version should provide the same functionality as the existing one now. It should contain the graphical modeler plus an extension of ECore to represent PRM formalism. All the steps needed to implement this should be thoroughly documented. However, if Eclipse Modeling Project cannot be used because it does not provide the needed functionality, this should be proved and documented thoroughly. Put this way, the objective differs slightly only in the first sentence in comparison to the originally set objective of this thesis. Thus it becomes more precise.

2.2. Scope
The things that are included in the scope of this thesis are listed here. They follow the overall requirements for carrying out a master thesis at ICS, KTH:
1. Giving brief introduction to EAT tool, Eclipse and its EMP and also the reasoning behind the need of porting the EAT to Eclipse - Introduction
2. Choosing a research method and justifying this choice by comparing it to other available methods – Method & Theory
3. Presenting the results of the design research in Iteration 1 and 2 – Analysis & Data
4. Discussion on the non-functional aspects of the systems after both Iteration 1 and 2 - Analysis and Discussion
5. Conclusions and suggestions for future research - Conclusions

2.3. Delimitations

Formal:
- The original duration of a master thesis is estimated to 20 weeks in accordance with the requirements of KTH.
- This master thesis officially started on 7 January 2011, but that whole week is counted as the first of the work on the thesis. The research and study should have been executed by 6 May according to the set schedule. The time after that (two more weeks thus counting up to 20) was left for writing this report and its check.
- Writing of this report started a week later from the original schedule due to the need to finish the research on the topic. Writing of the final report will be extended to one week before the final presentation scheduled for the 30 August 2011.
- All the research and analysis are presented in this report. However, there is one more document describing in technical details the built artifacts. This document is required by ICS as a proof of the overall work done by the author.

Implementation related:
- As the time frame is limited, not all of the functionality of EAT is taken into consideration. However, all of the functionality is taken into consideration when building the general architecture.
- Important functionality that was not implemented include:
  - evaluation function of the CM
  - further visualization of the analysis (pie charts, diagrams)
  - adding of evidence

Quality related:
- Two main non-functional requirements are discussed: extensibility and scalability.
- User feedback is not collected as it is not fully functional
3. Method

The goal of this thesis is put rather broadly. In short, this master thesis researches whether and how it is possible to build an Eclipse-based version of EAT tool. As there are limitations to the size of research, mainly the time frame of 20 weeks, it is not possible to build a fully-functional new version. However, building a design and partly implementing an artifact is possible. In order to achieve the main goal of this thesis, a design research is performed. “Design research involves the analysis of the use and performance of designed artifacts to understand, explain and very frequently to improve on the behavior of aspects of Information Systems.”[6]

The method used in this master thesis follows the general method of design research as outlined by Figure 1 and article [6].

![Figure 1 – General methodology of design research](image)

1. “Awareness of the problem” – analysis of the EAT is performed and main problems are outlined. This corresponds to output Constructs
2. “Suggestion” – the design for the next version of EAT is outlined. This corresponds to output Model
3. “Development” - an artifact using the designed is implemented. This corresponds to outputs Method and Instantiation
4. “Evaluation” - evaluation takes the form of a discussion of the architecture and its non-functional requirements,
5. “Conclusion” - results from the research are outlined

Those steps however are not directly corresponding to the chapters of this report (for better mapping, please refer to section Scope). In addition, the first four steps are organized into two iterations with the circumscript as the link between them. The first iteration is larger and is related to the design of the AM, while the second one is smaller and is related to the design of CM. The second iteration is a continuation of the research from the first one.
4. Theory

In order to achieve the goal of this thesis, it is essential to choose a suitable research method. As its goal is “to research” whether Eclipse Modeling Project (incl. EMF, ECore, DSL, etc.) can be used for building an Eclipse-based version of the EAT tool” it is clear that investigation about the possibilities of EMP should be made. Based on that investigation, critical analysis should be performed. This sequence of actions takes the direction of empirical research, because “Empirical research is any research that bases its findings on direct or indirect observation as its test of reality.”[17] However, as the main focus of this thesis is the specific EAT tool (which is an Information System), it is important to check upon the empirical research methods in Information Systems (IS) and software engineering. There is a number of sources discussing different empirical methods in IS and software engineering in general [19, 20, 21, 22, etc.]. Books [21 and 22] provide extensive information about those methods. Articles [19 and 20] provide an overview of those methods. It is important to have a broader perception of what methods are there, that is why the methods discussed in those articles are presented here. Both articles agree on the definitions of those methods and roughly provide the same summary of them:

- (Controlled) experiment – investigation of a testable hypothesis where one or more independent variables are manipulated to measure their effect on one or more dependent variables.
- Surveys – retrospective study of a situation that investigates relationships and outcomes.
- Case study – empirical inquiry that investigates a contemporary phenomenon within its real-life context, especially when the boundaries between phenomenon and context are not clearly evident
- Action research – iterative process involving researchers and practitioners acting together on a particular cycle of activities, including problem diagnosis, action intervention, and reflective learning.

However, all of those methods do not fit exactly the research needed in this master thesis. Surveys and case studies for example are more or less retrospective in a sense, they explore existing phenomena. Action research has a social focus – for example, changing the process of bug reporting and measuring its effect on team work or individual performance in the same time. Controlled experiments’ ”most important application is in testing theories and hypotheses”. Those are fitted best for more scientific theories, not like the practical one in this thesis.

In order to achieve the goal of this master thesis, a research about the design of a future version is needed. There is another type of empirical research that deals with design context, especial ly when the boundaries between phenomenon and context are not clearly evident. However, all of those methods do not fit exactly the research needed in this master thesis. Surveys and case studies are presented here.

In order to achieve the goal of this master thesis, it is crucial to understand the process of performing design research; however, it varies to some degree. Article [24] gives a bit more abstract view on design research. It provides a general framework, but does not go into details about the method of design research. In addition, it provides some guidelines about design research. They are summarized in Table 1:

<table>
<thead>
<tr>
<th>Guideline</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Guideline 1 : Design as an Artifact</td>
<td>Design-science research must produce a viable artifact in the form of a construct, a model, a method, or an instantiation.</td>
</tr>
<tr>
<td>Guideline 2 : Problem Relevance</td>
<td>The objective of design-science research is to develop technology-based solutions to important and relevant business problems.</td>
</tr>
<tr>
<td>Guideline 3 : Design Evaluation</td>
<td>The utility, quality and efficacy of a design artifact must be rigorously demonstrated via well-executed evaluation methods.</td>
</tr>
<tr>
<td>Guideline 4 : Research Contributions</td>
<td>Effective design-research must provide clear and verifiable contributions in the area of the design artifact, design foundations and/or design methodologies.</td>
</tr>
<tr>
<td>Guideline 5 : Research Rigor</td>
<td>Design-science research relies upon the application of rigorous methods in both the construction and evaluation of the design artifact.</td>
</tr>
<tr>
<td>Guideline 6 : Design as a Search Process</td>
<td>The search for an effective artifact requires utilizing available problem management.</td>
</tr>
<tr>
<td>Guideline 7 : Communication of Research</td>
<td>Design-science research must be presented effectively both to technology-oriented as well as management-oriented audiences.</td>
</tr>
</tbody>
</table>
They are incorporated throughout the carrying of this master thesis. On every step, an implicit attempt to follow to as much as possible those guidelines is made. Furthermore, they can be used in order to evaluate the overall quality of the research performed by this thesis.

Article [23] provides very good overview on design research – its definition, methodology, process, etc. It contains references, summaries and quotations of many other scientific articles about design research in IS. It also contains a very good summary of a design research method (shown in Fig. 1) based on other methods suggested by different authors. However, article [26] presents the design research not as a linear process like the one shown in Fig. 1 (awareness of the problem, suggestion, development, evaluation, conclusion), but rather as formed by a number of iterations. This is shown below in Figure 2.

Figure 2 – Empirical research through design process

The research performed in this master thesis is also non-linear and non-straightforward. It contains a number of iterations (generally two iterations – one connected with making the design for the AM and one with the design of CM). However, article [23] also recognizes the iterative nature of the design process – “Development, Evaluation and further Suggestion are frequently iteratively performed in the course of the research (design) effort. The basis of the iteration, the flow from partial completion of the cycle back to Awareness of the Problem, is indicated by the Circumscription arrow.” In fact, the problem how to organize the overall design research is only dependant on the way of evaluating the design. The case study presented in article [26] makes an evaluation after every iteration, but those evaluations are related more to general design rather than user satisfaction. Eventually the authors of the case study measure the total user satisfaction.

As already mentioned, there are two main iterations – one for making the design of the AM, the other – for solving the problems with making the design of CM and connecting it to the AM. After both iterations a brief evaluation is performed. This evaluation is orientated towards finding weak spots and improving the overall design on the next iteration. This does not come as a contradiction to the method used by this thesis, it just fits in it.

In the same time, it is implied from the case study presented by the same article that a brief evaluation of the design is also performed when iteration occurs.

It is now clear that there are two aspects of making evaluation when it comes to design:

- First, after every iteration an evaluation of the design from a technical perspective is made.
- Second, when the design research is done, an evaluation of the stakeholders’ / end-users’ satisfaction is performed.

Article [25] is widely cited paper, used also by article [23] when it comes to the outputs of the design research. The authors of [25] suggest a two-dimensional framework for research in information technology. However, it is further taken into consideration by article [23] and one of the dimensions is used as a basis for describing output of design research. This dimension propose four general outputs for design research: constructs, models, methods, and instantiations. They are explained in Table 2 below.

Table 2 – The outputs of design research

<table>
<thead>
<tr>
<th>Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Constructs</td>
</tr>
<tr>
<td>---</td>
<td>------------</td>
</tr>
<tr>
<td>2</td>
<td>Models</td>
</tr>
<tr>
<td>3</td>
<td>Methods</td>
</tr>
<tr>
<td>4</td>
<td>Instantiations</td>
</tr>
<tr>
<td>5</td>
<td>Better theories</td>
</tr>
</tbody>
</table>

For the purposes of this thesis, the first four types of constructs are used in order to present the outputs of the design research after every iteration. This is the essence of the thesis, as that lays the foundation of a future full scale development. In addition, it answers the original main objective of this thesis. It is important that evaluation on a larger scale of the proposed design is made. After it is justified in terms of functionality, there are some non-functional aspects that need to be analyzed as well. They are as important as the functionality itself. Usually they are incorporated in the design in advance. However, in this case they are analyzed after the design is made based only on the functionality. This is a consequence of the main goal. The goal of this thesis is to check whether EMP can be used. This incorporated uncertainty if this is plausible. That is why they should be seen as refinements of the quality of the design rather than requirements to it.
5. Iteration 1

5.1. Preface

The first iteration of the design research effort of this thesis is dedicated to designing the Abstract Modeler (AM) of the EAT tool. It results into the first four types of input described in the Theory section.

First a brief overview of the modeling concepts of EMP and their analogue in the current EAT version is presented. Based on that, a discussion on two potential architectures of the AM is made. One of them is chosen after analyzing the advantages and disadvantages of both proposed architectures. Thus, the Constructs and Model output of this iteration are formed. After that a plan on building a prototype of the chosen architecture is made. That plan forms the Method output. Eventually, the plan is executed and Instantiation is built.

5.2. EMP modeling concepts

5.2.1. EMF

Both books [28] and [29] start with an introduction of the general idea behind the modeling concepts of EMP. They both introduce EMF and ECore as fundamental part of modeling within EMP. “EMF is a framework and code generation facility that lets you define a model in any of these forms (Java, XML, UML) from which you can then generate the others also the corresponding implementation classes. Figure 3 shows how EMF unifies the three important technologies: Java, XML and UML. Regardless of which one is used to define it, an EMF model is the common high-level representation that “glues” them altogether.” [28] Figure 3 shows the place of EMF between XML, UML and Java:

![Figure 3 - Unifying JAVA, XML & UML.](image)

5.2.2. Introducing ECore

“The model used to represent models in EMF is called ECore” [28]. As we can see from Figure 4 below, no matter what kind of technology is used originally to represent the initial model, it is translated to an ECore model and eventually to a Java code (Java code with annotations itself can be the original source for the model) in order to instantiate an EMF concrete model. Figure 4 summarizes this idea:
5.2.3. ECore as a meta-model

ECore is itself an EMF model and thus is its own meta-model.” Why does EMF need its own model? Well, the answer is quite simply that ECore is a small and simplified subset of the whole UML. Full UML supports much more ambitious modeling than the core support in EMF.”[28] In other words, all the models made using EMP are defined in terms of ECore model fundamentally. ECore is then the meta-meta model used by EMP. This is expressed by Figure 5 below:

5.3. EAT modeling concepts

There is a number of important modeling concepts (meta-meta model, meta-model, concrete model) when it comes to EMP. It is important to show that those concepts appear also in EAT tool, even though they are not directly named this way. From a theoretical point of view, there are three main steps in instantiating a concrete model in EAT and assessing it. Those are shown in the Figure 6 below:
Figure 6 - Steps in EAT modeling

The architecture of the current version of EAT tool implements this process. There are two big parts of the EAT tool – the “Abstract modeler” and the “Concrete modeler”. In the context of the process that was described, the Abstract modeler supports the first activity, while the Concrete modeler supports the other two activities. From an architectural point of view, the Abstract modeler is an editor for defining abstract PRM models. After that Concrete modeler use the already defined models from the Abstract modeler. They are first imported when opened, then concrete models can be created and edited in an editor, evidence can be added and finally, an analysis is executed. Main components and their output can be seen in Figure 7:
However, in the context of the current discussion it is more important to show how the modeling concepts are implemented. First, we have the PRM formalism that defines the overall structure of the models produced in the Abstract Modeler part of EAT. Then the Concrete Modeler “makes the tool user able to instantiate and analyzes this defined structure”. This way the PRM formalism coded as java classes serves as the meta-meta model in EAT, while the models produced by the Abstract Modeler are in fact meta-models. This is shown in Figure 8 below.

Figure 8 - EAT modeling concepts

5.4. Using a transformation mechanism between Abstract Models and ECore

5.4.1. Conceptual base
First option of using EMP (as in an alternative implementation of EAT tool) is to match the modeling concepts of EAT and EMP. More specifically, meta-models in the sense used in EAT to become meta-models in EMP sense. Respectively, the instance models from EAT will be represented by concrete EMF models. A central point is using a transformation mechanism between EAT tool and EMP. This transformation mechanism aims at matching the models made with the Abstract Modeler part of current EAT tool to ECore models. This establishes a conceptual base for this option. It is important to dwell into the potential architecture and implementation.
5.4.2. Potential Architecture

One of the requirements of this thesis is if it is possible to make use of EMP, the GUI should resemble the now existing version of EAT tool's GUI. Following the ideas from the conceptual base, sample architecture is shown in Figure 10. The major software components are presented with round rectangular, while the output between them is denoted with rectangular.
First, a new editor for modeling of meta-models should be implemented (represented by “Eclipse Abstract modeler” in Figure 10). The design of the needed editor however, should be based on MVC pattern and in this sense would resemble the design of the “Abstract modeler” part from the current EAT version. The main output of this editor should be a PRM model in the form of Java classes or XML schema.

Second, a special purpose transformation mechanism would be needed. It is important to point out that it is the implementation of the concept from previous sub-chapter. Its main purpose would be to transform the PRM model into ECore model. Consequently, an ECore model would be the output of this mechanism.

Third, a new editor for modeling concrete models should be implemented (represented by “Eclipse Concrete modeler” in Figure 10). The design of the needed editor however, should be based on MVC pattern and in this sense would resemble the design of the “Concrete modeler” part from the current EAT version. It would take an ECore file as its input and would be able to produce concrete models based on it. The main output of this editor should be an EMF concrete model (XMI file).

Forth, the evaluation function from the Concrete modeler from the current EAT version should be incorporated in the code of “Eclipse Concrete modeler”.

Figure 10 - Potential architecture of the use of EMP and transformation mechanism
5.4.3. Analysis and potential issues

There is one main strength with this use of EMP in a future Eclipse-based version of EAT. Namely, it would provide a ready to use concrete models from abstract models.

However, there are a number of issues with this architecture:

1. The modeler of abstract models would have to follow the MVC model. This means extensive coding and it is not different from implementing the current version of EAT “Abstract Modeler” using Eclipse technologies following the same architecture model as in the current version.
2. Implementing the Generation mechanism would mean using reflection and use of Eclipse APIs. This is generally considered tricky and error-prone.
3. The modeler of concrete models would have to follow the MVC model. This means extensive coding and it is not different from implementing the current version of EAT “Concrete Modeler” using Eclipse technologies following the same architecture model as in the current version.
4. “Eclipse Concrete Modeler” should be sensible to the different ECore files imported in it and probably change its GUI according to them. This means extensive coding of different options. Those issues should be very seriously considered if this option is considered as a potential architecture.

5.5. Using EMP for modeling Abstract Models

5.5.1. Conceptual base

There is another option of using EMP. In the previous option, an attempt is made to “match” the modeling concepts of EAT current version and EMP. However, this is not a must and necessity. As the meta-meta-model of EAT currently is a set of Java classes (PRM formalism expressed as Java classes), those can be used as a meta-model in EMP. Accordingly, the concrete EMF models produced from this meta-model would “match” the abstract models of EAT current version. In this sense, the concrete models of EMP are “matched” to meta-models of EAT. This is shown in Figure 11.
5.5.2. Potential architecture
Following the ideas from the conceptual base, a sample architecture can be seen Figure 12. There are four major components:

1. Java annotated classes (that would represent PRM formalism)
2. ECore model (the PRM formalism in the form of ECore model)
3. Generated Eclipse application with editor for modeling abstract models (this can be called Eclipse Abstract Modeler)
4. XMI XML describing a concrete model
From those four components, only the first one should be implemented thoroughly. From it, the second is generated automatically. The third one, Eclipse Abstract Modeler is generated from the ECore model, but the editor it contains need to be partly enhanced. Finally, a XMI XML schema is the output that describes a specific abstract model that is instantiated.

5.5.3. Analysis and potential issues
Using EMP for modeling abstract models has a couple of advantages over the option of building from scratch. There are only two things to be actually developed. First, this is the PRM formalism expressed as Java classes. As there is already one implemented in the current version of EAT, it can be re-used to some extend or at least used as a helpful reference. Second, the editor should be more user-friendly using GMF. However, this is still much less complicated from building everything from scratch following the same architectural pattern as EAT current version.

There are a couple of issues associated with this potential use:
1. PRM formalism to be presented as Java annotated classes may prove a bit complicated, so special attention to this should be paid.
2. Editor's code should be enhanced, so that the interface becomes more user-friendly and convenient.
3. Special attention should be spent on how the Concrete Modeler part of the current Eat version would be implemented.

5.6. Choice of architecture
The two options presented are immediate consequence of the comparison of modeling concepts in the two projects – EAT and EMP.

The second option is preferred for research (and implementation) for the following reasons:

- The second option makes use of EMP in order to ease implementation. In contrast, the first option just tries to fit the modeling concepts without bringing any viable benefit to implementation.
- The second option is much more straightforward to implement, while the first option holds real implementation challenges.
- There is a limited period for the implementation and research on the topic. The second option is the one that can fit in that period.

5.7. Output: Constructs
At this stage, there is enough information to define the main constructs from this design stage. They concern only the Abstract Modeler part of the EAT tool. In the new Eclipse-based version this part is called Eclipse Abstract Modeler (EAM) in order to differentiate it from the current version. It is the final product (the artifact) of the overall design effort for this piece of software. EAM has two main parts:

1. PRM formalism expressed under some form.
2. Visual part for defining the PRM formalism.

In addition, the output of EAM is a file containing this definition. The overall picture of them is shown in Figure 13:
It is important to point out that those constructs also appear in the current version of the EAT tool. This is not a coincidence, but a result of the overall design analysis of the EAT tool.

5.8. Output: Model
The model should describe relationships between the different constructs that were already defined. As potential architecture was already chosen and described in chapters 5.5 and 5.6., the model of the EAM takes it into account together with the constructs from the previous chapter. Eventually, the model is a combination of both the constructs and the proposed architecture.
The PRM formalism is modeled and coded in EAM as Java annotated classes. Then, an ECore model is generated by them and this is represented by the dashed arrow between them. EMP provides some frameworks (GMF, EMF) that build on ECore in order to make build a graphical part for defining it. As a result of defining the PRM model, an output file containing those definitions is produced. However, it is not easy to predict whether using those frameworks can provide all the needed functionality for modeling a PRM model as it is now without actually implementing some of this functionality. Furthermore, this is necessary as it concerns Eclipse Concrete Modeler and how both will be connected. Documenting the output of implementing some of the functionality is the third type of output from this design research effort – Method. The artifact itself is the forth form of output – Instantiation.

5.9. Building the Eclipse Abstract Modeler (EAM)

Building of EAM on this iteration is a main topic of the Research Plan of this thesis. There it is called “proof of concept” as this is a popular term in software engineering. There is no exact definition how a “proof of concept” should be built, that is why its implementation is practical. To some extent, this implementation could be seen also as a prototype. However, further labeling may lead to confusion as there are numerous sources on prototypes. That is why it is limited to a “proof of concept” and referred to as such.

5.9.1. Goals structure
The above figure represents the goal breakdown structure for building the EAM. The general aim (overall goal) is “To make a proof of concept”. That is why it is put on top. It can be broken down into two main goals. The first is to “construct the proof” and second is to “evaluate the proof”. There are three main things in constructing the proof. The first thing is to “make a list of features”. After those essential features are defined, they should be codified by “Construct PRM formalism as ECore model” and “Develop UI based on GMF”. When all that is completed, evaluation based on the completion of those features must be done.

5.9.2. Delimitations
Making this proof of concept is limited by the list of features that is compiled. Those are essential features of the now-existing Abstract modeler so that a PRM model can be built. An attempt is made to implement them and after that an evaluation of this attempt is made. There is only information about GMF provided as a number of building sample applications and not a full and comprehensive reference to the features of GMF. Other resources like time and hardware may prove challenging, but they do not limit building the proof of concept.

5.9.3. Literature
Literature used concerns the technologies that are used in order to implement the next Eclipse-based version of EAT tool. It is already mentioned (previous section) that there is just limited number of sources on GMF. Book [29] provides a tutorial about GMF through examples and should be considered as a first-choice source as it is recommended by the Eclipse organization. A valuable source is the GMF tutorial [30] as provided on Eclipse site. However, most of it is covered in one of the examples in the book. The other technology that will be probably used – RCP is covered in book [31]. In addition, literature on how to build a proof of concept (mainly what methodology to use) turns out to be scarce (if any). No formal and standard process for building it exists.

5.9.4. Data
The data needed to make the proof of concept may be divided into two. The following table explains the need for those two types and their main source.

Table 3 - research data needed
<table>
<thead>
<tr>
<th>Type</th>
<th>Justification</th>
<th>Main sources</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Data needed to build a PRM model</td>
<td>It is essential that the proof of concept builds proper PRM models</td>
<td>1. PRM formalism definition [32] 2. The EAT tool source code implementation under NetBeans</td>
</tr>
<tr>
<td>2. Data about the user functionality features of EAT Abstract Modeler</td>
<td>It is essential that the proof of concept provides the UI functionality features the current version provides</td>
<td>1. EAT Abstract modeler current UI 2. The EAT tool source code implementation under NetBeans</td>
</tr>
</tbody>
</table>

### 5.9.5. Process

As there is no standard and largely accepted process for building a proof of concept, the process is tailored so that it helps in achieving the goals already defined. Due to the nature of type of project being built, namely a small software project with incrementally added features, the development methodology is agile.

Steps:
1. Make a list of user functionality features
2. Choose unimplemented feature from the list to be implemented
3. Attempt to implement the feature. This includes refining and updating the PRM ECore model and GMF implementation. PRM ECore model and GMF implementation depend on each other.
4. Analyze the implementation attempt

Steps 2, 3 and 4 are carried out in that sequence while there are unimplemented features from the list. However, if the list undertakes updates (like adding/removing a feature) this should be reflected in the following steps.

The making of the proof of concept follows the process in the Figure below. The move from step to step is reflected with a full arrow:

![Figure 16 - Development process](image)

**5.9.6. Framework for evaluation**

It is important to know if every existing feature of EAT Abstract modeler can be implemented using EMP. In order to do so, after an attempt to implement every feature on the list, it is evaluated against the scale below:

<table>
<thead>
<tr>
<th>Implementation degree</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Implementable</td>
<td>The feature can be implemented using GMF and looks like in the previous version</td>
</tr>
<tr>
<td>Modified</td>
<td>The feature can be implemented using GMF, but appears differently</td>
</tr>
</tbody>
</table>
Unimplementable | The feature cannot be implemented using GMF

Best case scenario is that all of the features are “Implementable”. However, if some (all of) them are “Modified” a further evaluation of user satisfaction should be made. If is important that no feature proves unimplementable. This will generally mean that GMF is incapable to provide all essential features.

5.10. Output: Method and Instantiation

As a result of the process described in previous sections, an instantiation is built. In addition, a step by step knowledge how to build it is generated and documented. First a list of features is compiled as is suggested by step 1 of the process described in section 5.9.5. This list is presented as Appendix A. Then, attempt to implement those features is undertaken. The results from it are:

- Evaluation of the features in Appendix A with the framework described in section 5.9.6 as well as further consideration about the implementation of the features (Appendix B)
- The instantiation itself

However, the instantiation is not fully functional version of the Eclipse Abstract Modeler. This is due to a number of reasons:

1. The intention of building this instantiation is just to check if it is possible to actually implement the functionality of the current AM using Eclipse and to generate step by step knowledge how to build it
2. The time is very limited
3. The instantiation grows big enough, so that there is a need of a version control system as well as a building tool. Version control system is not that hard to choose from, however, choosing a good and appropriate building tool may prove hard. The reason is that there the frameworks used to build the artifact are used in a specific sequence.
4. It is not clear if those features should not be changed. For example, some of the current features may be better to redesign as there is a feedback based on user experience. This should be decided if a decision to actually implement the Abstract Modeler is made
5. It is not clear whether it should be an Eclipse plug in or it should be a RCP application

It is important that after the implementation effort, it is clear if and how the features of the AM are implemented. This is essence of the method output.

As there are no features marked as Unimplementable, building an Eclipse version of the Abstract Modeler seems plausible and optimistic. In addition, building this artifact helps in making the design for the Concrete Modeler.

How the artifact is built and installed is a topic of another document [33]. Below in Figure 17, a screenshot of the artifact as an Eclipse plug in is shown.

![Figure 17 – Eclipse Abstract Modeler as Eclipse plug-in](image-url)
5.11. Discussion

5.11.1. Functional vs. non-functional requirements towards AM

So far the results of the design effort are presented as suggested by the method and theory used. However, those results and the ways of achieving them on that iteration are more technical and do not provide a good overview of the achieved success in a larger context. It is not only important that the new version just copies the features of the old version. In terms of user functionality, on that iteration is shown that building AM with EMP is plausible and optimistic (please, see the previous section). Even though this is a key challenge, it is also essential that this new version brings certain benefits over the old one.

Apart from the functional requirements towards the system (i.e. providing tools for graphical definition), there is a number of non-functional requirements that are important. Non-functional requirements can be described as requirements that "Describe the non-behavioral aspects of a system, capturing the properties and constraints under which a system must operate".

They are discussed on that stage, as they are important for the overall quality and future improvements of the design, but they are hard to explicitly state in advance. This is due to the fact that it was not sure that EMP can be used as a platform for building the future version. However, if a decision to build the next version of EAT using the research in this report, those requirements should be taken into consideration and be incorporated in advance.

There are some non-functional requirements that are usually listed. They may vary in their name and exclusiveness, but generally requirements like usability, security, extensibility, testability, scalability and maintainability and performance are listed.

5.11.2. Extensibility

Probably the most important non-functional requirement towards the Eclipse version of EAT is extensibility. This is due to the fact that on that stage of research, not the full functionality of EAT tool is considered. Nevertheless, all important functional features are at least identified and considered in building the overall architecture. Furthermore, new functionality is planned, for example, adding restrictions to the models.[16]

The definition of extensibility varies to some degree between sources, however [Fel! Hittar inte referenskälla.] gives a very good definition."Extensibility is a principle according to which software design should consider future, potential growth of solutions. Extensions are implemented sequentially by modifying current functionality or as incremental functionality. Changes should have minimal impact on existing implementations. ... In addition, XML and XBRL were designed specifically with the extensibility concept as their core feature.".

The overall architecture of the next version of EAT (particularly the AM) is making use of EMP and it follows its modeling principles. It uses ECore as its base for abstract models. Thus, it can make use easily of other frameworks included in EMP. Good example is the GMF. It provides very good tools for building a graphical interface on top of a ready model. Built on top of Java, those frameworks provide a lot of extension points with pure Java coding in addition to their main features. Thus, future extensions of the AM can rely on both the ready frameworks and Java language. It is enough to have a well built initial model in order to add extensions to it. However, this may become the bottle neck point for this project. Adding to or changing the original ECore, may make it necessary to change crucial parts of other parts of the system, i.e. the graphical parts. Even though there are frameworks in EMP (i.e. M2M) that deal with model transformation and can ease complex changes of ECore, this changes to the ECore remain a source of complication.

It is important that even with this flaw, this architecture is much more flexible and extensible from the current architecture. It requires much less effort in implementation provided the programmers are familiar with the EMP (model-driven development as a whole) and PRM.

In the current version, everything was built using Java and making use of few other frameworks. Most notably, this is the NetBeans visual library that was discussed earlier. As a consequence, everything had to be built from scratch. Not only the logic behind PRM model had to be implemented, but also extensive coding for the graphical modeling part was necessary. The effort for coding all that can be seen in the many classes the current version includes.

In a potential Eclipse version, extensive Java coding will not be needed. Instead, most of the functionality would be just utilizing the frameworks EMP provides. For example, building the ECore requires just stating the main abstractions in a form of Java classes with just one member. Building the graphical interface requires defining only 2 xml files. This is incomparably less effort than pure Java coding.

Another benefit is that this approach uses a standard frameworks, that are tested and working well
5.11.3. Scalability

Usually, when considering extensibility as an important quality of an architecture, scalability is also taken into consideration. Simply put, "Scalability is the ability to quickly meet demands for increased performance, processing power, network connectivity or data storage." It is important that an architecture should be scalable in order to make the most of its available extensibility. As the architecture for the new version of AM is heavily dependent on the EMP, it is up to Eclipse as a platform to be scalable (either if the new version is implemented as an Eclipse plug-in or standalone application using RCP). The question with the scalability of Eclipse as a platform is a bit tricky and not easy to answer. It is not a good idea to look for the answer at the development team of Eclipse about its scalability, as this opinion tends not be completely objective. There are two aspects of the scalability considered - scalability in terms of adding new functionality and in terms of making large models. Those two aspects are not mutually exclusive but rather co-existing, i.e. adding new functionality would make possible making larger models:

1. Adding new functionality may slow down the performance of graphical modeling. For example, full implementation of attributes modeling may require some Java coding as extension to GMF. This may compromise the performance. However, as most of the functionality of AM is already implemented, this is not seen as a big issue. Sound programming practices can help prevent problems on that aspect.

2. The second and more important aspect of scalability considers making of large models. That means abstract models having many elements in them. Most probably adding more elements to a model (i.e. many classes and attributes) will compromise the graphical modeling performance. However, this is not easy to know in advance.

Those two aspects are dependent how the AM is implemented:

1. First, if the AM is implemented as an Eclipse plug-in, its performance will be dependent on Eclipse. This means that further investigation about the modeling capabilities of Eclipse as a platform should be carried out. However, generally Eclipse is quite well fitted for managing increasing data volumes in its editors. Nevertheless, what is the optimum should be investigated.

2. Second, if the AM is implemented as RCP application, its performance will not be dependent on Eclipse. In this case it will be a standalone application and it will be rather more dependent on the underlying operation system and hardware. Again, as in the previous case, further investigation is necessary.

5.11.4. Other non-functional requirements

Apart from scalability and extensibility, there are other non-functional requirements that are often mentioned in literature. Some of them were already incorporated in the current discussion, like performance. Others like security are irrelevant on that stage. It is important that a decision to fully implement the EAT tool using EMP is taken. This way, more qualities can be analyzed and incorporated in the overall design of the tool. In addition, scalability needs to be further investigated. It is partly dependent on whether AM (and EAT generally) is implemented as Eclipse plug-in or a standalone application using RCP.
6. **Iteration 2**

6.1. **Preface**

The second iteration of the design research effort of this thesis is dedicated to designing the Concrete Modeler (AM) of the EAT tool. It results into the first four types of input described in the Theory section.

First a reconsideration of chapter 5.5 is made, taking into account the current implementation and further analysis of the architecture. Apart from the architecture of the CM, the other big challenge discussed is how the EAM and ECM will “connect” to each other. Thus, the Constructs and Model output of this iteration are formed. As there are two main goals, the scope of implementation (respectively the Method and Instantiation outputs) are limited to them.

6.2. **Modeling ECM with EMF**

When the EAT modeling concepts were introduced and discussed in chapter 5.3, the concrete models defined in the CM were put last. They are defined over an abstract model defined in the AM. This precondition of having an already defined abstract model in order to define a concrete model hinders the use of EMF in the development of CM. In the current version, when concrete models are defined, their skeleton is already known from the abstract model (their meta-model). The original idea of using ECore as meta-model could not function here; because the abstract model is not known in advance (it must be specified first by the AM). In other words, the skeleton of the model is dependent on the abstract definitions. Using ECore is very convenient, as a graphical editor can be built on top of it as well restrictions are easy to define. That is why a deep analysis of the functioning of EAT was performed. The output of it is presented in the next chapter.

6.3. **Architecture of the Concrete Modeler**

After a more thorough examination of the implementation of the Concrete Modeler, a number of important things were found. Generally, it contains 3 main parts:

1. **XPRM** – this part defines the skeleton of the concrete model
2. **Graphical part** – an editor for defining the concrete model
3. **Evaluation function** – this part contains code for evaluating a concrete model

The general flow is:

1. First, an input file with the abstract definitions is loaded in the modeler.
2. Based on them, graphical definitions are made in order to specify the concrete model.
3. After that evaluation is performed.
4. Output file with the definition of the concrete model is produced.

This is shown in Figure 18 below:

![Figure 18 - Constructs in CM](image)

The most important result of the analysis is that the XPRM part actually makes a heavy use of the AB’s part for specifying the Abstract models. It just “consults” the data specified in the input file to make
restrictions on the models defined with it. In a sense, the concrete model can be seen as an extension of the abstract model. This is crucial for the next steps of designing the Concrete Modeler.

6.4. Output: Constructs

After analyzing the current implementation of the Concrete Modeler and taking into account the capabilities of EMP, it is clear that the Eclipse CM should contain the following part:

1. A mechanism for translating the input file from the EAM and building the skeleton of the concrete model. It is referred to as “XPRM builder”
2. Graphical part for defining the concrete model
3. Evaluation function working on the ready model

In addition, the output of ECM is a file containing the definition of the model. The overall picture of them is shown in Figure 19:

![Figure 19 - Model of ECM](image)

The same constructs appear in the current version of the Concrete Modeler and this is result of the overall concept of the project. It is not dependant on the implementation and technologies used.

6.5. Output: Model

The model should describe relationships between the different constructs that were already defined. The big issue is how to make the XPRM builder using ECore. Eventually, the model proposed in Figure 20 below describes it.
A new ECore model coded as Java Annotated classes is the focal point of the XPRM builder. It is aware of the PRM ECore model of the Abstract Modeler. It reads an abstract model from an input file and thus defined the skeleton of the concrete model. As in the case of the EAM, EMP is used to build on ECore in order to make build a graphical part for defining a concrete model. As a result of defining the concrete model, an output file containing those definitions is produced. The most important functionality of the EAT tool – evaluation function, is another Java module that evaluate the model from the output file. The most important issue here is to check how the ECore model can be implemented. At this point it is not clear if it can be actually built in the way needed. An attempt for doing that is performed and then the result of it is described in the next section – Method. It is not important to check if EMP can provide all the graphical modeling features of the CM as when building the EAM. This is due to the fact that it is already clear that EMP can provide most (all with some modifications) of the needed graphical modeling features. In fact, the graphical modeling with the CM is not as extensive as that of the AM. Another part that is not subject of the current design effort of this stage is the Evaluation function. This is due to the time limit. However, it is clear that such a function can be built re-using most of the one incorporated into the current CM.

6.6. Output: Method

When describing the building of the EAM, making of the ECore model is not extensively explained. This is because it is actually not very complicated and follows the already coded PRM formalism as Java classes in the current EAT version. Generally, the PRM is modeled as number of connected POJOs in the current version of AM. EAM follows the same logic. This way a PRM is made an ECore model. It contains all the modeling concepts of EAT like classes, attribute, etc and it can be seen below in Figure 21.
The classes are contained in a package called PRM. As it was pointed out in chapter 6.3., the concrete model can be seen as an extension of the abstract model. A new package called IPRM is added, that follows the logic of the IPRM package of the current CM. It is referred to IPRM ECore. The classes there use the PRM package only for “consultation”, they just “read” it.

A GMF editor is built using the IPRM ECore. When opened, the output file from the EAM can be loaded into the definition of the IPRM ECore concrete model. This is done by simply adding one line to the output file. Thus the editor has access to the definitions of the abstract model. This is generally everything what is needed in order to establish the good communication between the EAM and ECM.

6.7. Output: Instantiation

As a check about the capabilities of the ECore was made, the final output is the instantiation of the Eclipse Concrete Modeler itself. It is by no means fully functional; however, it answers in a successful way the question of communication between the EAM and ECM as well as the overall look and feel of the ECM.

In the figure below the ECM instance is shown as Eclipse plug-in. On the left is the diagram of the concrete model. On the right is the output file (XMI schema instance). It is important to notice a couple of important things:

- The second last line contains the reference to the file containing the definition of the abstract model.
- There is a <prmClass> tag under the <iprmClass> denoting the reference to the abstract class from the concrete class.
- In the bottom part of the figure, the references from the output file are presented in a user-friendly way using EEF.
6.8. Discussion

6.8.1. Non-functional requirements

After the first iteration of the design effort, a big part of the discussion was devoted on non-functional requirements. On that stage, when biggest design issues about functionality have already found some solution, a re-consideration and update on the non-functional requirements is necessary.

Considering extensibility, it was demonstrated that taking over the advantages of ECore (EMF) can provide a lot of benefits. This is a very good example of the extensibility of the used framework. In addition, this also further proves the extensibility capabilities of EMP. As the overall architecture of ECM is similar to that of EAM in terms of used technologies and ideas, the discussion of extensibility from iteration 1 is valid on that iteration as well.

When it comes to scalability, not much progress is made investigating it. It still remains an issue that needs further work. In terms of the first aspect discussed after the previous iteration, it is clear that adding the functionality of the CM does not compromise performance even though the same platform and resources were used. However, the second aspect becomes even more important as the model now is larger. The situation with the other non-functional requirements remains the same.

6.8.2. Meta-Models modifications

The current version of EAT supports modifying (either removing, adding or updating parts) of both abstract and concrete models. This is incorporated in the code of EAT application. It is not explicitly incorporated as a separate construct in the Output sections on both that and the previous iteration. This is due to the fact that it considers supporting of the models and it is incorporated in all parts of the architecture rather than being a separate functionality (i.e. building the graphical part of the model or evaluating it).

The Eclipse EAT will use XMI for output of both concrete and abstract models as it was demonstrated on both iterations. This is the standard format used by ECore. Using a standard always is beneficial in software engineering as it boosts communication, using out-of-the-box solutions and reusability. The benefits in that case are rather connected to extensibility and they were already discussed in the discussion on the previous iteration. If a specified either abstract or concrete model needs to be updated, this can be done easily not only through the graphical interface, but also using other tools of EMP.

What is important here is what will happen with the models if the ECore files are changed. For example, if a new type of relationship between attributes is added or some of the properties of the attributes are changed. This is basically modifying the underlying meta-model and accordingly the specified models are sensitive to that. That is why it is important to show how those modifications will affect the versioning of models.

It is important to remind that the different constructs of both AM and CM are dependent just on the ECore, so updates to the meta-model should start with updating the ECore and the other constructs should be updated (graphical part, evaluation function, etc.). There are three possible scenarios when changing the ECore files.
1. Adding to the ECore models. For example adding a new type of relationship between the attributes and classes. This kind of modification to the meta-models will not affect the specified models themselves. It will be possible to be updated with the new feature. Simply put, the old models will be in a IS-A relationship with the new models.

2. Removing from the ECore models. For example, the relationship described in the previous becomes obsolete. Best case scenario, the ECore model will be remain untouched and just the graphical tools should be updated so that they not allow adding this type of relationship.

Those first two cases make things look very simple. However, it is very improbable that adding or remove to the ECore model would not require adding or removing of some logic. This is discussed in the next case.

3. Updates to the elements of ECore models. For example, if a relationship between classes faces some restriction. Thus, some of the already existing models may become irrelevant. However, those models most probably would contain information that is important and it is not easy to build them one by one so that they fit the new restriction. Fortunately, the designers of EMP thought of such a situation and M2M framework can help solve that kind of situations. With the help of it, the already existing models can be updated very easily. Just a file with the new definitions is needed and after that they just need to be processed. However, updates to the ECore models require updates to the other used frameworks from EMP as ECore is used as a base. In the case of the example defined, the graphical part should be also changed.

In order to perform modifications to the meta-models, it is important that the programmers should have solid knowledge in three different areas. First, this is PRM formalism and the current version of EAT. Second, knowledge in model-driven development. Third, experience with EMP. Having all of them would seriously enlighten the task. However, for someone lacking experience with EMP it may take some technical time to get used to the frameworks. Some of them like GMF are pretty sophisticated. The worst thing about EMP is that there is not much literature about it and that eventually make it hard to grasp and work with. Having knowledge about the ideas and concepts of model-driven development may sufficiently easy the use of EMF which will speed up implementation. Last but not least, good understanding of PRM can help finding simple programming solutions to coding the formalism as a ECore model.

It was already mentioned that if an update to the ECore model is needed, only a mapping file between the new and the old ECore model needs to be created, after that the models are simply processed automatically. For someone having the mentioned skills, making such a file would not take longer than a couple of days. On the other side, someone not having any of those skills, it will take most probably 2 weeks. In addition, updating the graphical interface may prove even trickier. Updating it may take up to a couple of days for simple updates like a new relationship with additional logic even for an experienced programmers.

### 6.8.3. Standalone application vs. Eclipse plug-in

In the current discussion as well as in the discussion on iteration 1, it was mentioned a couple of times that there are two possible implementations of EAT using Eclipse as platform. The first is to implement EAT as an Eclipse plug-in. The second is to implement it as a standalone application using RCP. In fact, the original idea was to implement EAT as an Eclipse plug-in (even though the current version is a standalone application.). In the course of implementation, the option to transfer the implemented plug-in to a RCP was discovered. The current implementation using RCP provides exactly the same functionality as the implementation as plug-in. In fact, it was generated from the plug-in. So far this can be done without a great effort. However, the more features and technologies used, the harder it will be to get one-to-one functionality in the plug-in and the standalone implementation. Furthermore, changing one of them would mean changing the other as well. Keeping two alternative implementations at the same time is tough goal to achieve. That is why it is important to make a decision which implementation is preferred. There are some factors to be considered before making that decision.

1. The target audience that will use the tool should be taken into consideration. For people used to Eclipse, using EAT as an plug-in would not be a problem. However, for people not into development Eclipse at first may prove challenging.

2. Scalability as already discussed should be taken into consideration (please, see the 5.11.2)

3. The provided support for both implementations be taken into consideration. Eclipse plug-in development is a largely discussed topic. There are quite some books on the topic as well as examples and information on the internet. This definitely ease the development of EAT Eclipse version as a plug-in. Furthermore Eclipse foundation constantly comes up with enhancements, new frameworks and bug fixes.
Another important advantage of implementing EAT Eclipse as a plug-in is that it can make use of other Eclipse frameworks apart from EMP (for example Mylyn). When it comes to RCP, it seems not as hot topic as plug-in development. Standalone applications nowadays are not as popular as web applications and plug-ins. Plug-ins have a very big advantage and that is the reusability the supporting platform (in this case Eclipse) provides. RCP is a complex framework with a steep learning curve[35]. There are also good literature sources for it even though not as many as those for plug-in development.

In conclusion, when making the decision between standalone application or plug-in for the EAT Eclipse version, all of those factors should be taken into consideration.
7. Conclusions

Throughout carrying out the research of this thesis, the main objective is pursued. Two iterations of design research were carried out corresponding to the two major parts of EAT - Abstract Modeler and Concrete Modeler. Those iterations include considerations about the functionality of the Eclipse based versions and design for the new architecture. In addition, considerations about non-functional requirements and overall architecture were incorporated.

After carrying out the research planned, it is clear that Eclipse Modeling Project (incl. EMF, GMF, EEF, etc.) have the potential to be used as an alternative platform to using NetBeans in implementing EAT tool. In addition, the most important result is that the foundations of future full-scale implementation are laid by providing design and basic implementation as Eclipse plug-ins. However, this research was limited by time and resource and does not explain all details of the architecture of EAT. It is important that those are not crucial and taken into consideration. Nevertheless, further research on the topic can take advantage of all the results from both iterations.

There are some implementation issues that are subject of future work. One of them is the evaluation function of the ECM. Another one is the definition of attributes in EAM. Other is adding of evidence in ECM. However, all of them depend more or less on the implementation of PRM and IPRM ECore files, which was already incorporated in the research of this thesis.
REFERENCES


27. McNamara, Carter, PhD. General Guidelines for Conducting Interviews, Minnesota, 1999
33. Instructions for building EAT Eclipse
Appendix A: EAT Abstract Modeler - User functionality features and looks

03 April 2011

This part appendix is created in order to fulfill Step 1 of the process defined in the Research plan. Step 1 assumes to “Make a list of user functionality features”. The following appendix contains not only the defined user functionality, but also the looks of main objects defined by EAT Abstract Modeler.

All the features provided by the EAT Abstract Modeler can be divided roughly in 4 groups. The first 3 groups are formed by modeling of objects. There are three main objects that can be modeled with the Abstract Modeler (Classes, Attributes and Relationships). In addition, there is one other group that contains general features of Abstract Modeler as an editor (named Editor Specific Features). Those features can be further classified as:

- “Essential” - meaning that their implementation is crucial for defining a PRM formalism
- “Non-essential” - meaning that their implementation is not essential to building a PRM formalism and can be omitted

In addition to the user functionality features of every main object, it is important to describe the looks of every of the objects on the canvass. That is why the first chapter of this document is devoted to the “Looks” of the main objects on the canvass.

On Step 2 of the process defined in the Research plan, a feature (or a number of features) is chosen for implementation. A feature from the list in “Chapter 2: User functionality features” is defined by the name of the group it belongs (Classes/Attributes/Relations/Editor Specific Features), classification (Essential/Non-Essential) and further numeration in that group and class. For example, feature with name 2A2b means that this feature is in group 2 (Attributes) classified as “Essential” and put under number 2b.

Chapter 1: Looks

1. Classes
   1. A class is represented by a rounded rectangular
   2. A class has two compartments separated by a line
      a. Upper one appears always
      b. Upper one contains:
         1. - the bold name of the class as label (obligatory and generated by default)
         2. - its super class’s name as label in double figure brackets (if set)
         3. - self reference icon before its name as a rounded arrow (if set)
         4. - special icon (if set)
      c. Lower one appears when there are attributes added:
         d. Lower one contains the names of the attributes as labels
   3. A class has an associated menu. It is available after selecting the class on the canvass and right clicks of the mouse

2. Attributes
   1. An attribute is represented by a label on its containing class’s lower compartment
   2. Discrete attribute is represented by bolded text
   3. Continuous attribute is represented by non-bolded text in italics
   4. An attribute has an associated menu available after pointing the attribute on the canvass and right click of the mouse over it

3. Relationships
   There are 4 types of relationships:
   - self-reference on a class
   - self-connection on an attribute
   - relationship between classes (reference slot)
   - relationship between attributes

   1. Self-reference on a class is represented by a rounded arrow in the upper compartment of a class before its name
   2. Self-connection on an attribute is represented by a triangle arrow before the attributes’ name

34
next to the bordering class. The color of the arrow depends on the type of the attribute
   a. Discrete attributes use black colored arrow
   b. Continuous attributes use gray colored arrow
3. Relationship between two classes is represented by undirected arrow between the borders of the classes
4. Relationship between two classes has an associated menu available after pointing the relationship on the canvas and right click of the mouse over it
5. Relationship between two attributes is represented by directed arrow between the borders of the classes that contain those attributes

4. Editor specific features
There are 3 special views apart from the canvas of the editor.
   1. There is a zoomed out view of the current canvass and its current position among other objects.
   2. There is a properties view on the objects. Some of objects expose their properties there
   3. There is a view with check boxes making certain objects visible or not on the main canvass

Chapter 2: User functionality features
1. Classes
   A. Essential features:
      1. New class is added by either:
         a. choosing this option from a button
         b. right click on an empty space on the canvas and choosing it as an option from the menu
      2. The following operations can be performed from the class menu list:
         a. set super class - choose one from a list of available super classes
         b. set description - setting a text
         c. delete the class - this includes also updates to related attributes, classes and relationships
   B. Non-essential features
      1. The following operations can be performed from the menu list:
         a. set shape (this feature is more complicated than simply entering a value)

2. Attributes
   A. Essential features:
      1. New attribute is added after selecting its containing class and choosing this option from its menu list
      2. The following operations can be performed from the menu list:
         a. show attribute configuration (see feature 2A3)
         b. set description - setting a text
         c. delete the attribute - this includes also updates to related attributes, classes and relationships

   Extension - 28 April
   Attribute configurations are essential in making the PRM model.
   3. Attribute configurations have two departments switched between 2 tabs:
      a. General tab” - contains a table with all the relationships between attributes that have this attribute as part of them listed here together with the aggregation function that apply to each. Some operations can be performed:
         1. “Add new connection” - available through a button. Please, see the last line of feature 3A10
         2. “Remove connection” - available through a button and provided that a relation from the list of available relations is selected.
         3. Change the aggregation function - available through selecting the aggregation function that needs to be changed and changing it with another one from a drop-down list.
         4. Change the name of the attribute - available through a text box containing the name of the attribute
      b. “Properties tab” - contains a mean to add values to discreet attributes and continuous attributes respectively. The view associated with doing so is dependent on the type of the attribute. (More detailed description may be added later depending on the need of it.)

End of extension
3. Relationships
There are 4 types of relationships:
- self-reference on a class
- self-connection on an attribute
- relationship between classes (reference slot)
- relationship between attributes (attribute kinship)

A. Essential
1. Self-reference on a class is added after selecting the class and choosing the “self-reference” option from its menu. Two options are provided:
   a. add “Directed” self-reference
   b. add “Undirected” self-reference
2. Self-reference on a class is updated after selecting the class and choosing the “edit” option from its menu. A dialog containing its properties (label, label on the parent side, label on the child side, municipality on the parent side, municipality on the child side) appears and all of the properties can be updated.
3. Self-reference on a class is removed after selecting the class and choosing the “edit” option from its menu. The dialog containing its properties appears, button for removing the reference should be selected and finally this should be saved.
4. Self-connection on an attribute is added after selecting the attribute and choosing the “add self connection” option from its menu.
5. Self-connection on an attribute is removed after selecting the attribute and choosing the “delete self connection” option from its menu.
6. Relationship between classes is added by selecting a class, pressing the Ctrl key and the left mouse button and dragging an arrow to another class.
7. Relationship between classes is updated either:
   a. After selecting the relationship and choosing the “set properties” option from its menu.
   b. Double click on the relationship.
   A dialog containing its properties (label, label on the parent side, label on the child side, municipality on the parent side, municipality on the child side) appears and all of the properties can be updated.
8. Description on a relationship between classes is updated after selecting the relationship and choosing the “description” option from its menu.
9. Relationship between classes is removed after selecting the relationship and choosing the “delete reference slot” option from its menu.

Extension - 28 April  (Features related to attribute kinship)

There are two essential constraints on relationships between attributes:
- A relationship is added between attributes of the same type (either between two discreet or two continuous attributes)
- There should be existing a chain of reference slots (relationships between classes) between the classes that contain the two attributes.

Every relationship between attributes has an aggregate function. By default it is always set to AVG when the relationship is first created.

10. Relationship between attributes is added in two ways:
   a. By selecting an attribute, pressing the Ctrl key and the left mouse button and dragging an arrow to another attribute.
   b. See feature 2A3a1
In order to define a relationship between attributes, a number of reference slots should be specified.
11. Relationship between attributes is updated either:
   a. After selecting the relationship and choosing the “set default” option from its menu.
   b. Double click on the relationship.
   The specific properties set as default to the relationship depend on the type of attributes (discreet or continuous) and it looks like setting values for a specific attribute of that kind. (See 2A3a)
12. The aggregation function on a relationship between attributes can be updated after selecting the
relationship and choosing the “description” option from its menu.

13. Relationship between attributes is removed after selecting the relationship and choosing the “delete reference slot” option from its menu.

4. Editor specific features
   B. Non-essential features
      a. On checking up a check-box in the view with the check-boxes, certain objects appear or disappear from the main canvass.
Appendix B: Status of features implementation progress  
15 April 2011

This appendix describes the research progress about the implementation potential of different features provided by the current version of EAT Abstract modeler with EMP.

According to Research Plan, there can be 3 values in ‘Current status’:
I  - implementable
M  - modifiable
U  - unimplementable

Here, we add two more statuses:
N  - not yet considered / status will be available later
R  - currently under Research

<table>
<thead>
<tr>
<th>Feature code</th>
<th>Current status</th>
<th>Potential</th>
</tr>
</thead>
<tbody>
<tr>
<td>1A1a</td>
<td>I</td>
<td>A button for creating a class is present on the palette. *</td>
</tr>
<tr>
<td>1A1b</td>
<td>I</td>
<td>Currently, this is not implemented. However, a current action can be used in order to implement this feature.** In addition, a short-cut can also be considered. *** As there is one way to perform this action, a second option is not that crucial.</td>
</tr>
<tr>
<td>1A2a</td>
<td>M / I</td>
<td>This is completely possible just with the use of GMF and the properties sheet for the class. However, this can be done also with the help of an a current action.**</td>
</tr>
<tr>
<td>1A2b</td>
<td>M / I</td>
<td>This is sheet for the class. However, this can be done also with the help of a current action.**completely possible just with the use of GMF and the properties</td>
</tr>
<tr>
<td>1A2c</td>
<td>M / I</td>
<td>Using GMF, an object is deleted after selecting it and pressing the delete button. ?</td>
</tr>
<tr>
<td>1B1a</td>
<td>N</td>
<td>The properties sheet contains a section for the appearance of an object. This need to be further explored if it can provide exactly the same functionality.</td>
</tr>
<tr>
<td>2A1</td>
<td>M / I</td>
<td>A button for creating an attribute is present on the palette. * However, a current action can be used in order to implement this feature.**</td>
</tr>
</tbody>
</table>
| 2A2a         | R              | Currently, attempt to incorporate EEF is made. However, further effort is needed. It can provide probably most of the needed functionality.  
Currently, it can be incorporated with the simple editor, but it is not possible to incorporate it |
<p>| 2A2b         | M / I          | This is completely possible just with the use of GMF and the properties sheet for the class. However, this can be done also with the help of a current action.** |
| 2A2c         | M / I          | Using GMF, an object is deleted after selecting it and pressing the delete button. ? |</p>
<table>
<thead>
<tr>
<th>Feature</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>3A1a / 3A1b</td>
<td>M</td>
<td>There is no difference between the self-references and reference slots (relationships between classes) for the object part of EAT, it is the same object represented in two different ways. However, a self-reference can be represented with an arrow from a class to itself, using the same visual object as reference slots. A button for creating a reference between classes is present on the palette. * Then the two classes referenced are pointed. If it is one and the same class, this is considered as self-reference. However, a current action can be used in order to implement this feature.** In addition, a short-cut can also be considered. ***</td>
</tr>
<tr>
<td>3A2</td>
<td>M / I</td>
<td>This is completely possible just with the use of GMF and the properties sheet for the reference. Properties are not shown in a dialog, but rather in a sheet. There they are fully available for change.</td>
</tr>
<tr>
<td>3A3</td>
<td>M</td>
<td>Using GMF, an object is deleted after selecting it and pressing the delete button. ?</td>
</tr>
<tr>
<td>3A4</td>
<td>M</td>
<td>There is no difference between the self-connection and attribute kinship (relationships between classes) for the object part of EAT, it is the same object represented in two different ways. However, a self-connection can be represented with an arrow from a class to itself, using the same visual object as reference slots. A button for creating a reference between attributes is present on the palette. * Then the two attributes referenced are pointed. If it is one and the same attribute, this is considered as self-connection. However, a current action can be used in order to implement this feature.** In addition, a short-cut can also be considered. ***</td>
</tr>
<tr>
<td>3A5</td>
<td>M</td>
<td>Using GMF, an object is deleted after selecting it and pressing the delete button. ?</td>
</tr>
<tr>
<td>3A6</td>
<td>M</td>
<td>A button for creating a reference between classes is present on the palette. * Then the two classes referenced are pointed. If it is one and the same class, this is considered as self-reference (Please see the Potential section of feature 3A1). However, a current action can be used in order to implement this feature.** In addition, a short-cut can also be considered. ***</td>
</tr>
<tr>
<td>3A7a / 3A7b</td>
<td>M / I</td>
<td>This is completely possible just with the use of GMF and the properties sheet for the reference. Properties are not shown in a dialog, but rather in a sheet. There they are fully available for change.</td>
</tr>
<tr>
<td>3A8</td>
<td>M</td>
<td>This is completely possible just with the use of GMF and the properties sheet for the reference. However, this can be done also with the help of an a current action.**</td>
</tr>
<tr>
<td>3A9</td>
<td>M</td>
<td>Using GMF, an object is deleted after selecting it and pressing the delete button. ?</td>
</tr>
<tr>
<td>3A10</td>
<td>R</td>
<td>Attribute kinship is heavily dependent on the properties sheets. (Please see also the Potential section of feature 2A2a)</td>
</tr>
</tbody>
</table>

*GMF defines a palette with tools for creating the modeled objects on the canvass

**GMF specifies actions that can be added to menus.

*** GMF specifies short-cuts that can be used.
Status of features implementation progress
28 April 2011

This part of the appendix describes the research progress about the implementation potential of different features provided by the current version of EAT Abstract modeler with EMP. It is the natural from 15 April 2011. There were 2 features which were under research and needed more input in order to be clear how they can be modeled. This is how they looked like on the first version of the document:

<table>
<thead>
<tr>
<th>Feature code</th>
<th>Current status</th>
<th>Potential</th>
</tr>
</thead>
<tbody>
<tr>
<td>2A2a</td>
<td>R</td>
<td>Currently, attempt to incorporate EEF is made. However, further effort is needed. It can provide probably most of the needed functionality. Currently, it can be incorporated with the simple editor, but it is not possible to incorporate it</td>
</tr>
<tr>
<td>3A10</td>
<td>R</td>
<td>Attribute kinship is heavily dependent on the properties sheets. (Please see also the Potential section of feature 2A2a)</td>
</tr>
</tbody>
</table>

According to Research Plan, there can be 3 values in ‘Current status’:
- I - implementable
- M - modifiable
- U - unimplementable

After some research performed and further elaboration of the them in the features list here are the results:

<table>
<thead>
<tr>
<th>Feature code</th>
<th>Current status</th>
<th>Potential</th>
</tr>
</thead>
<tbody>
<tr>
<td>2A3a</td>
<td>M</td>
<td>Adding this tab generally does not mean more than just providing the already existing functionality in another view. All of the functionality here can be simply performed with the other features as follows: Feature code - Feature with same functionality 2A3a1 - 3A10 2A3a2 - 3A13 2A3a3 - 3A12 2A3a4 - Changing names of attributes and classes are not described as a additional feature, however, it is completely possible with using GMF. However, if this proves as a crucial feature, please refer to the Potential section of feature 2A3b</td>
</tr>
<tr>
<td>2A3b</td>
<td>I/M/R</td>
<td>The research about this feature is dependent on EEF and its potential. As EEF is a brand new, there were difficulties in finding enough information about it. <a href="http://www.eclipse.org/forums/index.php?t=rview&amp;goto=666109">http://www.eclipse.org/forums/index.php?t=rview&amp;goto=666109</a> At first, there was a problem with incorporating them with GMF. After resolving it with a work-around, it ran successfully. It proves to be a powerful framework and could be used in order to literary model every part of the properties associated with the attributes.</td>
</tr>
</tbody>
</table>
However, the lack of a building tool and CVS at this stage makes it very time-consuming to make even minor changes to the original model. At this point it is clear that all the features needed are implementable using the Eclipse technologies (EMF, GMF and EEF) in order to make a new version of the Abstract modeler, even though some are not directly implemented.

<table>
<thead>
<tr>
<th>Feature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>3A10a</td>
<td>I/M/R</td>
</tr>
<tr>
<td></td>
<td>A button for creating a relationship between attributes is present on the palette. * Then the two attributes in the relationship are pointed. If it is one and the same attribute, this is considered as self-reference. However, a current action can be used in order to implement this feature. ** In addition, a short-cut can also be considered. *** A problem here is selecting the reference slots. They should be added via a special form. This is possible with EEF, however, not very clear at this stage exactly how. (Please see 2A3b)</td>
</tr>
<tr>
<td>3A11</td>
<td>M</td>
</tr>
<tr>
<td></td>
<td>This is completely possible just with the use of GMF and the properties sheet for the relations. However, this feature is heavily dependent on feature 3A10</td>
</tr>
<tr>
<td>3A12</td>
<td>I / M</td>
</tr>
<tr>
<td></td>
<td>The list of available aggregation functions is incorporated in the existing model and easy to switch between them.</td>
</tr>
<tr>
<td>3A13</td>
<td>M</td>
</tr>
<tr>
<td></td>
<td>Using GMF, an object is deleted after selecting it and pressing the delete button.</td>
</tr>
</tbody>
</table>