Technische Universität Berlin

DIPLOMA THESIS

Scenario-based Architectural Decision Support within Enterprise Architectures

- Concept and Implementation of a Prototype -

Markus Buschle
Matr.Nr: 300940
Contessaweg 4A
14089 Berlin

Torsten Derlat
Matr.Nr: 301759
Alte Flatower Straße 21
OT Tietzow
14641 Nauen

Daniel Feller
Matr.Nr: 300695
Schönburgstr. 22
12103 Berlin

March 2009

Institute for Business Informatics,
Faculty of Electrical Engineering and Computer Sciences,
Berlin Institute of Technology (BIT), Germany

In collaboration with

Department of Industrial Information and Control System,
Kungliga Tekniska Högskolan, Stockholm, Sweden

And

Deutsche Telekom Laboratories, Berlin, Germany.

Eidesstattliche Erklärung

Ich, Markus Buschle, erkläre hiermit an Eides statt, dass ich die vorliegende Diplomarbeit selbstständig und ohne unerlaubte Hilfe angefertigt, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt und die den benutzten Quellen wörtlich oder inhaltlich entnommen Stellen als solche kenntlich gemacht habe.

Berlin, den ....

Eidesstattliche Erklärung

Ich, Torsten Derlat, erkläre hiermit an Eides statt, dass ich die vorliegende Diplomarbeit selbstständig und ohne unerlaubte Hilfe angefertigt, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt und die den benutzten Quellen wörtlich oder inhaltlich entnommen Stellen als solche kenntlich gemacht habe.

Berlin, den ....

Eidesstattliche Erklärung

Ich, Daniel Feller, erkläre hiermit an Eides statt, dass ich die vorliegende Diplomarbeit selbstständig und ohne unerlaubte Hilfe angefertigt, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt und die den benutzten Quellen wörtlich oder inhaltlich entnommen Stellen als solche kenntlich gemacht habe.

Berlin, den ....

Zusammenfassung

Auf dem Gebiet der Unternehmensarchitektur ist es häufig notwendig, Entscheidungen zur Auswahl zwischen verschiedenen alternativen Datenverarbeitungssystemen zu treffen. Oftmals sind die Abhängigkeiten zwischen den Systemen und Einflüsse der verschiedenen Systeme untereinander nicht offensichtlich. Um die Entscheidungsfindung in einer solch komplexen Umgebung zu unterstützen, wird in dieser Arbeit ein Konzept der Unternehmensarchitekturanalyse vorgestellt. Die Methode ermöglicht die Analyse von verschiedenen Aspekten der Unternehmensarchitektur, wie zum Beispiel Verfügbarkeit oder Sicherheit.

Die Methode basiert auf verschiedenen Modellen, deren Aufbau erläutert wird. Im ersten Schritt wird ein Meta Modell, ein sogenanntes abstraktes Modell, erzeugt. Dieses stellt die unterschiedlichen Einflüsse zwischen den verschiedenen Systemen und ihren unterschiedlichen Eigenschaften dar. Im zweiten Schritt wird ein sogenanntes konkretes Modell des abstrakten Modells instanziiert. Dies geschieht durch das Sammeln von unterschiedlichen Belegen über die Existenz der Systeme und die Ausprägung ihrer Merkmale. Im dritten und letzten Schritt wird ein Bayessches Netzwerk aus diesem konkreten Modell einer Entscheidungsalternative generiert. Wenn mehrere verschiedene Szenarien durch jeweils ein konkretes Modell dargestellt werden, kann die Auswahl einer Alternative durch die quantitative Auswertung dieser Szenarien unterstützt werden.

Um die vorgestellte Methode der Analyse von Unternehmensarchitekturen verwendbarer zu machen, wird im Rahmen dieser Arbeit das Konzept und die Implementierung einer Software, genannt Enterprise Architecture Tool (EAT), vorgestellt. Zuerst werden vorab getroffene Entscheidungen und Überlegungen über den Aufbau des Programms und die verwendeten Komponenten beschrieben. Der Aufbau des Programms wurde so gewählt, dass eine möglichst gute Unterstützung für verteilte Entwicklung gegeben ist.

Die Implementierung fand unter Berücksichtigung eines Vier-Phasen-Konzeptes statt. Dabei wurden die folgenden Aufgaben „Planung der nächsten Phase", "Bestimmung von Vorgaben, Alternativen und Einschränkungen durch eine Prioritätenliste", „Entwicklung und Test" und „Feedback in täglichen und wöchentlichen Meetings" jeweils wiederholt durchgeführt.

Der letzte Teil der Arbeit beschreibt die Implementierung der Software, und es werden weitere Ideen und Ansätze zur möglichen Erweiterung des Programms aufgezeigt.

Acknowledgements

Many people in many places contributed to this diploma thesis. Especially as this thesis is the outcome of a cooperation between the Berlin Institute of Technology, the Royal Institute of Technology Stockholm (KTH), and the Deutsche Telekom Laboratories, several supporters aided us to fulfill this project successfully. We, Markus Buschle, Torsten Derlat, and Daniel Feller, have to thank everyone who was involved in the development of this diploma thesis. Following we want to mention all the helpers that accompanied us, while we worked on this thesis.

Special thanks to Elke Lorenz, who helped us with all organizational matters that we came across from the first day to the final colloquium. She always had time and lots of cookies for us.

We are also very grateful to Daniel's dad, Mr. Feller, who helped us to revise this thesis. Our document would not look like this without your assistance.

We want to thank Marten Schoenherr, who supported us during the whole time, while we wrote this thesis. Together with Per Närman and Johan Ullberg, he offered us the chance to get part of the cooperation mentioned at the beginning.

We are also thankful to Ulrik Franke, who visited us in Berlin and prepared our Stockholm stay with weekly Skype meetings. We want to thank Robert Lagerström, for representing our reception committee in Stockholm. We are also very grateful to Pontus Johnson, who was an important support and motivation to make this diploma thesis possible. Moreover, he provided us with valuable ideas to explore “Greater Stockholm” and the beautiful city itself. We also have to thank Teodor Sommestad for his creative suggestions and helping us booking a ferry to Helsinki. We would also like to thank all the other people who are working at the ICS department of the KTH and supported us before, during or after our stay in Stockholm.

In addition, we have to thank Oliver Holschke, who provided us with helpful ideas and sources.

We are very grateful to Prof. Dr. rer. nat. Peter Pepper and Prof. Dr. rer. nat. Klaus Obermayer, who agreed spontaneously to attend our final colloquium. Finally yet importantly, we thank our two supervising tutors Prof. Dr. rer. pol. Hermann Krallmann and Dr.-Ing. Matthias Trier.

Every one of us additionally has some people, he wants to say thank you to:

Markus: First of all I want to thank Daniel and Torsten: I won't have come so far, within the last years, without you. My French friends you avoided me from buying a carpet in Sweden and taught me how to eat canned salad. Many (hug)s go out to my sister, who is always just one text massage away. Thank you for all the favors, and please never forget the superman thing. I am also very grateful to my parents, who always believed in me, no matter what idea I was following. As well, many thanks to the Schmelzel family, who kept me practicing English during the last years. Finally, I want to say thank you to my friends. Together we brunched, danced, and talked away along the bumpy road leading to this diploma thesis. The Baltic Sea won't separate us.

Torsten: I want to thank Daniel and Markus for their reliability and friendship since 2004. They are the guarantors for our successful study and for the survival of the coffee industry in the current financial crisis. Then thanks go to my sister, Dana, for her eager reviewing-eye. Without her, many snippy faults would still be hidden in the document. I also thank Steve Ackerman for his reviewing efforts despite his full schedule. Thanks to the System Analysis department for introducing the excitement of research to me. Special thanks to my parents for their support throughout the whole studies. And last but not least, I want to express my gratitude to Ulrike for her unconditional love and motivation (no matter how far away I was).

Daniel: I want to thank my parents for making my academic studies possible and for supporting me in every possible aspect. In addition, I want to thank my grandmother for supporting me mentally during rainy days. Especially, I want to thank Andrea for her love, her support, her motivation and every thing else.
Finally, I want to thank Markus and Torsten for the great and most successful last four and a half years. Meeting you was one of the best events in my life.

Table of Contents

Figures

Tables

Abbreviations

AM	Abstract Model (used in names of java-packages)
API	Application Programming Interface
AWT	Abstract Window Toolkit
BN	Bayesian Network
CDDL	Common Development and Distribution License
CIO	Chief Information Officer
CM	Concrete Model (used in names of java-packages)
COBIT	Control Objectives for Information and elated Technology
CPM	Conditional Probability Matrix (synonym of CPT)
CPT	Conditional Probability Tables
CRD	Conditional Probability Distribution
CRM	Customer Relationship Management
CRUD	Create, Read, Update and Delete
CVS	Concurrent Versions System
DAG	Directed Acyclic Graph
DPN	Diagnostic Probability Networks
DSS	Decision Support Systems
DTD	Document Type Definition
EA	Enterprise Architecture
EAT	Enterprise Architecture Tool
EB	Expected Benefit
EPL	Eclipse Public License

ERD	Entity-Relationship Diagram
EV	Expected Value
GeNIe	Graphical Network Interface
GM	Graphical Models
GPL	GNU General Public License
GUI	Graphical User Interface
HTML	HyperText Markup Language
IDE	Integrated Development Environment
IT	Information Technology
JDK	Java Development Kit
JPA	Joint Probability Approach
JPD	Joint Probability Distribution
MCM	Multiple Cause Module
MPA	Marginal Probability Approach
MVC	Model–View–Controller
OMG	Object Management Group
PNG	Portable Network Graphics
PRM	Probabilistic Relational Models
QOC	Questions, Options and Criteria
SCM	Single Cause Module
SMILE	Structural Modeling, Inference, and Learning Engine
SOA	Service-Oriented Architecture
SQL	Structured Query Language
SVN	Subversion (version control system)
SWT	Standard Widget Toolkit

TS	Test Strength
UI	User Interface
UML	Unified Modeling Language
XML	Extensible Markup Language
XSD	XML Schema
XUL	XML User Interface Language
W3C	World Wide Web Consortium
WFMS	Workflow Management System

PART I – Introduction

1 Abstract

In the field of Enterprise Architecture, it is necessary to make decisions about different alternatives on information systems. Often dependencies and influences between different systems are not obvious. For the support of decision making in such complex environments, the concept of a model-based enterprise architecture analysis is presented. This method enables the analysis of various system goals like availability, responsibility or security. In a first step a Meta model, called abstract model, is created to describe influence paths between different system attributes. In a second step, evidence about the existence and the various entities and the state of their attributes is collected and compiled into a concrete model. In the last step, this concrete model is transformed into a Bayesian network and the state of non-observable attributes is calculated. When different scenarios are evaluated, the decision is supported by the quantitative measure of the attributes of the different scenarios. To support this method of enterprise architecture analysis, the conceptual work for a software tool, called Enterprise Architecture Tool (EAT), is presented. The architecture is laid out to support distributed development. Finally, the implemented software tool is documented along with ideas and possibilities for further extensions.

2 Motivation

With increasing spreading of information systems in enterprises, the complexity and heterogeneity of a business' IT landscape grew, too. By comparison the time available for adapting to market changes, i.e. the adjustment of business processes, decreased. However, the monolithic and disintegrated systems constrain a firm's ability to react to changes. Thus, there was a rising amount of IT consolidation and extension projects, as well as more research activities in the field of enterprise architecture. Obviously, the rising amount of projects necessitated research efforts in this area as well as research results within business informatics and system analysis, e.g. EAI, WFMS, SOA, boosted the number of projects. However, this resulted into an increasing demand of decisions within the field of enterprise information systems, e.g. the decision for a project or decisions within a project. Unfortunately, in practice the decisions made were subjective, hardly traceable, and poorly documented [114]- despite their mostly huge financial and pervasive impact. Neither supporting enterprise architecture tools, like Qualiware [86] or System Architect [102], nor frameworks from the software architecture field (e.g. ATAM [9], C2SADEL [62]) provided sufficient capabilities for measuring an information system's quality within the enterprise architecture domain [46].

The method presented in this work, a model-based approach for enterprise architecture analysis, supports decision-making in the context of enterprise information systems. It addresses the problem of handling uncertainty. Therefore, it considers probabilistic influences between system elements. To provide "good" models for analysis the method distinguishes between the modeling of generic and recurring information system architecture situations (in abstract models) and precise, situational modeling in concrete models. Therefore, the concentration on a particular system property in a concrete model ensures that it is most applicable for the analysis it is subjected. A tool that supports decision-making by implementing this method is described in this work. It is called the Enterprise Architecture Tool (EAT) and is based on a previous proof of concept. The usage of several technologies, which were already utilized in the proof of concept and in other evaluations of the method, is continued in the presented implementation. To realize uncertainty handling, the tool makes use of a library for building and calculating Bayesian networks.

3 Chapter Overview

This thesis is structured as follows:

Chapter 1: Abstract
This chapter introduces this diploma thesis.

Chapter 2: Motivation
The situation, which led to the necessity of conducting a tool-supported analysis within enterprise architecture, is introduced. Further, the need for the development of a new software tool is pointed out. The problem-solving characteristics of the underlying method are mentioned.

Chapter 3: Chapter Overview
Here, an overview of the chapters of this thesis is given.

Chapter 4: Definitions
Chapter 4 denotes relevant technical terms, which are essential for this document. By giving the definitions for terms, which are fuzzily defined usually, ambiguousnesses are prevented and a common understanding of the used expressions is attained.

Chapter 5: Information System Decision Making
This chapter explains the reasons for decision-making within enterprise architectures. Several aspects that are relevant for decision-making are discussed here. The information system goals, which decision makers in an enterprise architecture context strive for, are also illustrated in this section.

Chapter 6: Enterprise Architecture Analysis
The method of enterprise architecture analysis is described in this chapter. Its underlying models and their mathematical background are presented as well. Finally, an etension of the analysis method and an alternative approach for decision support is presented.

Chapter 7: Concept of a Prototype
In the “Concept of a Prototype” section, the architecture of the presented implementation is described. Therefore, the components used, and how they interact with each other, are illustrated. Furthermore, the internal structure of the tool and the composition of the model elements’ visualization are illustrated and explained.

Chapter 8: Implementation
The structure of the implementation is considered in this chapter. Here, class- and package-structure is explained in detail. In addition, the user interface and the calculation, which is performed by the concrete modeler, are described. This chapter ends by pointing out individual solutions used during the implementation process.

Chapter 9: Usage Example
The application of the presented implementation is demonstrated in the "Usage Example" part. For this purpose, a simple enterprise architecture analysis scenario is considered. The modeling process, using the abstract modeler and the concrete modeler, is illustrated stepwise.

Chapter 10: Extension Guide
The extension guide offers information about the addition of features to the presented tool. Detailed information about the data flow of the implementation is given, too. Exemplary two possible extension scenarios are discussed. Finally, further ideas are presented and recommendations are made.

Chapter 11: Discussion and Conclusion
This chapter gives a summary and discusses the result of this diploma thesis. Furthermore, it gives an overview what kind of future research is still necessary in the context of probabilistic decision-support.

Appendix A: Calculation
"Appendix A" provides additional information about the calculation process, illustrated in chapter 8.3.4.

Appendix B: XSD
In this appendix, the underlying XSD documents of the abstract model and the concrete model are visualized. A tree-like representation allows a brief overview of the models' structure.

Appendix C: Evaluation Table
The table used for the evaluation of graph libraries and frameworks is illustrated in this part of the thesis. The table contains the four candidates, the evaluation criteria, the assessment scale, the results, and according statements. The evaluation itself is explained in chapter 7.1.4.

PART II – Theory and Concept

This part deals with an approach for scenario based decision support within enterprise architecture. To achieve a common understanding of the terms used in this thesis, definitions of scenario and enterprise architecture are given. The scope of operations is set out by providing an introduction into information system decision making. The method for enterprise architecture analysis, its models and its mathematical foundation, are introduced and described in detail.

Finally, a concept of a prototype for supporting the analysis method is depicted. The description of the concept consists of a presentation of the used technologies, libraries and software. In preparation of the prototype's implementation, several architectural decisions, that enable an interaction of the different components, are documented.

4 Definitions

As this thesis has been written within the field of Enterprise Architecture (EA), this chapter takes a closer look at EA. Thereby section 4.1 examines EA's current situation of an extensively used, but still unsatisfactorily defined buzzword. Moreover, it tries to define the understanding of EA in this thesis. Chapter 4.2 then explains the origin, the necessity, and usage of models for visualizing EA's necessary constitutive elements. These models are the basis of the scenario-based architectural decision-support – Enterprise Architecture Analysis.

4.1 Enterprise Architecture

As mentioned before, the term Enterprise Architecture is not used uniformly in the literature – thus there is no common terminology today. [56, 82]

One reason is that technology professionals have a wide-ranging view of Enterprise Architecture –

"[...] starting from a structured family of technical guidelines including concepts, principles, rules, patterns and interfaces, and the relationships among them, to use when building a new IT capability. In contrast, business professionals tend to ignore the term as an IT-only issue. Rather, they talk in terms of business models, business processes and, sometimes, the business architecture". [13]

Moreover, Marten Schoenherr discovered in his literature analysis [88] that only 6% of 126 considered EA publications give its own definition and differentiate it to others by referring to their definitions. He also noticed that other authors "[…] are extending

their architectural understanding from a Software Systems Architecture view towards a wider perspective considering organizational and/or social aspects in an enterprise." He found out that almost 10% explicitly state that there is no common EA-Terminology. The lack of a common understanding is indicated by the fact that the focus of existing (proprietary) definitions either is technology-driven, systematic, method-driven, or a mixture of them [88]. Furthermore, the vagueness of EA's focus is underlined by the fact that different phrases with the same connotation as Enterprise Architecture are being used in literature, such as Enterprise IT Architecture, Enterprise Information System Architecture, Information System Architecture, Enterprise Software System Architecture, and Enterprise System Architecture [16].

The evolvement of the Enterprise Architecture discipline started in 1987. At that time Zachman introduced his Framework for Information Systems Architecture [112], which he extended in collaboration with Sowa in 1992. It was the first approach in Enterprise Architecture as his framework supports analyzing the enterprise's business and IT structure. The term Enterprise Architecture was not introduced in that period. Instead, according to [60] the term was created in 1996 after the Clinger-Cohen Act [10] had been passed. This act directed the US public sector to take an integrated approach for aligning business with IT. Then the interest in Enterprise Architecture rose dramatically as the number of publications show [88].

The problems in defining the term Enterprise Architecture become obvious when trying to explain the term "architecture". Etymologically the word architecture is composed of the Greek word *αρχη* [*arché*], meaning "origin", "beginning", "foundation", or "first", and the word *τεχνη* [*techné*], meaning "art" or "handcraft". However, it also can be derived from the Latin word *tectum* ("building"). Combined, architecture can be translated as "original art" or "first handcraft". Nevertheless, the second part of the word was further related to *techné*, which could also mean "technology" or "tectonics". This would not relate the term to an activity or person, but to the structured and organized relationship of elements [1]. From the etymological origin of the word "enterprise" follows that it means nothing more than "an undertaking" [78]. Today's understanding is about the entirety of material, personal and idealistic elements that make up a business. Considering this, the term Enterprise Architecture could describe the structured and organized relationship of material, personal and imaginary elements within the system[1] of an institutional organization. As discovered in [88] EA literature definitions vary in the issue of the contribution, the level of maturity of a contribution's focused issue, constituting elements, drivers, and the addressed (and number of) architectural layers. Classifying the usage of the term Enterprise Architecture within this work into these "categories", the main issue of this contribution is to support Enterprise Modeling (precisely: Enterprise Information System Scenario Modeling) with a tool. The level of maturity of this issue can be stated as a conceptual implementation. The use of the term Enterprise Architecture in this thesis comprises the architectural layers information architecture, information systems and –infrastructure and its organizational context. Constituting elements of the Enterprise Architecture term in this document are the relationship between EA

[1] The term "system" in this context means a set of elements, which have attributes and functions. These elements are connected through a set of relationships. A system has a border that divides it from its environment, cf. [51]

elements, AS IS and TO BE models, organizational, as well as technical aspects of an organization. EA drivers are internal, as the main goal of the tool is to support management and cost-reduction. As this work is based on [46], its aim, the application of property assessment criteria on enterprise information system scenarios modeled as enterprise architecture models (see chapter 6), denotes a technical and systemic understanding of the EA term.

4.2 Models

Enterprises grow as time goes by, technology advances and managements take new strategic approaches. Because of this, the architecture of an enterprise evolves as new systems are added to an existing environment but old ones are often kept for compatibility reasons. Growing environments get more and more complex and it gets more challenging to maintain an overview.

As described in the previous chapter enterprise architecture is the discipline to monitor, analyze, and plan the IT environment of a business. Many of the enterprise architecture frameworks that were already described consist of modeling techniques. This is also shown in [40]. The reason for the need of models can be seen while having a look at the definition of a model as given by [94]. Three characteristics of a model are defined. First, a model is a mapping of a real object. This means that characteristics of the original should be visible and preserved in their core character in the model. Second, a model should be a shortening of an original. This is necessary because models are used to simplify the original and keep it to a manageable size. This is especially true in the environment of enterprise architecture as the IT landscape evolved and grew over time. The third character of a model is the character of pragmatism. This means that a model is always created to serve a certain aim. As there are multiple properties of an enterprise architecture that can be investigated this is one of the most important characteristics of models.

Traditionally models play an important role in architecture. For many years, architects created construction drawings and plans for building workers to erect a building. In addition, small-scale models of buildings, which are made of wood or paper, are a commonly used method to show the architects' vision to his investor. This shows how different models of the same original, the building, are created for different aims of the architect.

In the field of computer science there are also a multitude of models in existence. Especially in software engineering, the UML specified by the OMG [77] is a widespread approach of modeling different perspectives of a software system. In software engineering models create a specification for the developer and serve as common understanding between engineer and customer. Secondly, models also serve as documentation of the development process [58].

Enterprise architecture can be seen in direct relation to the architecture of buildings described in the previous paragraph. So the need for models in enterprise architecture is evident and it is "an important mission of enterprise architecture to provide useful models for the various decision-making activities […]" [40].

Since models are a fundamental part of enterprise architecture, it is necessary to maintain a very high quality of the models. Models that look good do not guarantee high quality of the content, as the modeled situation could be observed wrong. Therefore, it is possible to make a good model of a bad situation. On the other hand, it is possible to create a model of a very good observed actual situation, which is not understandable by others than the modeler himself. Standards and modeling conventions help to maintain the quality and readability of models. They make sure that different modelers share a common understanding of the model and help prevent the creation of defective models [53]. Naturally, the experience of the modeler also has high influence on the quality of a model, as his power of observation is superior. Both features of high quality models can be accomplished in part by using specialized modeling software. This software makes it possible to stick to modeling conventions by assisting the user. In addition, knowledge of experienced modelers can be integrated as guidelines into the software.

4.3 Scenario

A scenario is an

"Internally consistent verbal picture of a phenomenon, sequence of events, or situation, based on certain assumptions and factors (variables) chosen by its creator. Scenarios are used in estimating the probable effects of one or more variables, and are an integral part of situation analysis and long-range planning. The name comes from a script used in film/television industry that contains all the details on the appearance of characters, scenes, and the sequence of episodes." [6]

In this thesis and in the context of enterprise architecture analysis the word scenario, often used in conjunction with information system (scenario), means a possible information system architecture state (in the future). Information system scenarios, modeled as architecture models, are usually variations of the as-is model of an enterprise's information system landscape. The model representations of these scenarios (based on a meta-model) contain all relevant elements, attributes, and relationships to visualize and assess it.

5 Information System Decision Making

This chapter justifies the need of decision making within IT governance. The process of decision-making is explained. Hereby the focus is set on information system goals. These goals are differentiated from business goals and IT organization goals. After the process description, several information system goals are presented. A classification of these objectives is presented. Besides this categorization, each goal is analyzed and characterized in detail.

5.1 Decision Making within Enterprise Architectures

The making of rational decisions concerning information systems in an organization should be supported by enterprise architecture models and by conducting analyses on these models [41].

With this approach, the enterprise architecture models build the foundation of the decisions made afterwards. The quality of the models determines the quality of the choices; therefore, they have to be as good as possible. This is somehow unsatisfying, as thus far it has not been clearly defined what constitutes a "good" enterprise architecture model [46].

As IT decision making is an important aspect of IT governance, the stakeholders have to be considered [91]. One of the most common governance frameworks COBIT (Control Objectives for Information and related Technology) presents 19 different stakeholders [29]. These different parties are likely to have distinct requirements for an enterprise architecture model. Thus, the quality of a model depends on the model's ability to fulfill its intended use. In case the analyses are doable, the model serves its purpose. The more the evaluation criterion matches to the model the better the quality of the model is in this certain inquiry [45].

Decision-making can be considered as a process consisting of several activities [65]. At first, the goal needs to be set. There are three goals that can be considered in the context of enterprise information systems management [42] - business goals, information system goals, and IT organization goals.

Business goals are improved business process efficiency, improved product quality, improved internal communication, as well as a range of other goals. These goals can be found on a very high level or even on the highest level as they address the organization as a whole.

On a lower level, the information system goals can be found. These goals abstract away from the enterprise and focus on the information system. An overview of several information system goals is presented in the following paragraph (5.2). A comparison between several systems takes place based on examination of a certain system property, such as availability, security, or usability.

Lastly, IT organization goals are set, in case the operational measurements inside an enterprise could be improved. For example, this could be an optimization of fields of responsibilities with the objective to be able to react quicker in case of a system failure.

In the following, the focus is set on information system goals regarding the problem definition of this thesis. The typical questions, which have to be answered by a decision maker, are "What is feasible?", "What is desirable?" and "What is the best alternative according to the notion of desirability, given the feasibility constraint?" [87, 32, 47]. Within enterprise architecture analysis, these three questions can be responded to in three corresponding ways. "What is feasible" can be answered by different scenarios.

"Answers to the second question regarding desirables, are often expressed in terms of the change of various information system properties such as increased information security, increased interoperability, increased availability, etc." [47]

Enterprise architecture analysis itself answers the third question as it acts as a mediator by consideration of different scenarios under a certain system property aspect.

Having specified a goal (in this case an information system goal), different scenarios have to be drafted, as they represent the different options that can be selected, to fulfill the desired goal. This sometimes turns out to be more difficult than expected, as distinctions between two scenarios might not be obvious [42].

Unfortunately, enterprise architectures are often very complex systems. They are not easy to understand and some internal dependencies are not obvious or require expertise to be detectable by a decision maker. Therefore, it is not easy to identify the connection between a certain decision and its influence on the goal, especially as the fulfilling of a goal requires more than one decision. It is essential for the decision maker to identify chains of interdependencies inside the enterprise architecture. A goal needs to be broken down into several measurable objects for a detailed understanding of its properties and structure [55]. With this knowledge, he is able to identify the effects clearly that are related to his decisions.

Before decisions can be made in the last step, information on the considered system needs to be gathered. This has to be done to see how chains of interdependencies, which were found before, were reflected in practice. It sometimes happens that the information found is not completely compatible to the breakdown of the goal performed earlier. No one-to-one mapping can be done. In this case the decisions are based on incomplete information [42]. Data collection, especially if humans are involved, is a complicated process. It has to be clarified, in which way the gathered information is credible. Therefore, the collected information is associated with a degree of uncertainty.

Finally, a decision maker is able to make good decisions when he is supported by a credible analysis of the effects of the decisions on his goals. Simplystated, the option that has the best effect on the goals can be chosen [42]. Here it should be stressed that the different stakeholders, which were mentioned above, influence the decision. As a diversity of interests has to be considered, lobbying and negotiation become a large part in decision-making.

"[...], such lobbying and negotiation oftentimes concerns whether the correct goal has been selected, whether the identified decision alternatives are reasonable, whether the causal theory linking the decisions to the goals are really correct, and whether the collected information is credible or not." [42]

5.2 Information System Goals

In the following, an overview of several system qualities is presented. In a scientific context, system qualities can also be defined by the term system properties. These properties are examples of information system goals. Often interdependencies between several systems exist, which are sometimes not obvious. The system qualities presented here, are very important for enterprise architecture analysis as their decomposition provides the decision maker with several measurable objects that have to be

considered during scenario creation. The following system qualities are taken from [105] and [43].

Their structure can be identified in the following figure:

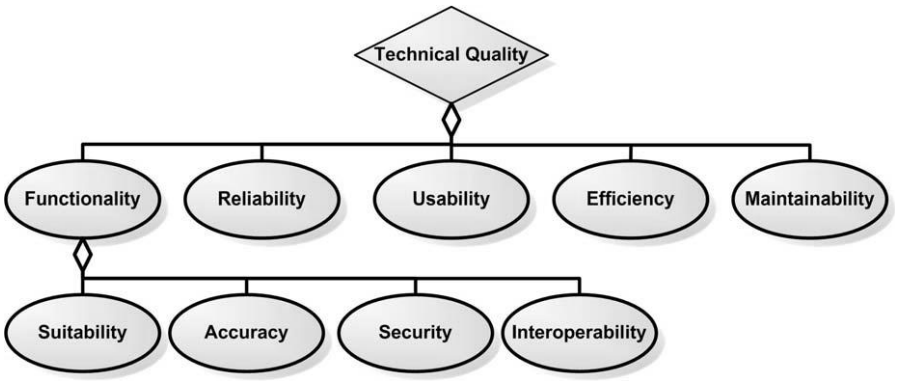


Fig. 1. Components of System Quality [68]

Availability: "The availability of a system is the probability that it will be up and running and able to deliver useful services at any given time." [92]. Availability is defined as the ratio between the system's time in service and the total time, i.e. uptime plus downtime. Availability is also reffered to as reliability (see *Fig. 1*).

Performance:

"The capability of the software product to provide appropriate response time, processing time and throughput rates when performing its function under stated conditions. It is an attribute that can be measured for each functionality of the system." [57]

Performance defines how much work a system can perform and how fast it does the work. This system property is also described as efficiency [33].

Interoperability: Interoperability describes the "the ability to interact with one or more specified systems" [57]. Interoperability has to be separated from integrability and replaceability [43]. Interoperability itself is a subproperty of functionality.

Security: Security can be expressed by the ability "[...] to prevent unauthorized access to programs or data." [57] "Security itself includes integrity and confidentiality, whereas it is another subproperty of functionality." [92].

Usability: "This property reflects how easy it is to use the system. It depends on the technical system components, its operators and its operating system." [92]. Usability subsumes the characteristics understandability, learnability and operability [57] as well as attractiveness [43].

Accuracy: Accuracy is defined as the ability "to provide the right or agreed results or effects with the needed degree of precision." [57]. In case the real output of a system is consistent to the expected output, the accuracy of the system is high. Accuracy is also a part of the functional abilities of a system.

Maintainability: Maintainability means the capability of a system to be changed and adopted Modifications may include corrections, improvements or adaptations of the system based on changes in the environment and in the requirements and functional specifications [57]. As enterprise architectures get more complex, there maintainability gets more important. Therefore maintainability has been subdivided into flexibility, reusability, extensibility, portability, and integrability, to be more precise [43].

Functional Suitability: Functional Suitability is the ability to meet the user's expectations about functionality. This is very specific for the domain the system is intended to support [43].

6 Enterprise Architecture Analysis

"Enterprise architecture analysis is the application of property assessment criteria on enterprise architecture models." [46] In this work, enterprise architecture analysis is seen as the analysis of architectural models of the enterprise information system. EAT is a tool for the creation and assessment of enterprise information system scenarios (in the form of enterprise architecture models, see 4.3), which are descriptions of the systems and their environments. According to [45], architecture analysis supports (information system) decision making (see 5.1) which usually runs through the phases: "formulate scenarios", "decide upon evaluation criteria", "analyze scenarios", and "select scenarios". Scenario formulation can be done with EAT. These scenarios are mostly modifications of the as-is model and include a new or changed system proposed to be integrated in the future. Then evaluation criteria are necessary to provide relevant answers to the CIO's questions and to choose appropriate scenarios. For example, these criteria can be availability, security, or performance. In the analysis phase, appropriate scenarios are assessed against the chosen evaluation criteria. This phase is supported by EAT. Having done a quantitative assessment, the CIO has the option to realize the scenario that was evaluated best.

The overall process of enterprise architecture analysis, referring to [46] can be found in the following subsection.

6.1 Method for Enterprise Architecture Analysis

As mentioned before, enterprise architecture analysis is to support decision making in the field of enterprise information systems. The method to apply enterprise architecture analysis on enterprise architecture models with EAT is illustrated in *Fig. 2*.

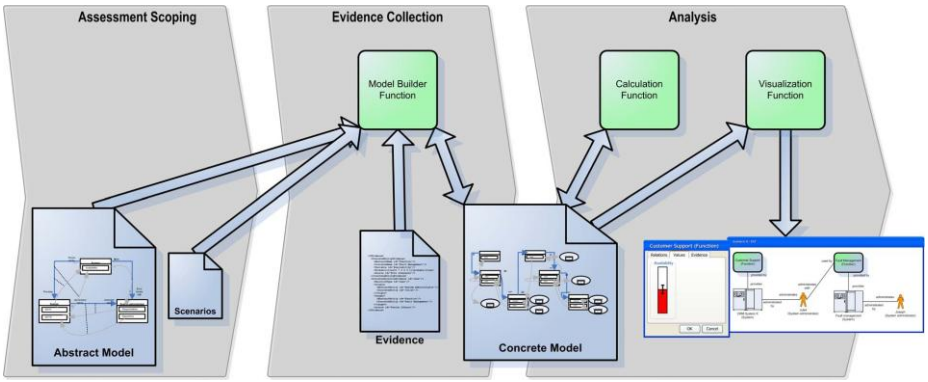


Fig. 2. Process of enterprise architecture analysis

As one can see, the process is divided into three steps. In the first step, called "Assessment Scoping", the problem for the decision maker needs to be formalized with different scenarios (4.3 Scenario), representing potential different future states of the enterprise's information system. The scenarios focus on assessment criteria (cf. 5.2), where they will be evaluated against (qualitative attributes). These scenarios are modeled as abstract models, which are "…enterprise architecture Metamodel[s] augmented with the causal links …" [46] between qualities.

The second step, "Evidence Collection", is to instantiate appropriate abstract models (scenarios) with concrete information. The relevant evidence has to be collected manually. The generic elements of the abstract model are then instantiated into concrete models representing a specific (future) enterprise information system environment. The instantiation of a concrete model out of an abstract model can be seen in the same way that an object is an instantiation of a class in object-oriented programming languages.

The last step, the "Analysis", is to calculate quantitative values of the model's assessment criteria (qualitative attributes). Further information about the functionality of a Bayesian network can be found in chapters 6.3 and 6.6. The creation of a Bayesian network from a concrete model is described in detail in chapter 8.3.4. The result of the analysis, quantitative assessments of the concrete models, supports the decision maker in selecting and realizing the appropriate scenario.

6.2 Probabilistic Influences and Uncertainty

In 5.1 it was explained that a decision maker has to answer three different questions ("What is feasible?", "What is desirable?" and "What is the best alternative according to the notion of desirability, given the feasibility constraint?"). Thereby, "What is feasible?" the question which compares different scenarios, can be replied by a Bayesian network [47]. Bayesian networks, which are explained in chapter 6.3.2 have abilities to model the real world. A feasible scenario can be translated into a network, which can be calculated. The dependencies inside a scenario are represented as rela-

tionships in its corresponding network. During enterprise architecture analysis, it has to be considered that most of these relations are probabilistic rather than deterministic in nature.

Even though a certain behavior or reaction is likely, the expected outcome might not happen in each possible situation. Especially when it comes to decisions, this not-determinism has to be considered. A decision maker would like to compare systems based on deterministic dependencies, as their behavior can be predicted exactly. In [47] it was discovered that "Decision making, whether the decisions apply to IT or not, is rarely performed under conditions of complete certainty". Therefore, probability has to be considered to compensate for uncertainty and missing information, when analyses are performed.

The authors of "Enterprise Architecture Analysis with Extended Influence Diagrams" describe several types of uncertainty. Especially in an enterprise architecture analysis context, causal uncertainty is a wide spread phenomena. This term describes that it is normally not possible to describe how the world actually behaves. The dependencies between several phenomena are oftentimes not obvious and exact predictions are seldom. An example of this type of uncertainty given by the authors is: "uncertainty with respect to the causal effect the percentage of systems with updated virus protection may have on the level of information security". Another example is, the uncertainty how much the reliability is improved, when a second backup-server is installed.

The other considerable type of uncertainty is called empirical uncertainty. The phenomenon described with this concept is that all the gathered information has to be scrutinized. "The information was collected a while ago and has now become obsolete, or perhaps the information was gathered from a source that might have been incorrect" [47]. In cases where the system administrator describes his abilities as good, they are probably good indeed. Eventually the administrator did not tell the truth or good in his eyes does not have the same meaning than the term good has for the enterprise he is working at. This situation has to be considered in an analysis. When information is gathered, contextual information about the evidence should also be gathered [46]. Knowledge about the age, the source and the reason why this information was given helps to estimate the evidence's credibility. Thereby the credibility of the assessment as a whole can be defragmented from the credibility of its parts.

Besides the possibility of getting knowledge about the credibility from a bottom-up approach deriving credibility is also practical - in some situations it's not feasible to directly collect evidences about properties such as availability and maintainability whereas it is possible to collect evidence pointing in a certain direction (e.g. whether the system administrator is experienced or not).

6.3 Mathematical Modeling using Bayesian Networks

This chapter provides information about the mathematical background of this thesis. First an introduction into Bayesian networks is given. This introduction ends by men-

tioning the main advantages of these networks. How they are relevant and can be used within an enterprise architecture analysis context is presented next. This part is followed by a detailed explanation of the underlying mathematical principles, which are used during the calculation of Bayesian networks. For this introduction, an example of the enterprise architecture domain is considered. Afterwards a mechanism to reduce the complexity within mathematical networks is presented. It necessity is motivated with use of a small enterprise architecture analysis scenario. Finally, this chapter introduces Probabilistic Relational Models, which are an extension of Bayesian networks, to increase their abilities.

6.3.1 Introduction into Bayesian Networks

Bayesian networks (BNs) belong to the family of probabilistic graphical models (GMs). In a BN, knowledge about an uncertain domain is represented in a graphical structure. A scenario is translated into a graph; thereby each node (consisting of several states) represents a random variable, whereas the edges represent probabilistic dependencies of corresponding random variables. This way "webs" of causes and effects are created [18]. Static computational methods help to estimate the conditional dependencies inside the graph structure. Bayesian networks combine principles of graph theory, probability theory, computer science, and statistics [2].

The networks correspond to directed acyclic graphs (DAGs), which is another GM structure. DAGs are very popular in machine learning, artificial intelligence, and statistics. A Bayesian network enables "an effective representation and computation of the joint probability distribution (JPD) over a set of random variables" [80]. The character of the BNs ensures that no node can be its own ancestor or its own descendant. Even though a directed graph is considered reasoning of the BN can be performed in any direction by propagating in all directions [2, 79].

Usually the DAG structure is considered as the "qualitative" part of the reflected model. The "quantitative" characteristics are described in a manner which is consistent with a Markovian property. In this case, the conditional probability distribution (CPD) at each node depends only on its parents in the graph structure.

If discrete random (as in the tool, presented in this thesis) variables are considered, the conditional probability is often represented by a table, listing

"[...] the local probability that a child node takes on each of the feasible values – for each combination of values of its parents. The joint distribution of a collection of variables can be determined uniquely by these local conditional probability tables (CPTs)." [2]

The resulting network has good capabilities for the rigorous quantification of risks and the clear communication of results [18]. BNs can combine historical information that can be processed by iterative calculation of the network, as well as data with expert judgment. Another important aspect of Bayesian networks is the fact that new information about a certain state of a variable influences the states of all other variables. This is the so-called propagation of BNs, which is for instance used in artificial intelligence.

Several advantages of Bayesian networks were discovered by the authors of [18]. In a case study of Norman Fenton and Martin Neil, these advantages were verified in practice [17]. In the following part of this section, these advantages should be considered, with focus on their relation to enterprise architecture analysis:

- Explicitly model causal factors: During the decision-making within enterprise architecture analysis, dependency chains have to be found and modeled. These chains represent the causal structure of a scenario and therefore the graph of the network, as a mapping is performed during the analysis (cf. 6.2).
- Reason from effect to cause and vice versa: In an enterprise architecture analysis, the reasoning from effect to cause is obvious. This is done by following the architectures dependency chains during the calculation. The other way is taken when uncertainty is considered.
- Overturn previous beliefs in the light of new evidence: As the authors of this collection of advantages describe "The notion of explaining away evidence is one example of this." This example fits perfectly into enterprise architecture analysis, as it might happen that evidence gets obsolete because of the gathering of newer information.
- Make predictions with incomplete data: Often during analysis, not all information can be gathered. This might happen because of several reasons (e.g., collection of evidences is too time consuming, too expensive, not possible because no expert is available). Therefore, it is very important for the decision maker that reasoning can be performed, even though there remaines uncertainty during the information gathering process. Even if no knowledge about the inputs of a node in a BN is available, this can be compensated, as the considered node acts as a prior.
- Combine diverse types of evidence including both subjective beliefs and objective data: In enterprise architecture, both subjective beliefs and objective data occur. Human influences are especially difficult to measure, as humans tend to behave in ways other than expected. Therefore, they are assessed very subjectively, depending on the interviewer, the interview situation and the person who is interviewed.
- Arrive at decisions based on visible auditable reasoning: In the context of enterprise architecture analysis, this advantage means that no hidden variables are left. In case the dependency chains have been collected, they were translated in a BN. This network is a 1:1 mapping of the real world dependency chains. All nodes in the net have a foundation in the enterprise architecture. Bayesian networks are a long-established theory of science; therefore, their computation is very approved and reliable.

6.3.2 Explanation of Bayesian Networks

Having presented a general introduction on Bayesian Networks in the previous paragraph, this section explores the mathematical background of a BN. Therefore a small example (an extract of the scenario presented in [46] is considered).

The following situation is to be analyzed: An administrator is occupied in a certain enterprise. Her name is Juliet and her talents of experience and responsiveness (depending on the availability of the fault management system, which she uses) are important for this scenario. She is in charge of a CRM (Customer Relationship Management) system. Juliet's abilities have influence on the availability of the system. This availability is necessary to have the customer support of the enterprise available. The customer support also depends on the reliability of the CRM system. In *Fig. 3*, the described scenario is modeled in a DAG.

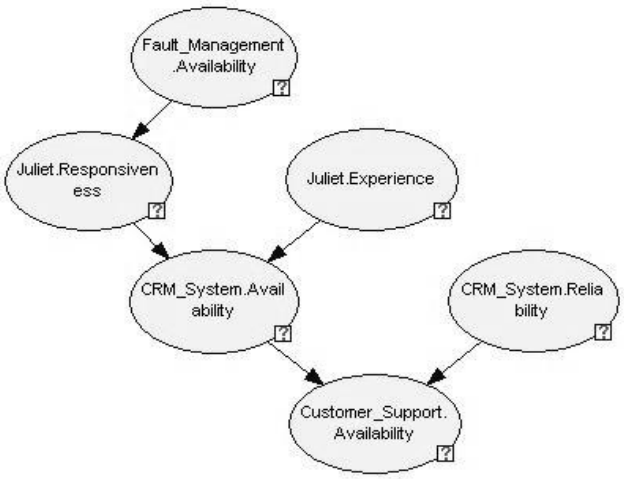


Fig. 3. The Bayesian network representing the scenario

In this scenario six variables are considered, namely *Fault_Management.Availability (FMA)*, *Juliet.Responsiveness (JR)*, *Juliet.Experience (JE)*, *CRM_System.Availability (CSA)*, *CRM_System.Reliability (CSR)*, and *Customer_Support.Availability (CuSA)*. Here they may assume values from the finite domain {*High*, *Medium*, *Low*}.

Table 1. Probabilities for the states of Fault_Management.Availability

FMA	High	0.9
	Medium	0.1
	Low	0

Table 1 shows the probabilities for the different possible states of the *Fault_Management.Availability*.

Since the *Fault_Management.Availability* influences *Juliet.Responsiveness* (as it is its parent), *Fault_Management.Availability* has a conditional probability distribution of *Juliet.Responsiveness*. This is visualized in *Table 2*. Conditional probability table of the variable *Juliet.Responsiveness* conditioned on the variable *Fault_Management.Availability*.

Table 2. CPT for Juliet's responsiveness (JR) conditioned on the availability of the Fault_Management (FMA)

FMA		High	Medium	Low
JR	High	1	0	0
	Medium	0	1	0
	Low	0	0	1

Before the consideration of the scenario is continued, it is necessary to have a closer look at the mathematical theory of a BN. Especially the naming has to be explained.

In a finite set of discrete random variables V, where each variable $X \in V$ is denoted as a capital letter, e.g., *X, Y, Z*. Each state of a variable is denoted as a corresponding lowercase letter, e.g., *x, y, z*. The set of all states, the so-called domain, within a variable X, is denoted as DX. The probability distribution over a random variable X is denoted as $P(X)$ and the probability of a state $x \in DX$ as $P(X = x)$ or in shorter form $P(x)$. The negation of a state x is denoted as $\bar{x}$ and represents all the states apart from the state x in the variable. The probability of the negation, $P(\bar{x})$ is always equal to $1 - P(x)$.

In EAT the probabilities for a certain variable, which has no parents are collected within interviews or other ways of information gathering. This variable A is called prior variable, with its belonging prior probability $P(A)$. Whereas the information about the probabilities of a node inside the network (a child node, also called posterior) are calculated. If B is a parent of variable A this is described as $P(A|B)$. Hereby it has to considered, that all parents influence their common child. "If variables are independent of each other, the posterior probability and the prior probability are equal, $P(A|B) = P(A)$." [34]

Also all permutations of states of the involved variables may occur (except the situation that the probability of a certain state is exactly zero). This is visualized for the *CRM_System.Availability* in *Table 3*:

Table 3. Nine possible combinations of states

CSA_high & JR_high & JE_high	CSA_high & JR_medium & JE_high	CSA_high & JR_low & JE_high
CSA_high & JR_high & JE_medium	CSA_high & JR_medium & JE_medium	CSA_high & JR_low & JE_medium
CSA_high & JR_high & JE_low	CSA_high & JR_medium & JE_low	CSA_high & JR_low & JE_low
CSA_medium & JR_high & JE_high	CSA_medium & JR_medium & JE_high	CSA_medium & JR_low & JE_high
CSA_medium & JR_high & JE_medium	CSA_medium & JR_medium & JE_medium	CSA_medium & JR_low & JE_medium
CSA_medium & JR_high & JE_low	CSA_medium & JR_medium & JE_low	CSA_medium & JR_low & JE_low
CSA_low & JR_high & JE_high	CSA_low & JR_medium & JE_high	CSA_low & JR_low & JE_high
CSA_low & JR_high & JE_medium	CSA_low & JR_medium & JE_medium	CSA_low & JR_low & JE_medium
CSA_low & JR_high & JE_low	CSA_low & JR_medium & JE_low	CSA_low & JR_low & JE_low

The probability of a child is defined by the joint probability over the parental states in the scenario. The set of parents of a variable X is denoted as πX. The calculation of a BN is founded on Bayes' rule:

$$P(B|A) = \frac{P(A|B)P(B)}{P(A)}$$

Applied to the scenario described earlier the following instantiation can be made:

$$P(FMA|JR) = \frac{P(JR|FMA)P(FMA)}{P(JR)}$$

After a short transformation, the probability of *Juliet.Responsiveness* can be calculated:

$$P(JR) = \frac{P(JR|FMA)P(FMA)}{P(FMA|JR)}$$

Bayes' rule, as it was described above, is only applicably for simple network structures, where a parent node has one or more children. For the calculation of more complex BNs, it is necessary to introduce the chain rule.

Chain rule: Let BN be a Bayesian network over a finite set of discrete random variables $V = \{V_1, ..., V_n\}$. The joint probability distribution P(V) is then:

$$P(V) = \prod_{i=1}^{n} P(V_i|\pi_{Vi})$$

In the considered scenario, an application enables the calculation of all state-probabilities of all nodes, no matter where they are located. Consider the following exemplary application, corresponding to the network described in advance:

$$P(FMA, JR, JE, CSA, CSR, CuSA) =$$

$$P(FMA)P(JR|FMA)P(CSA|JR, JA)P(CuSA|CSA, CSR)$$

This way, the Bayesian network can be calculated and the scenario can be evaluated. Especially the comparison of the probabilities of the states of the *Customer_Support.Availability*, which is the drain node, put the decision maker into a position to decide which scenario should be implemented. As its goal is typically to ensure the highest possible availability, he is likely to select the scenario whose *Customer_Support.Availability_high* state is the highest in his evaluation.

This example showed how Bayesian networks could be calculated. The proceeding of those networks does not require a special structure. Therefore, the inference on Baysian networks presents itself to be used whenever a fast transformation from a model to a network is possible, as it is the case in the approach described in this thesis.

6.3.3 Complexity Reduction in Bayesian Networks

In 6.3.2 the use of Bayesian networks to support decision-making was explained. For simple scenarios, such as the scenario presented, the described proceeding ensures a fast and simple computation of the considered networks.

Unfortunately in the case of more complex structures, which means that the number of parent nodes and relationships between nodes increases, the time to perform inference and the number of variables that have to be saved rises in an unpropitious way. “It is well known that the problem of computing a conditional probability in a probabilistic network is NP-hard” was noticed by Michael P. Wellman and Chao-Lin Liu in [107].

During the enterprise architectures analysis, a solution has to be found to increase the inference performance by decreasing the needed computations. If the number of variables that are saved is reduced, fewer computations have to be made, as the number of dependencies between the variables is reduced indirectly. This is necessary as enterprise architecture scenarios can typically get very complex consisting of many nodes and links between them. The following example illustrates in a simple scenario the number of calculations that need to be done. Here three CRM systems influence the availability of the customer support (*Fig. 4*).

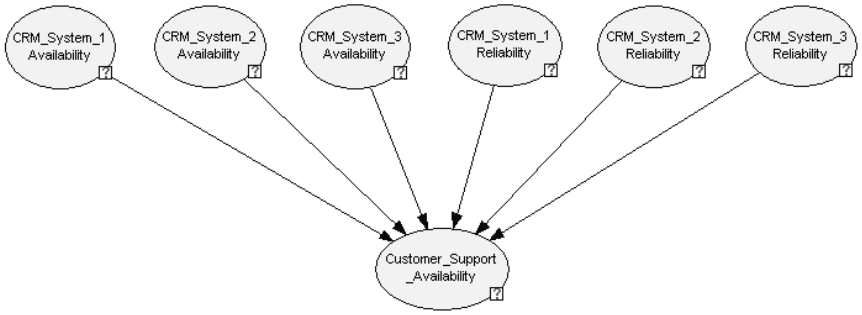


Fig. 4. Scenario in which many variables need to be saved

In 6.3.2 it was explained that each variable holds three different states {*High, Medium, Low*}. This means that for the computation of *Customer_Support_Availability* $3^7 = 2187$ variables are stored and considered.

Even though the network consists of only seven nodes this number is very high. The theoretically extension of the scenario with a fourth CRM system exposes how the number of considered variables increases tremendously. Now $3^9 = 19683$ variables have to be inferred.

The solution to this vast amount of calculations is called model structure abstraction [107, 63]. This means that the BN is approximated by another network, which can be inferred with a reduced number of calculations.

In this thesis, the approximation is done by introduction of additional nodes, to reduce the absolute number of parents, each node has. This means that nodes, which

have a common structure, are collected and processed, by an intermediate node. The outcome of the intermediate node is connected to the original child node of the collection. The intermediate nodes have no correspondent in the real word and are added for mathematical reasons. This is demonstrated in the following scenario (*Fig. 5*).

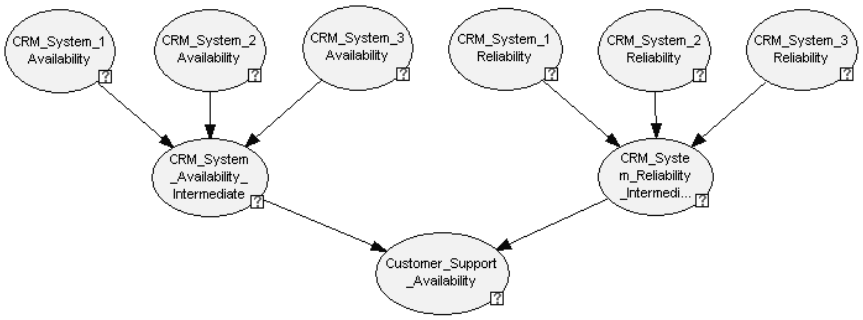


Fig. 5. Scenario in which fewer variables need to be saved

The introduction of intermediate nodes reduces the number of needed calculations. For each new node $3^4 = 81$ variables are saved, whereas at the *Customer_Support_Availability* node only $3^3 = 27$ variables are considered. All in all $3^4 + 3^4 + 3^3 = 189$ computations need to be done.

This example illustrates that a small adoption of the network leads to remarkable improvements. Especially when this transformation is used frequently, speed increases can be expected.

6.3.4 Probabilistic Relational Models

Probabilistic Relational Models (PRMs) are an extension of Bayesian networks but incorporate a much richer relational structure [84].

"In a way, PRMs are to BNs as a set of rules in firstorder logic is to a set of rules in propositional logic: A rule such as ∀ x,y,z. Parent (x,y)∧ Parent (y,z) => Grandparent (x,z) induces a potentially infinite set of ground (propositional) instantiations." [25]

PRMs specify relational schema for a domain. Attributes with entities exist in each domain. Between these attributes, probabilistic dependencies can be found. In addition, a probability distribution over the database is specified. "These models represent the uncertainty over the properties of an entity, capturing its probabilistic dependence both on other properties of that entity and on properties of related entities." [50]

In *Fig. 6*, which can be found at [11], a small PRM is symbolized. Before the explanation of PRMs is continued, this image should be discussed, to establish understanding of Probabilistic Relational Models.

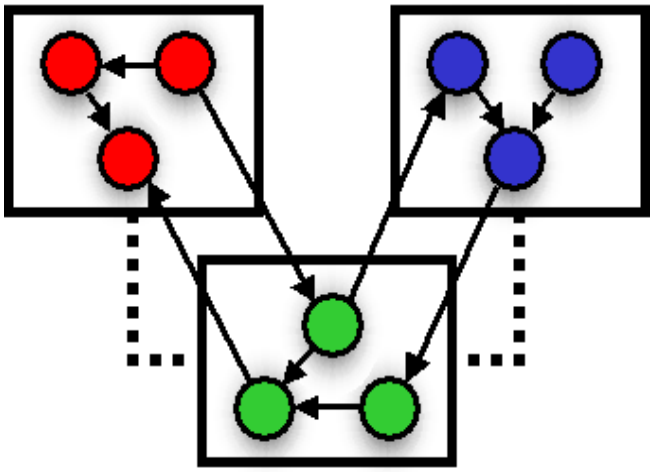


Fig. 6. An exemplary PRM

Fig. 6, which was published by some of the most important PRM experts (L. Getoor, N. Friedman, D. Koller, and A. Pfeffer), considers three domains (colored red, blue, and green). The red and green domain, as well as the green and blue domains is related. Three entities with attributes exist for each domain. Encore, between these attributes, relationships exist. It has to be stressed that not only the attributes of a certain domain are connected. There exist also relations between attributes of two independent domains.

Relational schemas are used to describe databases. Often enterprise business information, CRM related datasets or scientific data is stored in a relational database. These schemas are modeled in Entity-Relationship Diagrams (ERD) [83]. A relational modeling of the considered domain helps to understand complex dependencies. Especially when it comes to enterprise architecture analysis, it is an advantage to have a widespread and easily understandable notation. Multiplicities, which are heavily used in relational databases, allow expressing scenarios very detailed. This way, it can be expressed how many entities of one domain are related to entities of the corresponding one. This makes it easier to visualize and discuss a specific scenario. *Fig. 7* below visualizes a simple PRM in an enterprise architecture analysis specific domain:

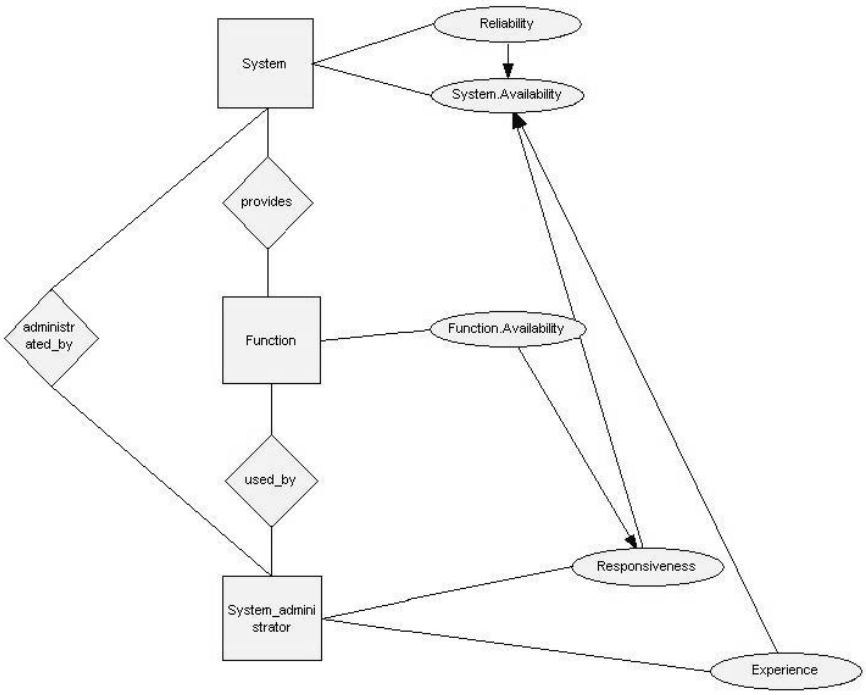


Fig. 7. PRM visualized in an entity-relationship (type) diagram

The entities function, system, and system administrator are considered. They are related in an ERD. Each of these entities has one or more attributes. The attributes are related no mater to which entity they belong. It is comprehensible that a Bayesian network can be deduced from their relationships. The calculation of this network allows a scenario evaluation.

Besides the use of relational Schema, the other important introduction, which is made with PRMs, is slot chains. This is a special connection mechanism for creating models that are more complex.

"Finally, we define the notion of a slot chain, which allows us to compose slots, defining functions from objects to other objects to which they are indirectly related." [26]

If there is a relationship from entity A to entity B and from entity B to C than a connection from A to C exists in an indirect way. In this example, one could say that there is a slot from A to B; attributes of B (the domain) are mapped to attributes of B (the range). With the use of slot chains, a function from A to C is indirectly introduced. The attributes of A (which remains the domain) are mapped on attributes of C, through a use of the slot from B to C, applied on the result of the slot from A to B.

The combination of the slots and the multiplicities from the relational schema allows using aggregation functions between the domains. An attribute of all entities of a

domain can be combined with a slot and its result can influence all the range-attributes.

"More formally our language allows a notion of an aggregate slot. The slot takes a multiset of values of some ground type and returns a summary of it […]" [25]

Examples for this aggregation functions are median, maximum or minimum functions. They all have in common, that the number of domain attributes is not restricted, whereas in each case only one output element will result.

There exist several algorithms to infer on PRMs. As PRMs can be broken down into Bayesian networks, which can be done by building a network based on the attributes that are related, one way to calculate PRMs is to infer on the underlying BN.

6.4 Abstract Model

Chapter 6.1 describes a method for enterprise architecture analysis. The first step is to define the domain of the assessment in correspondence to the relevant system goal (5.2). This is done by creating a so-called abstract model.

An abstract model consists of four different components: entities, attributes, entity relationships and attribute relationships. These four components can be used to depict the assessment scope of the current analysis graphically. The graphical appearance is based on notation of class diagrams of the Unified Modeling Language (UML) [77].

Entities describe the central parts of the model. They can be derived from real objects, such as "person" or "system". Also a more abstract level of entities is possible, such as "function" or "process", which have no physical real world counterpart. Entities are depicted like classes in a class diagram as a rectangle with the name of the entity at the top of the box and a line separating it from the rest of the box. Entities in the abstract model can be considered as domains within PRMs (see 6.3.4).

The connection of two entities can be displayed using an entity relationship. An entity relationship appears as line between two entities. The role names are annotated in accordance to the UML on both ends of the relationship. In addition, the multiplicities are shown on the ends of the relationship. These multiplicities determine whether dynamic or static CPTs are used during the creation of the Bayesian network. A detailed description for this influence can be found at 8.3.4.

Attributes are always parts of an entity and are therefore drawn inside the lower part of entity-rectangle. They may assume values from a finite domain like {True, False}, {High, Medium, Low}, {0-10}, etc.

The last component of an abstract model is the attribute relationship. Relationships between attributes show the way of influence between the attributes. A change of the values of the source attribute implies a change of the values of the target attribute. Each attribute relation is connected to a particular entity relationship by a line between the relationships. To accommodate the approach described in 6.3.4, it is also possible to connect two attributes by using an indirect connection constituted by several entity relationships. Translations of the term attribute and attribute relations into

the language of Bayesian networks would be node and their connecting relationships. The indirect connections are realized via the PRM slot chain mechanism (cf. 6.3.4)

The abstract model enables one to model influence dependencies between different entities, for example, "the experience of the system administrator may affect the availability of the system he or she administrates." [46]. These influences are described by a conditional probability table as described in 6.2. The CPTs have to be specified for each attribute that is dependent on other attributes. They represent the degree of influence of each parent attribute on the current attribute. Attributes that have no parent attributes have to be provided with priors. Priors represent source nodes in the resulting Bayesian network. The source nodes are not influenced by other nodes. Therefore, the probability of each of their states can be stated.

Since these CPTs can grow quite large (6.3.3), it is possible to specify a so-called aggregation function for an attribute relationship. These aggregation functions result in the insertion of auxiliary nodes into the resulting Bayesian network. The auxiliary nodes aggregate the influence of several congeneric attributes. In addition, the introduction of Aggregation functions and later auxiliary nodes enables the realization of probabilistic relational models (6.3.4). These functions are a representation of the aggregation functions described in [25].

Information for creating abstract models can be found in specific literature. Another source can be the knowledge of experts or empirical data.

In the following, a simplified example of an abstract model is shown and described to summarize the different model elements (*Fig. 8*). The example is taken from [46].

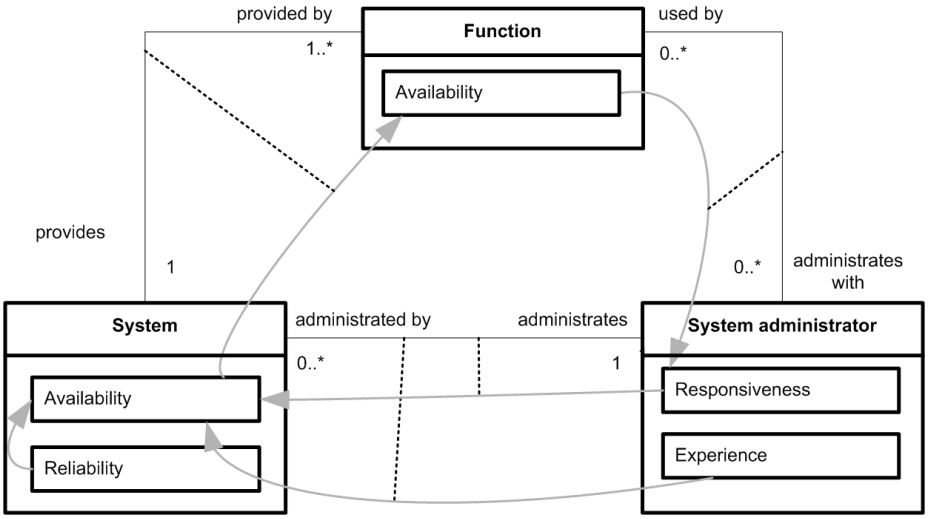


Fig. 8. Example of an abstract model [46]

In *Fig. 8* above, an example of an abstract model is given. Main goal was to determine the availability of a certain function of a software system. The model consists of three entities "Function", "System" and "System administrator". The "Function" is provided by a "System" as depicted using an entity relationship between them. In

addition, the existence of a “Function” that is used by a “System administrator” is possible. Finally, the “System administrator” administrates a “System”.

As the goal was to calculate the “Availability” of a “Function”, the “Availability” is modeled as attribute of the entity “Function”. This “Availability” is dependent of the “Availability” of the whole “System”. The “Availability” of the “System” itself is dependent from the “Reliability” of the “System” and in large parts from the “Experience” and “Responsiveness” of the “System administrator” which are modeled as attributes of the “System administrator”. All these dependencies are attribute relationships and are shown in the figure as gray arcs.

The next step of the enterprise architecture analysis consists of collecting all needed and possible evidence of the model described above (cf. 6.5 Evidence Collection). Although it is not possible to observe or to assess values for all attributes, it is possible to calculate the missing ones by creating a Bayesian network from the evidence and the modeled attribute relationships (cf. 6.2 Probabilistic Influences and Uncertainty).

6.5 Evidence Collection

In order to instantiate a concrete model based on an abstract model, detailed information are necessary. As mentioned in 6.1, the instantiation of a concrete model out of an abstract model can be seen in the same way that an object is an instantiation of a class in object-oriented programming languages. Taking for instance a class `Administrator` which has attributes `name` and `experience` the programmer is better off knowing these specifications when instantiating an `Administrator` object, e.g. `crm_admin`, whose `name` is “Juliet” and `experience` is “high” (assuming that there are no default values and the constructor needs these attributes as arguments). Gathering relevant information for the instantiation of a concrete model is called *evidence collection*. Evidences are “… used to create one concrete model per scenario […] where the generic concepts of the abstract model are instantiated as enterprise-specific (concrete) entities and attributes” [46]. The word evidence in the term evidence collection is used since instantiated variables in Bayesian networks are usually referred to as evidences [34]. According to Johnson et al., there are three types of evidence that can be supplied:

- Evidence on the existence of various *entities*;
- Evidence on the *relationships* between entities;
- Evidence on the value of *attributes* (data collection).

The first two types determine the structure of the concrete model and the last type fills the structure with indications of attribute states so that the quality assessment can be performed. Only evidence on entities, relationships and attributes present in the abstract model is permissible evidence.

Before collecting evidences, considerations have to be made.

“To get the necessary information, the decision-maker needs to know what kind of information to collect and then insure that this information is indeed collected and properly modeled.” [55]

First, as shown in 6.1, assessment scoping is necessary to define the goal of the assessment and to identify appropriate scenarios. Doing this reduces the complete environmental system to a relevant domain of examination. Thus the amount of evidences to be gathered decreases and hence the effort of collecting them declines, too.

Second, the ways of gathering information have to be considered. Usually, the collection can be done by reading documents, performing interviews/manual test, from first-hand experience, or a combination of these [67].
Referring to [31] the collection of type three evidences (data collection) can be done as follows:

- Direct collection (of technical parameters of the actual EA elements; read out log files, etc.);
- Indirect collection (of data bases at distributed locations that allow to draw conclusions about element dependencies);
- User-based estimation of causal dependencies (by querying the users via interview or questionnaire).

According to [31] *"The manner of data elicitation may also depend on the individual collection strategy of a company. Methods one and two usually require additional technical efforts beforehand […]. Method three does not require […] technical efforts. This approach collects relevant conditional probabilities through interviews with e.g. architecture experts, programmers, and system users as well as through analysis of the participating EA elements."*

The collection of evidences not only focuses on the evidence's core informational contents but also on contextual information. For example, the source, the age, and the type of the evidence are gathered, too. This is done to estimate the credibility of evidences because the suitability of sources for providing certain information can differ. For instance, evidences as an outcome of interviews can be colored and subjective. It is obvious that an administrator itself would rate his responsiveness higher than a related employee would. Thus, the credibility of evidences is quantified and processed in CPTs. Moreover the "… evidential credibility is subsequently used to estimate the credibility of the assessment as a whole." [46]

Närman [67] states that there are two ways for determining the credibility of evidences:

- Employ heuristics [38] or
- Use quantitative estimations of the credibility by experts [14, 48]

One example of heuristic simply could be the matching of the interviewee's domain of competence with that required by the questions. The better the matching is the better the credibility. Furthermore, the actuality of the information is an indicator for its credibility, too.

The data collected in the evidence collection process is crucial for the process of analysis as conditional probability distributions are derived from the evidences. Having collected data for all CPTs (and hence specifying the Bayesian Belief Network) enables reasoning (calculation process).

6.6 Concrete Model

The abstract model and the collected evidence together constitute a concrete model. The concrete model can be understood as an instantiation of the abstract model like an object diagram is the instantiation of a class diagram in the UML.

As the abstract model serves as Meta model for the concrete model a concrete model can only contain instances of entities and relationships that have been modeled in the abstract model before.

While the abstract model is about general correlation of objects or concepts, the concrete model is about the concrete real world instance of the object or concept. "The concrete model […] specifies which system and what administrator are under consideration […]" [46].

Most of the time more than one concrete model of an abstract model exists. This is, because each concrete model is the mapping of one scenario from which the decision maker wants to choose. Different scenarios do not necessary have to consist of different types of evidence, but rather can differentiate in the values that are assumed for them.

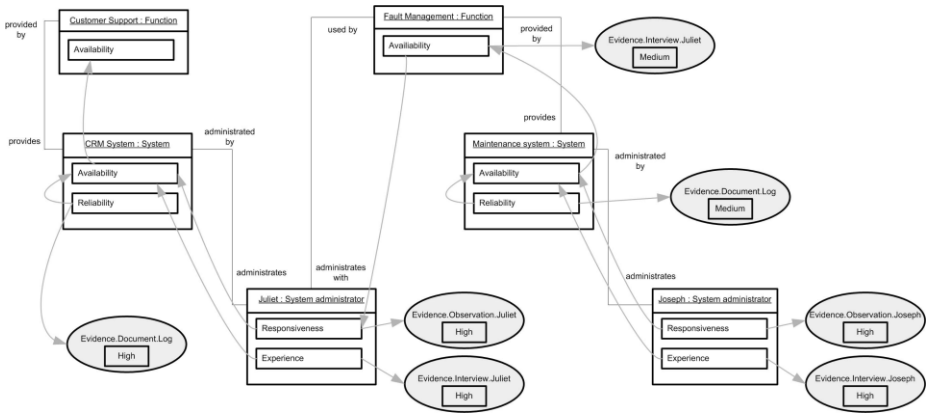


Fig. 9. Example of a concrete model [46]

Fig. 9 shows an example of a concrete model corresponding to the previously introduced abstract model. The goal is still to assess the "Availability" of a "System", in this case the "Customer Support"-function of a "CRM System". During the evidence collection process, information about the existence of concrete entities and relationships are gathered. A "System administrator" called "Juliet" who uses a "Fault Management"-function of a "Maintenance system" to get notifications about possible system errors, administrates the "CRM System". Another "System administrator" called "Joseph" administrates the "Maintenance system". "Joseph" does not use any other systems for his work. Therefore, the "Availability" of the "Maintenance system" is only dependent on the "Reliability" of the system and the "Responsiveness" and "Experience" of Joseph. Evidence collection on the values of attributes identified that both attributes of Joseph are stated as "High" either by observation or by interviewing

him. By observing the error log of the “Maintenance system”, its “Reliability” is stated as “Medium”, what is consistent with the interview of Juliet, who said that the availability of the fault management function is “Medium”. Juliet also said her “Experience” is “High” and it was observed that her “Responsiveness” is also “High”.

With this information given and the addition of further information about the reliability of the observed evidence it is possible to create a Bayesian network and with this to calculate a possible value for the availability of the customer support in this example.

The resulting concrete model is translated into a Bayesian network. The evidences are used as parent nodes, as well as the priors of attributes, which are not influenced by other attributes. Then auxiliary nodes are inserted to represent the aggregation functions of the attribute relationships. Next all created nodes are connected according to the specified relationships, which were defined in the abstract model. In the last step, inference is made. After this calculation, the information of the values is stored in the properties of the concrete model.

Further information about the functionality of a Bayesian network can be found in chapter 6.3. The creation of a Bayesian network from a concrete model is described in detail in chapter 8.3.4.

6.7 Further Considerations

The approach for decision-support taken in the enterprise architecture analysis method examined before is determined by mathematical modeling with Bayesian networks. It is obvious that the dependency on a large and credible amount of evidences necessary for instantiating a good concrete model, can result in extensive efforts gathering them. That is why section 6.7.1 provides a heuristic method of resolution for this dilemma. Section 6.7.2 then shows that the probabilistic based method is only one alternative for supporting architectural decisions. The section describes the approach of Zimmermann et al., which is determined by a Meta model-based decision model.

6.7.1 Data Collection Prioritization

In general, there is one problem in evidence collection. The amount of evidences to be gathered rises rapidly with the complexity of the concrete model. In a highly meshed and cross-linked Bayesian network, the addition of an extra element increases the number of attributes surpassingly. Considering multiple sources of evidences lets the number of pieces to be collected rise dramatically. There can easily be more than ten thousand pieces of evidence. It is obvious that trying to gather all pieces of evidence will result in an unwarranted expenditure of time. Consequently, it is advisable not to try to collect each and every piece of evidence, but only those which have the biggest impact on the assessment result while having the lowest collection cost. An algorithm to determine which piece of evidence to collect next is examined in [67].
Based on the work of Jagt [34] Närman et al turn to theory of diagnosis in Bayesian networks. [95] states that diagnosis is best known as the process of identifying the

disease or disorder of a patient or a machine by considering its history of symptoms and other signs. Diagnosis itself is done by conducting two tasks: (i) determining the cause(s) of the observed effects, and (ii) increase the certainty of the true cause(s) by gathering information, e.g. from tests or historic data. The collected information has to comply with three assumptions:

- The information is perfect, i.e. not wrong or incomplete
- The information is non-intervening, i.e. does not change the world
- The information decreases uncertainty

As information usually cannot be collected without cost, there needs to be a strategy to determining which information is worth being collected (next). As stated above the most valuable information has the biggest decreasing impact on uncertainty while having the lowest collection cost.

In order to conduct diagnosis on Bayesian networks Jagt associates them with a structure that distinguishes the variables necessary for diagnosis. In these so-called diagnostic probability networks (DPN), task (i) can be executed by applying Bayesian network reasoning algorithms, and task (ii) can be done by using the *value of information* concept described later.

A DPN is defined as a Bayesian network where at least one random variable H is a hypothesis variable and at least one other random variable T is a test variable (here variables of the model that one potentially can collect information about).

- Let H be a hypothesis variable and *H* a set of all hypothesis variables also referred to as hypothesis set. A state of a hypothesis variable h ∈ H is denoted as a hypothesis state. Each hypothesis state may represent a possible disease, fault in a system, or any other discomfort.
- Let T be a test variable and *T* a set of all test variables, also referred to as test set. A state of a test variable t ∈ T is denoted as a test state. Each test state may represent an observation, physical sign, indicant, symptom or laboratory result. Each test is associated with a cost value Cost (T), if a test variable has no cost, the cost value is set to 0.

Fig. 10 shows the application of the diagnosis structure on a Bayesian network, which represents an extended version of the example from *Fig. 8*. For didactic purposes, it introduces an additional attribute to the *System* entity: *Security* and another attribute to the *Function* entity: *Reliability*. Moreover, some influence relationships have been added. The structure of this example Bayesian network is based on the fictitious Asia example from [93]. This network illustrates that the reliability of the customer support function (*CSR*) is dependent on the availability (*CA*), security (*CS*), or reliability (*CR*) of the CRM system. Thereby the responsiveness (*JR*) of the administrator Juliet increases the probability of the CRM system's availability, while her experience (*JE*) has influence on the probability of both security and reliability. Neither the fact that the customer support function is available nor the fact that it is reliable discriminates between the CRM system's availability and reliability. Each of the variables is associated with a probability distribution. In the colored DPN from *Fig. 10* the red colored "caused effects" are part of a hypothesis set H = {*CA*, *CS*, *CR*}, whereas the green colored nodes are part of the test set T = {*JR*, *JE*, *CSA*, *CSR*}. Assuming that each hypothesis variable has only two states, this would result into $2^3 = 8$ scenarios.

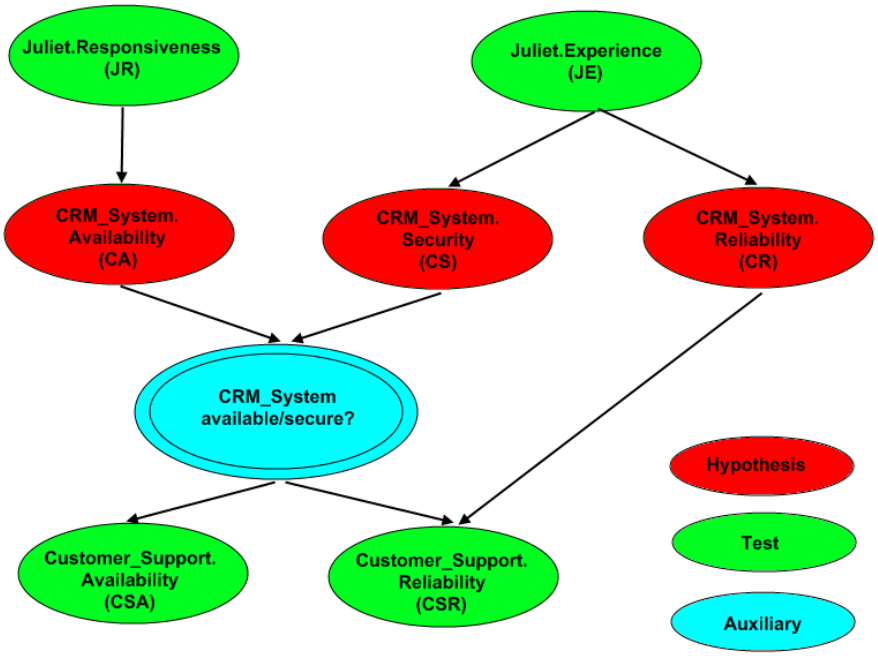


Fig. 10. The DPN of the example (based on fictitious Asia example in [34])

It is obvious that the number of scenarios grows exponentially with the number of hypothesis variables. The amount of the number of scenarios results from $number\_of\_states^{number\_of\_hypothesis\_variables}$. The exponential growth of the scenarios, which can make systems hard to compute, can be antagonized by applying so-called "idiot's Bayes". This is a naive Bayes structure where only one hypothesis value is allowed and a conditional independency between each test variable is assumed. This results in a single cause assessment.

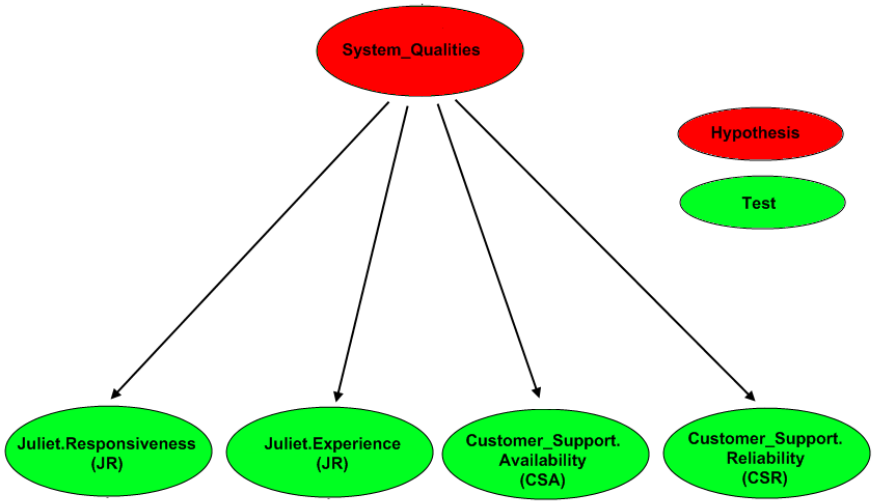


Fig. 11. Example transformed into naive Bayes structure

In *Fig. 11* the example from *Fig. 10* is transformed to a naive Bayes structure. With this naive Bayes structure, there iss only one overall system quality state left. The simplicity of such naïve Bayes structures make it easy to assess and to compute. However, obviously there is a mismatch between the Bayesian network in *Fig. 11* and the "real world" in *Fig. 10*. Thus large enterprise architecture models are likely not to be transformed into naïve Bayesian networks as they should provide the most relevant and exact picture of a company's enterprise architecture (moreover it is hardly achievable to do so due to the complexity). That is why it is necessary to have information collection prioritization within multi cause networks. Since the *value of information* is based on the naïve Bayes structure and a foundation for understanding the *marginal probability approach*, this topic is examined in the following paragraphs briefly.

As explained earlier the uncertainty of causes can be reduced by gathering additional information. Since usually there is a restriction of resources, e.g. time or personnel limitations, it has to be assessed which information is worth collecting. This can be done by applying the value of information concept. This concept computes, which test to perform, i.e. which information to acquire. The formalization of this procedure is in a similar notation as [35]

Consider a DPN with a set of hypothesis variables H, a set of test variables T, and a value function $V(\Pr(H)) : [0; 1] \rightarrow \mathrm{R}$. The *expected value* (*EV*) of performing a test $\mathrm{T} \in T$ is used:

$$EV(T) = \sum_{t \in T} V(\Pr(\mathcal{H}|t)) \Pr(t).$$

Whether there will be a benefit when performing a test can be computed by calculating the *expected benefit* (*EB*) of performing a test, which is defined as the difference

between the expected value of performing a test and the value without performing a test:

$$EB(T) = EV(T) - V(\Pr(\mathcal{H})) =$$

$$\sum_{t \in T} V(\Pr(\mathcal{H}|t)) \Pr(t) - V(\Pr(\mathcal{H})).$$

For assigning a ranking to each test based on the benefit of the test and the cost of the test T, the *test strength* (*TS*) is used:

$$TS(\mathcal{H}, T) = \frac{EB(T)}{V(\Pr(\mathcal{H}))} - K\ Cost(T).$$

The coefficient *K* determines how much the cost of a test weighs in combination with the benefit of a test. There is no standard formulation for the value of *K*, so it can be set by the user.

Lastly, a value function is necessary to compute a prioritization ranking of all tests that can be performed. Therefore linear functions are useless as they return always an expected benefit of zero [35]. Jensen also discovered that convex value functions always return a positive expected benefit, which complies with the assumption that information never increases the uncertainty. According to [27] functions which have the ability to determine most informative tests, are called quasi-utility based functions. These functions reward tests that reduce uncertainty with high value and vice versa. One of the most popular quasi-utility functions is the entropy function [90].

With these concepts, a prioritization of information collection can be achieved. A small example of how to conduct single cause diagnosis with the help of GeNIe (7.1.1.2) can be found in [34].

Jagt states "… the major disadvantage of this application is the restriction of pursuing only one state instead of multiple states. If all the hypothesis states were mutually exclusive as in the naive Bayes structure, this application would be logical and even useful. However, if multiple causes are possible this application totally ignores the other causes and only focuses on proving the presence or absence of the selected cause." To support the data collection prioritization for multiple hypothesis networks (and hence solve the problems of presentational and computational complexity) he introduced an approximation approach – the marginal probability approach[2] (MPA). Jagt gives a short summary of this approach:

"It uses the relation between the marginal and joint probability distribution to justify the use of marginal probabilities. This approach saves a lot of computational effort, since the joint probability distribution is no longer calculated. Furthermore, the approach allows the presentation of the hypothesis states instead of the enormous number of hypothesis scenarios."

[2] In fact he introduced two approaches: the marginal probability approach (MPA) and the "less radical" joint probability approach (JPA), but the focus lies on the function based on the MPA

Therefore, he formalizes the relationship between marginal probability and joined probability as a discrete derivation of the Frèchet-Hoeffding bounds [35]. This relationship allowed him to formalize a corollary that states that interesting probabilities are only possible if the marginal probabilities are close to zero and one (and thus certain). This characteristic is called marginal strength. With the help of this quality, there is the ability to reduce the uncertainty between scenarios to the goal of decreasing uncertainty of marginal probabilities. Jagt states that the benefits of this approach are noticeable in the number of computations, since it is no longer necessary to determine the computationally expensive joint probability distribution. Moreover, the user is able to interpret the marginal probability states directly instead of needing to look at the numerous scenarios and their probabilities.

An appropriate value function that determines which test to be selected should assign high values to marginal probabilities close to zero or one (certain) and should have its minimum at 1 - (1/n) (most uncertain), where n is the number of targets in a set of hypothesis states. To assure that each information decreases uncertainty the function has to be convex again. As the final value function, Jagt defines the Marginal Strength1:

$$MS1\left(\Pr\left(\mathcal{F}\right)\right) \equiv \left(\frac{\sum_{f\in\mathcal{F}}\left(f-0.5\right)^2}{\left(\frac{1}{2}\right)^2} - n_{\mathcal{F}}\right) * \frac{1}{n_{\mathcal{F}}}.$$

Where $\mathcal{F}$ is a set of target states where each target state *f* represents a hypothesis state, which the user wishes to pursue, and $n_{\mathcal{F}}$ is the total number of target states.
Fig. 12 shows that the function is convex and has its maxima at (0, 0), (1, 0), (0, 1), and (1, 1) and its minimum at (0.5, 0.5) in the case of two random targets.
The final value function that is necessary for our purposes is the sum of the marginal strength for all target states:

$$V(P(F)) = \sum_{f\in F} MS1(P(f))$$

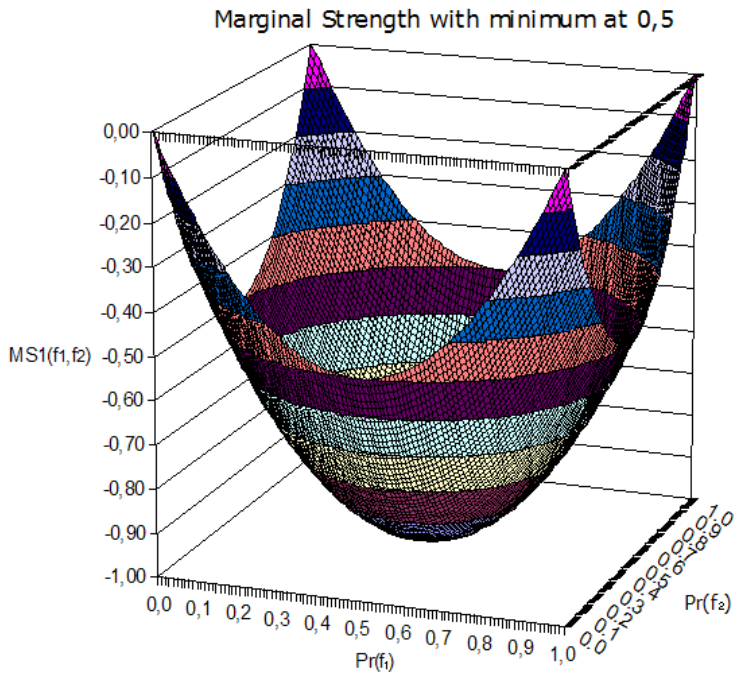


Fig. 12. The MS1 function over two random targets f1 and f2 taken from [34]

Närman et al. have applied the value function in GeNIe's (see 7.1.1.2) multiple cause module (MCM) which itself is implemented by Jagt as a modification of the single cause module (SCM). GeNIe itself is a graphical development environment for building graphical probabilistic and decision-theoretic models. It provides a user interface to the underlying SMILE library (see 7.1.1.1).
Rather than staying with the fictitious Asia example it is now referred to a more complex (multi cause) example from [46] within the domain of enterprise architecture analysis. In short, the probable outcome of the attribute Interoperability in the entity Service Cluster is the target. All possible causes can be resolved in *Fig. 13*. *Fig. 14* shows the calculated ranking of the tests inside GeNIe's MCM UI. Once a test has been selected and performed, the remaining list will be recalculated. For more information about abstract models, it can be referred to section 6.4.

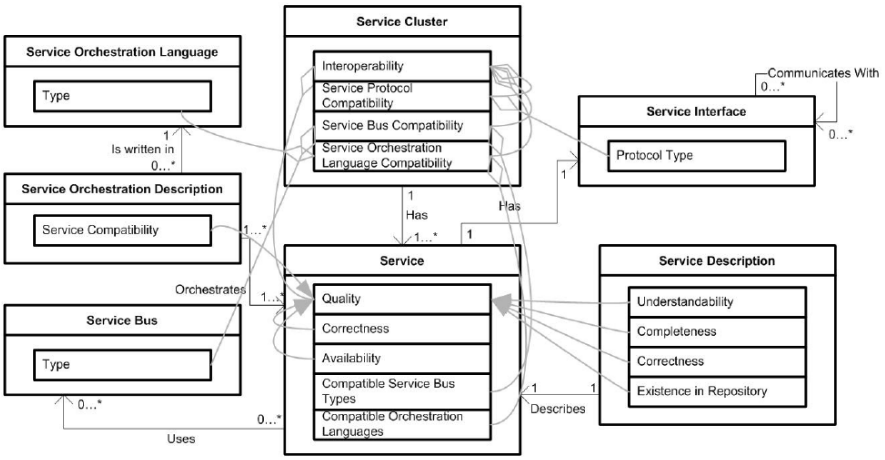


Fig. 13. Example: abstract model for Service Cluster Interoperability from [67]

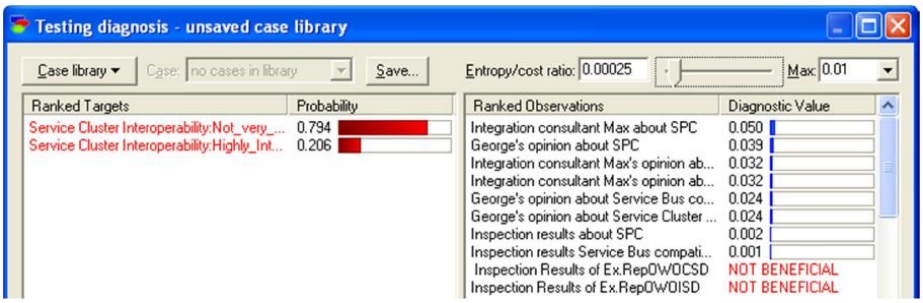


Fig. 14. Test ranking result in GeNIe's MCM user interface taken from [67]

The example shows that that the use of the diagnosis algorithm reduces the cost of evidence collection by providing a strategy of which data is to be collected first. In comparison to not having a calculation-based particular strategy this approach can save temporal and personnel efforts and thus reduce the cost of enterprise architecture analysis. In future versions of EAT it will be worth integrating SMILE's MCM capabilities and providing an integrated user interface for data collection prioritization.

6.7.2 Alternative

Alternatives in the domain of decision support have been developed since the 1990s, e.g. Questions, Options and Criteria (QOC) [59] or Decision Support Systems (DSS) [61, 98] as well as this work, but they mainly focus on decision-making. This chapter presents a work that additionally takes decision identification and decision enforcement into account, too.

This approach for architectural decision support has been developed by Zimmermann et al. [114]. In their work, they propose a framework that facilitates proactive decision support with the three conceptual steps - decision identification, decision-making, and enforcement. They aim for the easement of reusing architectural decision rationale considering collaboration and automation aspects. With this framework, they address the limitations of existing decision capturing methods, which are often "… regarded as a retrospective and therefore unwelcome documentation task…" that is not available to decision makers when they actually need it – before making decisions.

The core idea of their framework is to instantiate Meta model-based decision models with the help of requirement models and reusable decision templates.

In the first step of the conceptual framework, decision identification, requirements and earlier decisions help to identify individual decisions. In the second step, decision-making, the selection of alternatives according to specific decision drivers is made. The last step, decision enforcement, then focuses on publishing the decision, achieving acceptance, and reducing enforcement efforts.

In order to connect the steps of the framework and to enforce the identified design goals *team collaboration* and *previously-made-decision-harvesting* they developed a decision modeling framework that also supports basic decision-making requirements (use cases), which have been identified by interviewing 100 practicing software architects [114]:

1. Obtain architectural knowledge from third parties
2. Adopt and filter obtained decision knowledge according to project specific needs
3. Delegate decision making authorities
4. Involve network of peers in search of additional architectural expertise
5. Enforce decision outcome via pattern-based generation of work products
6. Inject decisions into design models, code, and deployment artifacts
7. Share gained architectural knowledge with third parties

The Meta model is illustrated in *Fig. 15*:

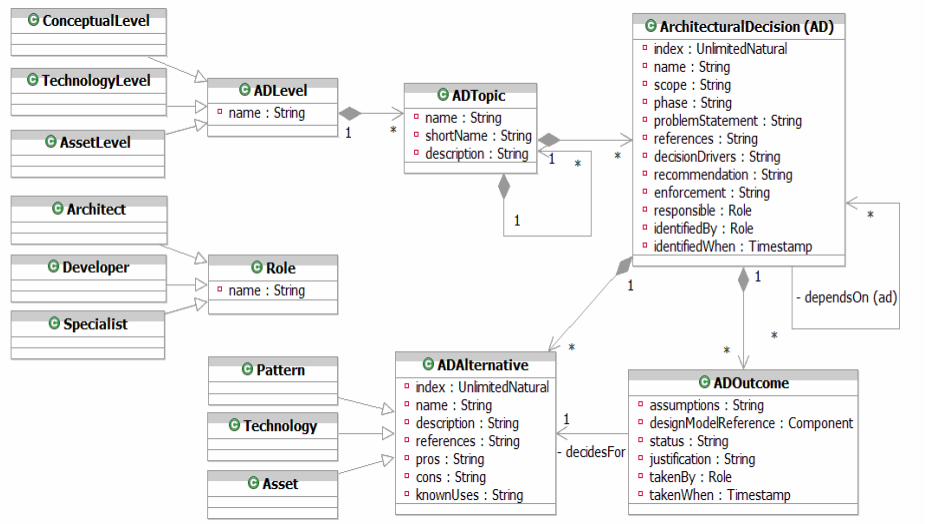


Fig. 15. Meta model for architectural decision reuse taken from [114]

This Meta model provides a description for architectural decisions and their background(s). The entities Architectural Decision (AD) and ADAlternative provide background information, whereas ADOutcome, as a third core entity, represents the separated outcome. The ADTopic Entity groups closely relates to AD (hierarchically). These topics are assigned to one of the three ADLevel (abstraction level) which should separate related but unmergeable topics from each other in order not to underestimate either strategic or generic decisions [114].

With the help of this Meta model *Fig. 16* shows the modus operandi for step 1, decision identification. There the initial decision model should be instantiated from specific requirements and decision templates (that serves as a completeness checklist). Therefore, reference architectures can provide common vocabulary, certain domain patterns, and design model elements.

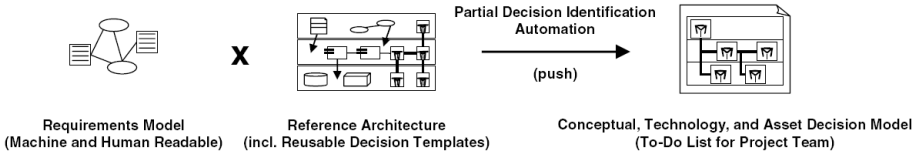


Fig. 16. Semi-automatic decision identification in requirements model and reference architecture taken from [114]

The second step, illustrated in *Fig. 17*, then integrates decision support systems into the framework that use the decision models from step 1. Moreover, a list of decision drivers per decision provides additional information (functional, non-functional, technical, non-technical) for decision-making.

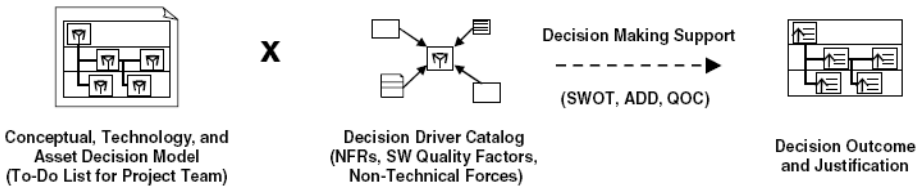


Fig. 17. Decision models, decision drivers and techniques for decision making taken from [114]

The third step, illustrated in *Fig. 18* enforces machine-readable decisions by automatic code generation. This reduces "…unnecessary development efforts and ensure[s] architectural consistency." [114]

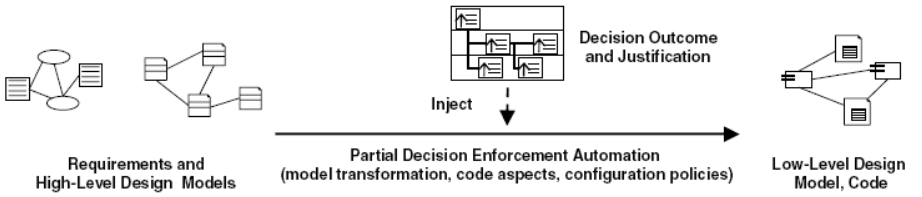


Fig. 18. Decision enforcement via injection into model transformations and code generation taken from [114]

An application of the conceptual framework to the SOA design space can be found in chapter 4 of the Zimmermann paper [114]. ADkwik, a Web 2.0 collaboration front end implementing the concepts presented in that paper can be found in [89].

Concluding, this proactive decision-making approach has as its goals to improve decision reusability and rationale sharing. The approach "…is generally applicable if several applications are built in the same or a similar context and if full decision automation is an illusion." [114]. It is based on activities and results from software architecture research, design decision rationale research, and pattern research. For this approach, a well-structured requirements model and one reference architecture at least are prerequisites. In comparison to the EAT approach "…decision making support in [the decision model] approach empowers the architects to make informed decisions based on collective insight …" [114] instead of being based on quantitative assessment of enterprise information systems. Generally this approach distinguishes from the EAT approach as it utilizes past decisions and needs explicit knowledge. Lastly, the authors admit that their approach is not to fully automate decision-making. In their opinion, "…the importance of tradeoffs in specific contexts and design drivers naturally makes full automation impossible; heuristic solutions are required." [114]

7 Concept of a Prototype

In this chapter, a concept for the implementation of the Enterprise Architecture Tool is given. Therefore, the used technologies and software are presented first. Moreover, the underlying data model and the architectural design of the tool are explained.

7.1 Chosen Platform

In this section, chosen technologies and the reason why they have been selected are presented. In the process underlying technologies, implementation libraries, and development environments are considered. The chosen platform defines the relevant environment for the EAT implementation.

7.1.1 SMILE and GeNIe

In order to calculate the Bayesian networks described in 6.3 the SMILE library and the GeNIe platform, provided by the Decision Systems Laboratory of the University of Pittsburgh [22] have been used[3]. This software package, implemented for academic purposes, consists of two independent parts, which are explained, in the following.

7.1.1.1 Structural Modeling, Inference, and Learning Engine (SMILE)

The platform independent library SMILE (Structural Modeling, Inference, and Learning Engine) provides functionality for implementing graphical probabilistic and decision-theoretic models. It supports different types of models, namely influence diagrams, structural equation models and Bayesian networks. The SMILE API provides load, save, edit, and create functionality as much as operations for making inferences in such models.

The library is implemented in C++, but several wrappers are available on the SMILE website. In the program described, the Java wrapper jSMILE was used.

SMILE itself does not provide any visualization functionality. Its API can be used by other software. Thus, a connection from the user interface of the application to SMILE can be implemented (see more in 8.3.4).

[3] There are several other tools which support decision making based on probabilistic inference, an overview can be found at [111]. The SMILE and GeNIe platform has been successfully used in the previous implementation. Positive experiences were also made by other research groups, an example can be found at [30].

7.1.1.2 Graphical Network Interface GeNIe

An example of an implementation of a user interface for SMILE is GeNIe. It is a graphical editor to consume the functionality provided by the SMILE API, in order to create decision theoretic models by using the graphical click-and-drop interface. "Models developed using GeNIe can be embedded into any applications and run on any computing platform, using SMILE, which is fully portable" [12]. As all models that were created with SMILE can be processed by GeNIe, the application can be utilized as a visualizer for SMILE-models.

The tool helped to implement and follow the calculation process described in 8.3.4, a control and comparison of the passed steps was eased. After each stage of the calculation process, a temporary SMILE network was build. This net was visualized with GeNIe and compared to the target network that was built manually.

Discrepancies were found and corrected immediately. This ensured that network creation was implemented correctly and the resulting networks were identical to networks, which were created by hand.

This approach turned out to be a wise proceeding, because the calculation process was extended and got more complicated as models grew.

7.1.2 XSD (XML Schema)

XML Schema Definition (XSD), in the following often referred to as "XML Schema", is one of several XML schema languages that have been developed in the last years [54]. This language was published as a recommendation by the W3C. Its target purpose is verbalized at the W3C homepage:

"The purpose of the XML schema language is to provide an inventory of XML markup constructs with which to write schemas." [110]

In an XSD document, a class of XML documents is described and defined. An XSD file can be considered as the construction plan of multiple XML documents. Their components, especially relations, way of use, and content are restrained and documented. All components of an XML structure that will ever occur must be considered and described in the underlying schema. The way these components were nested (e.g. if a component consists of just one subcomponent or if a sequence with zero to infinity elements of the same subcomponent is allowed) is regulated in an XSD document. Elements and their meaning, data types, attributes and their values, as well as content and notations of entities were limited [110]. Besides the typical data types such as String, Integer and Boolean, user-defined types are possible as well. These user-defined data types have to be introduced and described, in order to be usable. Likewise, default values for properties can be set, so that elements are initialized on start-up.

The requirements, which a certain XML Schema must fulfill, are documented in the XML Scheme Requirements [110]. They were split into three categories; their gists are listed on the following page:

- **Structural Requirements:**
  - Mechanisms for constraining document structure and content
  - Mechanisms to enable inheritance
  - Mechanism for embedded documentation

- **Data type Requirements:**
  - Define a type system that is adequate for import/export from database systems
  - Distinguish requirements relating to lexical data representation vs. those governing an underlying information set
  - Allow creation of user-defined data types, such as data types that are derived from existing data types and which may constrain certain of its properties

- **Conformance:**
  - Describe the responsibilities of conforming processors
  - Define the relationship between schemas and XML documents
  - Define the relationship between schema validity and XML validity;
  - Define the relationship between schemas and XML DTDs, and their information sets
  - Define the relationship among schemas, namespaces, and validity
  - Define a useful XML schema for XML schemas

By fulfilling these requirements, a powerful language was created to catalog, describe and define XML structures and their underlying vocabularies. With the use of these features a large benefit was achieved in EAT (see 7.3.1), as type checking was eased. XML documents were specified very detailed and so the data structure can be understood quickly by examining the underlying XSD file. The XML files contain the input and output of the tool, since they were used for loading and saving. This means that an understanding of the XSD document is a first and important step on the way of comprehending the complete data model as the XSD file manifests start and end of the data flow.

XML Schema documents were likewise represented in valid XML documents and were therefore recursively defined in an XSD document. In comparison to Document Type Definition (DTD)[4], always two files were created: A specifying XSD document and an instantiating XML. Additionally, XSD is known to be namespace aware, which is a missing feature of DTD documents [66].

An analysis of the most used XML schema languages discovered that XML Schema Definition has a very strong expressive power. In [54] three levels of "expression

[4] DTD is another wide spread XML Schema language, which is under W3C recommendation status, too.

"DTD is a description in XML declaration syntax of a particular type or class of document. It sets out what names are to be used for the different types of elements, where they may occur, and how they all fit together." [103]

Its features were defined to be similar to the abilities of XSD documents; on the other hand, no type checking was implemented. So a successor of DTD can be seen in XSD documents.

power" were defined with the result that XML Schema Definition was assigned to the most powerful class. "XML Schema, Schematron and DSD have the strongest expressive power. [...] XML Schema supports features for schema data type and structure fully ..."

XSD was chosen as the description language for the Meta models in EAT (7.3).

7.1.3 Castor

In this chapter a brief description of the Castor library and the way it is utilized in this project is given. Castor is used as a tool to create a Java class structure from an existing XSD description of a data structure. The created Java classes contain all methods for checking their contents for data-structure compliance and provide the needed functionality for writing and reading the according XML files.

7.1.3.1 Features

Castor is

"[...] an Open Source data binding framework for Java[tm]. It's the shortest path between Java objects, XML documents and relational tables. Castor provides Java-to-XML binding, Java-to-SQL persistence, and more." [8]

In this project, only the binding between Java and XML was used. There are two ways in which Castor supports binding between Java objects and XML files. One way is the possibility to map an existing Java class structure to an existing, or for this purpose, created XML structure.

The other way is the creation of a Java class structure from an existing XSD file. XSD is a description of the general structure of concrete XML Files (cf. 7.1.2). Castor is able to create Java classes by using these definitions. The classes implement the typing that is defined in the XSD file, for example the definition of a Double-Type or a String-Type. Extensions in the XSD File are realized in the Java structure by the use of inheritance. Castor is able to utilize Java-Collections for managing lists of elements.

7.1.3.2 Castor in EAT

In the EAT project, the way from an existing XSD document to a Java class structure was chosen. This is because of the existence of XSD files from previous versions, which describe the model structure (see chapter 7.3). It is also possible to build models with other XML editors and validate them against the XSD.

During development, Castor was not used as a library that provided methods used in the code. It was used as an external tool inside the development workflow instead.

The workflow used to generate Java classes by utilizing Castor is described later in this chapter.

The auto generated Java classes implemented all needed features for using them as data basis. There are checking methods implemented, that examine whether objects are valid and whether it is possible to marshal them into an XML document. The marshalling ability makes it possible to save a model into an XML document by using only one method-call. In addition to this marshalling feature, unmarshalling was also implemented by the Castor code generation.

7.1.3.3 Usage Workflow

Since the data structure in EAT is generated externally there are precautions to be made when altering the underlying structure by changing the XSD.

In this project, Castor was used in Version 1.2. To generate the Java classes from the XSD file it is necessary to run the class `SourceGeneratorMain` from the package `org.exolab.castor.builder`. The following command-line parameters were used:

- `-i <pathToXSDFile>` specifies the complete absolute path to the used XSD file.
- `-package <targetPackage>` specifies the desired packager which contains the generated Java classes
- `-types j2` tells Castor to use Java collections
- `-dest <outDirectory>` specifies the absolute path to the directory where the classes are generated
- `-f` dismisses all minor warnings and errors of the code generation.

Warnings should be disabled because in default mode a warning is given and the generation is aborted when the generated files already exist. This is improvable because while developing iteratively (as described later) most files exist already exist.

It turned out that the impacts on the code of the EA Tool were minimized when changes were made using the described workflow. It was experienced that it is beneficial to keep older parts in the model. Certainly, the structure becomes tainted but implementing the feature throughout the whole EA Tool is not necessary anymore.

As the EA Tool was implemented, it was realized that changing the model structure and putting the changes through the whole code base works fine as follows:

1. Changing the model in a way that new features are added as optional elements and keeping everything else.
2. Running Castor[5] and copying the new Java class files into the code base.
3. Implementing the new features in the tool. Doing this it is necessary to notice that the new parts are optional. Thus, there is no automated checking.

[5] Castor was parameterized with the command line options specified earlier in this chapter.

4. Changing the XSD again and make new must-features obligatory.
5. Running Castor and copying the new Java class files into the code base.
6. Check the whole program for errors by going through the complete workflow and focusing especially on the new features.
7. Altering the model to remove old features that are no longer used.
8. Running Castor and copying the new Java Class files into the code base.
9. Cleaning the code from the usage of all old features.

7.1.4 Evaluation of Graph Libraries and Frameworks

This chapter first explaines why it was necessary that a graph library/framework[6] evaluation was conducted. Then it describes how the evaluation was realized by stating the candidates, explaining the criteria, and finally presenting the result of the assessment. Based on the evaluation it was decided to use NetBeans Visual Library 2.0 pre3 for implementing graph management and interface actions.

7.1.4.1 Motivation

The implementation of a modeling tool makes high demands on the underlying graph library, respectively framework. This is because most user actions take place on a drawing surface that cannot be managed by default interface structures like tables, trees, lists, and frames. The user is able to draw things somewhere on the surface that can get easily confusing, unwieldy, and thus unusable if not managed properly. More specifically, the Enterprise Architecture Tool requires features that are not usual in other modeling or meta-modeling tools and turned out to go beyond the capabilities of AWT and Swing. For instance, there has to be a possibility of drawing one or more connections between an Attribute relationship Widget and its related Entity Relationship Widget(s) automatically. Moreover the underlying model of EAT requires a management that handles data model and visual representation separate but synchronous.

As mentioned previously the version of EAT described in this work is a reimplementation of an older version. In that older version, NetBeans Visual Library was used, too. Thus, a verification of this selection was intended, too. Some problems, especially with the low performing orthogonal router algorithm, emphasized the need for a library/framework assessment before reimplementing the tool.

That is why it had to be examined what graph library or framework implementations exist and whether they fulfill our requirements at all (especially with respect to chapter 7.4). Pay attention to the fact that the aim of the evaluation was to find the

6 A framework defines most of the control flow of developed software. Via inheritance and overriding new functionality and look and feel can be added. A library “only” provides routines and methods that can be used for the implementation of software. Summarized: a framework calls the developed code; a library is called by the developed code. [52]

most feasible solution within the context of the EAT project. It was not to declare any solution to be the best graph library/framework implementation at all.

7.1.4.2 The Candidates

In general, the proposition of graph libraries is complex and extensive. Besides widely known technologies like Java AWT [96] or Swing [97], also SWT [15], SwingWT [101], Zaval GUI Designer Package [113], and Buoy [5] are worth mentioning here. Finally, four candidates out of the large range of implementations namely NetBeans Visual Library [75], QtJambi [85], JHotDraw [36], and GEF (Graphical Editing Framework) [20], were chosen. This section explains why they have been chosen and gives a small introduction of each candidate.

The selection was determined by the decision that the interface of this tool should be Java-based. This is because Java was used in the EAT project before and thus is familiar to those who are going to extend the tool. In addition, Java is in conjunction with the other technologies that have been used during the implementation, e.g. Castor. Choosing Java as the preferred programming language excluded XML User Interface Language (XUL)-based alternatives like SwixML from the choice. The four candidates mentioned above fulfilled the requirements of being Java-based, feature-rich, prevalent, and free of charge.

NetBeans Visual Library
As the name says, NetBeans Visual Library is a library implementation. It is a further development of the original Graph Library and supports graph-oriented modeling [75]. The library is part of the NetBeans 6.0 platform, which provides a graphical designer for modeling user interfaces. NetBeans Visual Library serves as the reference candidate because it was used in the preceding EAT project. Detailed information about NetBeans Visual Library can be found in chapter 7.2.

QtJambi
QtJambi is a wrapper of the widespread Qt library and enables implementing platform independent user interfaces in Java [85]. QtJambi was developed by Trolltech and was released in June 2007. A specific feature of this library is the management of object interactions via the “Signal-Slot-Concept” which enhances maintenance and extension of software implemented with QtJambi. Moreover, it provides a graphical UI designer named Qt-Designer.

JHotDraw
The Open-Source project JHotDraw is a framework for the development of modeling tools [36]. Hence, it provides graphical objects, toolbars, and views for implementing appropriate editors. It is often used for academic purpose, especially for teaching the advantages and the use of design patterns. Erich Gamma, one of the developers, is also one of the authors of the famous “Gang of Four” book “Design Patterns, Elements of Reusable Object Oriented Software”.

Graphical Editing Framework
GEF is another candidate of the class of IDE bundled libraries like NetBeans [20]. Because it is the graphical underlying of Eclipse's graphical modeling capabilities, it is very prevalent. GEF is used "…to build a variety of applications, including state diagrams, activity diagrams, class diagrams, GUI builders for AWT, Swing and SWT, and process flow editors" [21].

7.1.4.3 The Criteria

This section introduces the criteria the four candidates have been evaluated against.

The criteria used to assess the candidates resulted from two requirements. On the one hand, there was the scope of the project and difficulties that occurred in the previous EAT implementation, which required specific features to be fulfilled. On the other hand, there are fundamental requirements on libraries/frameworks that had to be met.

From the previous EAT project, three important categories were identified that contain at least one criterion: "Drawing and Layout", "Graphical Appearance", and "Documentation (Learnability)". The category "General" includes criteria that are fundamental requirements for the project. The following *Table 4* lists and explains the criteria of each category:

Table 4. Assessed criteria

Category	Criteria	Explanation
Drawing and Layout	Routing Algorithm	Provides automatic routing (orthogonal, curve, free)?
	Built-in Functions	Features, e.g. zoom, grid layout, satellite view, etc.
Graphical Appearance	Customizability	Capabilities for changing provided figures, shapes, etc.
Documentation (Learnability)	API Documentation	Provides well-arranged, complete API?
	Samples	Are there executable samples or tutorial files? Could be videos, too.
General	Dependencies	Ease of install and diffusion of the final application.
	Diffusion Rate	Popularity. Degree of spreading.
	Maturity (Stability)	Version, known bugs, release cycle, community activity.
	License	GPL, EPL, free at all? Developed code under GPL, EPL, or commercial license?

Initially there was an additional criterion “Feasibility of Managing Widgets” in the “Graphical Appearance” section, but this was not evaluated because it depends more on the manner of implementation than on built-in capabilities provided by the libraries/frameworks.

7.1.4.4 Evaluation and Result

Each candidate was evaluated against each criterion. For grading the candidates, a numerical rating scale was used. The scale goes from “1” meaning the criterion was fulfilled “very bad” up to “5” meaning it was fulfilled “very good”. The abbreviation “n/a” stands for “not applicable” and means that there was no information found. For calculation purposes, the numerical representation was set to “1.5” because we assumed that it is very likely that the criterion is fulfilled “bad” or “very bad” when there is no public information found.

The evaluation itself was conducted in a way that all candidates were stepwise assessed against each criterion. *Table 5* shows the allocation of points for each candidate and criterion:

Table 5. Evaluation Result

Criteria	NetBeans Visual Library	Qt Jambi	JHotDraw	GEF
Routing Algorithm	3	1.5	2	4
Built-in Functions	5	3	4	4
Customizability	5	4	3	5
API Documentation	5	3	3	3
Samples	5	4	2	3
Dependencies	4	2	4	3
Diffusion Rate	5	2	3	5
Maturity (Stability)	4	3	1	4
License	5	3	5	4
Result	**41**	**25.5**	**27**	**35**

NetBeans Visual Library
Table 5 shows that NetBeans provided the best overall performance. It convinced with its superior functionality, clean and complete documentation, and its popularity. It has several partners like eBay or Ricoh [71] and is used in various projects, for instance by Systinet, Software AG, and Flashline [72]. Furthermore, there is a large knowledge base on the web that is well kept and extended by a diligent community. The final application can be distributed within a simple .jar-File. SUN as the vendor of NetBeans is very experienced and the development team of this library is very active. The main problem of NetBeans Visual Library, the routing algorithm, is already a known issue and it is foreseeable that a patch will fix this soon [74].

QtJambi
The Qt implementation provided the second best documentation, which makes it easy to learn. Nevertheless, Qt Jambi neither convinced with routing algorithm capabilities nor maturity qualities. Although Qt is widespread (e.g. used for KDE, Opera, Skype, Mathematica, GoogleEarth or VirtualBox) and proved to be mature, Qt Jambi has no real reference until now. In addition, the native libraries for each operating system make it inflexible.

JHotDraw
This framework convinced with many graphical built-in capabilities. Because of its framework characteristic, a lot of functions and activities that usually occur on a user interface were already implemented. It focuses on graphical visualization and provides a good API. Unfortunately, it delivers only rudimental routing and only offers `interfaces`. Another negative is the fact that JHotDraw is a private project and thus release cycle and bug fixes will take long. Furthermore, the current version 7 is marked as unstable [37].

Graphical Editing Framework
GEF as the runner-up provides several routing algorithms. As the only library/framework in the field, it supports Bezier-routing, too. As a framework, it also offers easy access to complex, graphical routines, because they are already implemented. As a part of Eclipse, it is prevalent and licensed under EPL. Third party applications that were built using GEF can be made available under one's own license. However, because of the poor and chaotic documentation as well as dependencies on DLL-Files its usability and flexibility is unsatisfactory.

Overall
Finally NetBeans Visual Library was picked for the implementation of EAT. Despite the fact that the implementation would be possible with any of these libraries/frameworks, the effort of achieving this would be influenced by the quality and quantity of built-in functions and documentation. These are strengths of NetBeans Visual Library. The disadvantage of a library in comparison to a framework, which is that complex graph and typical control flow routines are not implemented, is compensated with the use of NetBeans IDE (see 7.1.5). This IDE provides a user interface designer for building and managing windows and dialogs. Furthermore the overall performance of GEF, Qt Jambi and JHotDraw did not exclude the possibility of facing currently unknown issues during the EAT implementation. That is why NetBeans Visual Library was chosen for the implementation of the Enterprise Architecture Tool.

For further information about the results of the evaluation and explanations of the grading, conduct Appendix C: Evaluation Table.

7.1.5 Chosen IDE – NetBeans

This section explains why the development of EAT required an integrated development environment (IDE) and why NetBeans IDE was chosen. Moreover, it outlines the history of NetBeans IDE and shows what features it provides referring to the debug mode user interface of the IDE.

7.1.5.1 IDE

Formerly, software developers used several tools from different vendors for their development project. For example, there was a text-editor for main coding in Java or C, an XML editor for building and validating XML schema definition files, a separate CVS tool for versioning support, and SQL and HTML editors if the developed application was web-based. Identifying that this situation slows down and complicates software development projects, multiple vendors like Oracle, Microsoft, IBM, Borland, and later NetBeans produced the idea of integrated development environments – IDEs.

"An IDE is generally a programming environment that has been packaged as an application program, typically consisting of a code editor, compiler, debugger, and graphical user interface (GUI) builder." [106]

Beside these fundamental functionality provisions, IDEs also provided tools for less important uses at the beginning of their evolvement, e.g. profiling, refactoring, and testing. The growth of IDEs including more and more of small helpers, which became unusable when their use cases exceeded the basic tasks, still forced developers to use several vendor-specific and specialized software for development. This evolvement led Caspar Boekhoudt to compare it with the "Big Bang" theory of the universe [4].

Nowadays IDEs have a modularized architecture and can are customizable in any way. For example, the NetBeans IDE can be downloaded in various versions providing capabilities for specific domains, e.g. basic Java development, SOA development, etc. More functionality can be added via plug-ins. However, although the maturity of plugged-in tools improved and the physical growth of IDEs seems under control it was experienced in the EAT project that IDEs are still far away from being a "truly Schwarzeneggian" [4] development environment as the unavoidable use of XMLSpy and Eclipse proves.

Due to the integration of editor, compiler, debugger, and version management in one environment an IDE was chosen for the development of EAT.

7.1.5.2 NetBeans IDE

For the implementation of EAT the NetBeans IDE 6.1 was chosen. NetBeans IDE is a free, open source, integrated development environment that is part of SUN's software portfolio. It is implemented in pure Java and hence installable on all platforms pre-

suming a Java Virtual Machine. NetBeans IDE is dual-licensed under CDDL (v1) and GPL (v2) with Classpath Exception [70].

Referring to [106] the IDE was developed in the mid-1990s by Roman Stanek and other undergraduate students in the Czech Republic. The name of the IDE was Xelfi. Stanek later founded NetBeans for selling the new IDE. NetBeans was acquired by Sun in 1999 and declared as open source a year later. The main purpose was the popularization of Java with a Java-based, open IDE. Besides the free NetBeans IDE there were two commercial products called Sun ONE Studio and Sun Java Studio Creator with additional functionality, e.g. web development. Nevertheless, since November 2005 all products were available in NetBeans as plug-ins.

As *Fig. 19* shows, the modular architecture of NetBeans IDE makes it possible to provide multiple variants of the IDE depending on the domain (e.g. desktop application, web application, composite application, etc.) and chosen programming language for a development project. For the EAT project the Java SE bundle was chosen as EAT is a Java desktop application.

NetBeans IDE Download Bundles

NetBeans Packs *	Web & Java EE	Mobility	Java SE	Ruby	C/C++	Early Access for PHP	All
Base IDE	●	●	●	●	●	●	●
Java SE	●	●	●				●
Web & Java EE	●						●
Mobility		●[1]					●
UML							●
SOA							●
Ruby				●			●
C/C++					●		●
Early Access for PHP						●	
Bundled Servers							
GlassFish V2 UR2	●						●
Apache Tomcat 6.0.16	●						●

Fig. 19. NetBeans IDE download bundles with according feature packs, taken from [70]

Why NetBeans IDE?

There were several reasons for choosing NetBeans as the preferred IDE in the EAT project instead of Eclipse or Borland JBuilder.

First, it was reasonable to choose NetBeans IDE because NetBeans Visual Library was the selected graph library (cf. 7.1.4 and 7.2). Thus, it was expected that the interplay of two NetBeans products is less problematic and more mature. In addition, NetBeans as Open Source software provides a large community, comprehensive documentation (tutorials and samples as well as documents) and an extensive pool of plug-ins in case additional capabilities are needed. Auto-update functionality and the provision of three perspectives on a project are supporting capabilities, too.

Secondly, Eclipse did not provide a graphical user interface (GUI) builder out of the box as JBuilder and NetBeans IDE did. The presence of a GUI builder was rated very high because it could save lots of time during the modeling of a user interface, especially when it comes to handle Swing-Layouts. A GUI builder manages layouts automatically and provides palettes of typical Swing or AWT elements, which can be dragged and dropped onto a form representing the background of a window. Examples for these components can be buttons, menu items, text areas, and many others.

Furthermore, attributes like height, width, name, etc. can be edited easily. Subsequently the according code is generated. The GUI builder of the NetBeans IDE referred to as "Designer", can be seen in *Fig. 21*. It shows the palette with exemplary elements like Swing Menu Items. Moreover, the property window with the "text" attribute of the selected "Entity" button is shown. In the middle of the figure, one can see the form, which represents the abstract modeler user interface.

Finally, JBuilder, specifically Turbo JBuilder 2007, was dismissed because it is proprietary and ponderous. While there were no extraordinary requirements to the build-in capabilities of the IDEs, both fulfilled them by providing an editor, a debugger, a compiler, and version management. However, looking at soft goals like performance and costs NetBeans IDE did better than Borland JBuilder did. NetBeans is available free whereas JBuilder only provides a feature-reduced, free version named Turbo JBuilder. Moreover, the free Turbo JBuilder 2007 packages were up to ten times larger than comparable NetBeans IDE bundles and thus seem to represent the Big Bang philosophy that Caspar Boekhoudt postulates. That may be one reason why Turbo JBuilder 2007 proved to be slower than NetBeans IDE 6.1.

Getting Started with EAT and NetBeans IDE 6.1

Having used NetBeans IDE in the EAT project had required some software prerequisites. First NetBeans IDE 6.1 demanded at least Java Development Kit (JDK) 5 and a local subversion client in order to use the versioning plug-in. More information on how to setup the IDE for using subversion can be found here: [76].

EAT was implemented as a Java Desktop Application project in NetBeans IDE. In order to implement and compile it additional libraries were necessary:

- castor-1.2.jar
- commons-logging-1.1.jar
- smile.jar
- xerxes-2.8.0.jar
- org-netbeans-api-visual.jar (from Visual Library 2.0 pre3)
- org-openide-util.jar

Libraries can be added via right-click on project → Properties → Libraries.

Fig. 20 does not show the standard view but the user interface in the debug mode as it only extents the standard view with the green bordered tabbed pane. Here one can find additional information from the debugger, e.g. current local variables or set breakpoints. As local variables could also be objects, it is possible to extend them to see the object's attributes. With the continue button to the right of the disabled pause button in the icon bar the program will be executed until it reaches the next breakpoint. The stop button (red square) terminates the program.

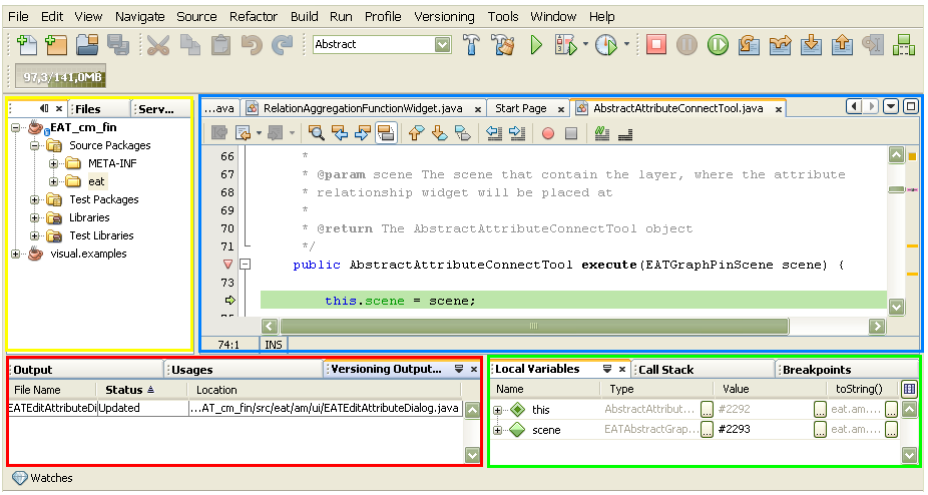


Fig. 20. NetBeans IDE user interface (debug mode), red: version management information; green: debug mode information (active local variables); blue: main editor; yellow: project explorer

If the program should only be built and executed without debug mode, the use of the green triangle button will be sufficient.

The red-bordered part in the lower left corner of the IDE interface gives additional status information of the executed program (Output – also serves as the standard output) and about versioning activity status (Versioning Output). Moreover, it is the standard tabbed pane for displaying results from other activities, e.g. search queries.

The yellow-bordered pane in the upper left area represents the project explorer where all projects with all its subdirectories and files are listed. The small blue barrel in the lower right corner of the cup icon of the "EAT_cm_fin" project indicates that this project is under version control. Via right-click on the project, a directory, or a file various features can be accessed, e.g. refactoring methods, JUnit Test creation, and CRUD operations.

The blue-bordered part, the upper right pane of the interface, displays the files that have been edited recently. Each file is presented as a tab. In the editor window, the actual coding takes place. The developer is supported by code folding, profiling, camel case code completion, and code template insertion, just to name a few.

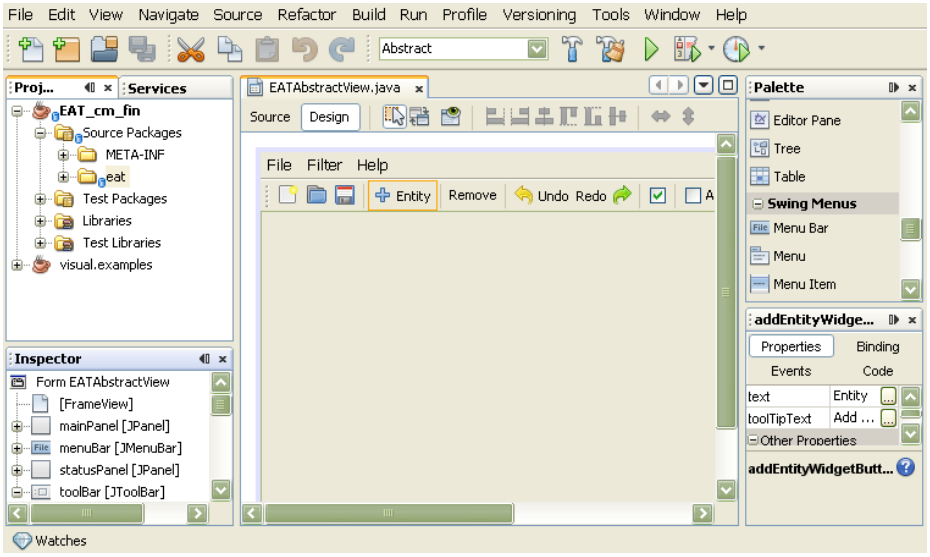


Fig. 21. NetBeans IDE Designer perspective

This section examined why it was necessary to use an IDE for the development process. It also showed that NetBeans as an open source integrated development environment was more suitable for the EAT project than Eclipse or Borland JBuilder. It is available free, provides a large amount of documentation, an active community, a modularized architecture, and goes well together with the Visual Library 2.0 pre3. Additionally important features like editor capabilities, debug mode, and version management of NetBeans IDE were sketched with the help of an example.

7.2 NetBeans Visual Library

NetBeans Visual Library is a high-level drawing library chosen for implementing the user interface of EAT. It is built on top of Swing and Java2D and requires JRE 5.0+. This general visualization library is especially "… designed to support applications that need to display editable graphs …" [108] such as UML diagrams.
As 7.1.4 shows, it is the best graph visualization library/framework that has been evaluated. This is due to the superior functionality, the clean, structured, and extensive documentation and the reliable and active developer community.

This chapter performs a closer look at the architecture and the capabilities of the library. For this, important concepts and classes are described.

7.2.1 The Architecture

David Kaspar, Chief Engineer of the Visual Library API describes the architecture of the library as follows:

"The API provides a set of reusable pieces - widgets. By composing them, you are creating a visualization. Each widget has various properties including layout, border, assigned actions, [etc.]. The library contains a set of pre-defined widgets that can be extended. All pluggable pieces are declared as interfaces or abstract classes - WidgetAction, Anchor, AnchorShape, PointShape, Animator, Border, GraphLayout, LookFeel, Layout, SceneLayout, Router, [and] CollisionsCollector. Also they all have their built-in implementation." [69]

The introduction of the official documentation says,

"The programming style is similar to Swing. You are building and modifying a tree of visual elements that are called Widgets. The root of the tree is represented by a Scene class, which holds all visual data of the scene. Since neither Widget nor Scene is an AWT/Swing component you have to call Scene.createView method for creating a Swing component which renders the scene. The created `JComponent` *could be used anywhere."* [73]

The following figure taken from [108] shows the top-level class structure of the library.

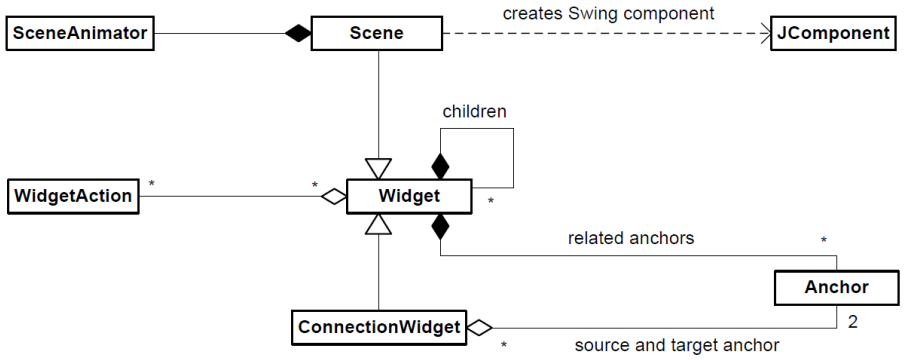


Fig. 22. Class diagram of the NetBeans Visual Library

Fig. 22 confirms the above descriptions of the architecture. The central class in the architecture represents a Widget that is always part of the Scene and can have multiple Widgets. The Scene is the bridge to Swing by providing a `JComponent` via the `createView` method. Moreover, Widgets representing UML classes for example need to be connected to each other. Therefore, `ConnectionWidgets` – specializations of Widgets – are used. `ConnectionWidgets` hold source and target anchors

that are related to the source and target Widget. The role of the `SceneAnimator` and `WidgetAction` classes is explained in the subsequent section.

7.2.2 Capabilities

This paragraph introduces important classes mostly referring to [73]. Important functionalities and concepts are explained and finally a short mapping of the Visual Library capabilities to the EAT architecture is shown.

Widget
`Widget` is the class that provides the central visual element. According to the documentation, it is similar to a `JComponent` in Swing. It holds information about its location, boundary, preferred location/boundary, preferred/minimal/maximal sizes, layout, border, foreground, background, font, cursor, tooltip, accessible context, etc.
A Widget can serve as a container for other Widgets. Mapped to the tree structure, this means that it is the parent of other Widgets. There are many specializations of Widgets. For instance, a `LabelWidget` simply holds a label and is used for displaying lines of text. An `ImageWidget` represents a static or animated image, which can be manipulated. It is possible to implement custom specializations of the Widget class. Each widget can have `WidgetActions` assigned. Often used methods for managing and manipulating widgets are `getParentWidget`, `addChild`, `isVisible`, and `getLocation`. An important Widget specialization for graph-oriented modeling is explained in the following paragraph.

ConnectionWidget
This Widget represents a path between a source and target Widget. The source and target are specified by anchors. The appearance of the connection can be changed by customizing the line color, the anchor shape (e.g. `AnchorShape. TRIANGLE_HOLLOW` or `AnchorShape.TRIANGLE_FILLED`) or the router. By default, there is a direct router, which routes the path in a straight line from source to target. Nevertheless, an orthogonal router can be used instead. It sets multiple control points forcing the path to flow around other Widgets for instance. Control points can be set and deleted manually, too. For more information about the class `Router` and routing policies, classes refer to [73].

LayerWidget
Another specialization of Widget is the `LayerWidget`. It represents an important concept in the implementation of EAT. Usually not all widgets of every kind are added to the scene directly. For the purpose of separation concerns, several layers are introduced. A `LayerWidget` "… represents a glass-pane - similarly to JGlassPane from Swing. It is transparent by default." [73] By implementing LayerWidgets in EAT it was bound to the recommendation in the documentation:

- **backgroundLayer** for temporary background interaction widgets […]
- **mainLayer** for main widgets,
- **connectionLayer** for connections […]
- **interactionLayer** for temporary foreground interaction widgets created by decorators of interactive actions, e.g. grid[…]

Scene
As mentioned before, this Widget represents the root node of the Widget tree. It is a specialization of the Widget class and contains additional functionality for controlling the whole scene, its views, repainting and tree validation. The Scene class is extended by multiple classes, where GraphPinScene was the most relevant for our purposes.

The GraphPinScene class is especially suitable for graph-oriented modeling. It manages a model with nodes, pins and edges. A pin is always attached to a node. An edge could be connected to a source and a target pin only. This class uses generics, which makes it possible to specify one's own implementation of nodes, pins, and edges. The class is abstract and has to be extended. Important methods of this class are `addNode (Node)`, `addPin (Node, Pin)`, and `addEdge (Edge)` that call appropriate methods for registering and validating new widgets on the Scene. According methods for removing nodes, pins, and edges exist, too. As described in 7.4.3.2 the Scene is an important management class and part of the controller referring to the MVC architecture. It is the linking instance between the data model and the view.

WidgetAction
WidgetActions define the behavior of the widget, which they are assigned to. Actions are usually created through the `ActionFactory` class. Factory methods usually require a `Decorator` or `Provider` parameter. "Decorator is used for specifying the visual appearance of objects temporarily created by an action. E.g., RectangularSelectAction has RectangularSelectDecorator for acquiring the widget that will represent the current selection rectangle. Provider is used for specifying the custom behavior of an action. E.g., EditAction has EditProvider with edit method, which is called when a user double-clicks on a widget where the action is assigned. It is up to the EditProvider implementation whether it invokes an editor or run an application or does something very different. Usually there are default implementations of the decorators and providers available in ActionFactory class." [73]

All Widget's actions are grouped in a chain that is hold by a Widget. In this chain new actions can be added, old actions can be removed and all actions can be resolved by the `getActions` method.

Popular actions are for example `MoveAction` (drag and move widgets), `EditAction` (edit widget on double-click), and `PopupMenuAction` (opens a popup menu supplied by the provider parameter).

High-Level Capabilities
NetBeans Visual Library provides functionality that can be added with only a few lines of code. These built-in capabilities supply useful features that do not need to be implemented anymore.

As an example, the previously described `ActionFactory` offers a `ZoomAtion`, which is usually assigned to the Scene and provides zoom-in and zoom-out via mouse scroll-wheel / middle-button. Moreover a `SceneAnimator` class provides 500ms-long animations, for instance set the preferred location of a widget in an animated way. In addition only few lines are needed to export the Scene or the current view into a PNG-file [73]. One last feature worth mentioning is the `createSatelliteView` method of the Scene class which supplies an overview of the scene including a view-port that shows a rectangle representing the viewable area on the scene (e.g. if the Scene is larger than the window and hence the frame creates scrollbars).

Visual Library in EAT
The Enterprise Architecture Tool provides animation, zoom, export to PNG, and satellite view capabilities. Moreover, actions facilitating multiple selections, in-place editing, expanding lists, and adding/removing/moving control points are used.

The class structure in EAT sticks very close to the Visual Library defaults. An overall EATGraphPinScene defines the previously explained `LayerWidgets`. This scene is extended by the abstract and concrete representations. The abstract modeler uses `AbstractEATGraphPinScene` as the Scene provider; the concrete modeler uses `ConcreteEATGraphPinScene` for that. The Scenes know appropriate `JComponents`, called Views (`EATAbstractView`, `EATConcreteView`), that supply the user interface and create the satellite view, for example.

The main Widget in EAT is the Entity Widget, and its abstract and concrete specializations. It contains actions like `createSelectAction`, `createMoveAction`, or `createPopupMenuAction`. Entity Widgets hold multiple children e.g. Attribute Widgets and `LabelWidgets`.

The `EATConnectionWidget` extends the `ConnectionWidget` class and serves as the base for `EntityRelationshipWidget` as well as `External`- and `InternalAttributeRelationshipWidget` and their abstract and concrete correspondents.
The Scene has methods that register or unregister respectively add or remove main and connection Widgets.

This chapter introduced the characteristics of NetBeans Visual Library as a general visualization and graph-oriented modeling library. It explained its architecture with Scene and Widget as the main components and gave examples for built-in functionalities (zoom action or satellite view). Finally the mapping of the intended use case architecture to the EAT class structure showed the applicability of the library.

7.3 XSD and Model Structure

In this paragraph, the use of the XSD documents (cf. 7.1.2) and the way the implementation benefited from their application is described. It is followed by an explanation why two models were necessary. Finally, the abstract and concrete models are explained.

7.3.1 Motivation - The Intended Way of Use

For the tool two XSD files were used, one delineation per model[7]. In this use case the style of the models created with the tool (see 7.3.3 for a detailed view on the model structures) were specified in the XSD files. The schemas were considered as Meta models.

Application of an XSD schema to build modeling tools is considered obvious, as the functionality provided by an XSD complies very well with typical modeling conventions. The ability to represent cardinality (e.g. the amount of attributes within an entity can vary between zero and infinity) and generalization (which is found in generalization of entities) was adopted. The hierarchical structure defined by an XSD is reflected in the Scene structure. A refinement in the XSD is mapped to a refinement in the Meta model.

The abstract and concrete models were created based on XSD[8] to be flexible and adaptable. Additions in these Meta models were rapidly transferred to the model's Java classes and in this way to the model's structure.

The Castor generated classes are a one-to-one representation of the model definitions. Therefore, a valid instantiation of the classes is reflected in compliance with the model structure.

The model saving is eased: Castor (cf. 7.1.3) offers functionality to serialize its instantiated classes to XML documents. The loading of models is also controlled by Castor; Java classes are instantiated based on their corresponding values within XML documents.

During the implementation process and with regard to future changes, Meta model extensions and restructuring are handled easily and quickly. The demand for manual implementation activity was reduced. An advantage is achieved when typical refactoring tasks, such as pull up of fields or generalization, are performed.

A certain model (stored in an XML document) can be processed by any tool that is able to handle XML documents, so it can be reworked and adopted when needed.

Concrete or abstract models are helped in getting compliant again manually when they got incompatible through Meta model changes.

---

[7] Section 7.3.2 explains the need of two separate files

[8] Chapter 7.1.3 describes the way from XSD to Java classes with the use of Castor

7.3.2 The Need of Two Separate Models

In the previous version of the tool, two separate XSD files were used. The abstract model's content was described in a schema, the concrete model's structure was determined in a second one. Redundancy between these two structures was created and an understanding of both definitions was needed to change the modeler. Thus, a reorganization of these underlying concepts was discussed.

The initial idea discussed was to build a unified XSD structure for both models. Existing code from the abstract modeler could have been reused in the concrete modeler. Super types of the elements, on which the tools (see 7.4 for an overview of the tool concept) could work, were planned in order to make this possible. The idea was that general modeling elements were introduced, from which the special elements of the abstract and concrete model would be derived.

Unfortunately, several challenges were found, while attempting to introduce such a super element structure.

Extension or restriction of defined types is not allowed by the XSD structure. Restriction was only found to be feasible on the value range of an element rather than on the type of an element. The use of this restriction is considered necessary to make sure that a subtype would only accept elements of another subtype.

Furthermore, the necessary overriding capability, by extension, turned out to be impossible with respect to the project resources. The introduction of most of the elements on a very low subtype level was considered as the consequence. This means that a similar structure as seen in the previous version of the modeler was expected.

Because these issues were considered very limiting, several possible solutions were considered:

7.3.2.1 Use of General Model with Given Problems

The introduction of generalization classes that serve only as types could be a solution. The introduction of attributes in subtypes was expected to be the consequence. A type-certainty seemed possible on this way, but no real benefit was expected. The super types neither would provide attributes nor needed signature for their use. No difference in later usage compared to the existing versions of the model files was expected.

7.3.2.2 Development of a Completely New Model

Another possibility was seen in the creation of an entire new approach implementing the model structure. This could be a data structure completely written in Java with the usage of inheritance, overriding, and multiple-inheritance by the usage of interfaces. The desired uniform model structure was expected as the result. The abandonment of the use of Castor was implied this way. The loss of advantages provided by Castor (cf. 7.1.3 and 7.3.1) was stated as the costs of this approach.
The loss of an underlying Meta model was especially seen as a very high price.

7.3.2.3 Staying with Current Model and Separating Code

Due to the lack of benefit from building a new model structure, staying with the existing one was considered in detail. Two ways of handling the differentiation of the abstract and the concrete modeler were identified.

The first way could be to copy all existing classes to a `cm` package and to start to adopt all occurrences of the abstract model elements to concrete model elements. A completely new branch with heavy code doubling was expected as the result. The setting apart of the two branches, which was seen as the start of two separate applications, was a concern.

The other way is explained in the following subsection.

7.3.2.4 Staying with Current Model, Enforcing Code Reuse

The other possibility of handling the different parts of EAT while staying with the existing model structure was seen in the approach to extend the existing tools in a way that was applicable for elements of both models. The formation of two parts in each tool was seen as the consequence. The abstract model was handled by one part, the concrete model by another part. The avoidance of doubled code parts was not expected to be feasible in all tools. On the other hand, side-by-side synergy effects were expected to be more obvious. Individual decisions in each situation were expected to be necessary, especially when it comes to the point where general and special code was disjoined.

The same functionalities were still found at the same place, which was seen as a great benefit. An identification of the relevant points, which were influenced by changes, was expected to be very fast.

7.3.2.5 Conclusion/Final Decision

In the end, the application of the current XSD data structure was determined to be the best way. The use of Castor was seen as elementary and important. Its benefit was considered not achievable manually. An enlargement of the tools, to make them utilizable for abstract and concrete models, was agreed. Having one single point of truth was seen as best fitting to the Object-Oriented Software approach that the tool was designed to follow. With regard to further extensions, this decision was seen to be very understandable and adoptable on the other hand. The functional classification was kept. However, more Java-typecasts and `instanceof` checks had to be implemented. That way, many already existing algorithms were reused, while the data structure was extended.

The practicability of this solution was also seen as a large benefit, especially as no familiarization with XSD serialization and creation by hand was needed.

7.3.3 Models

A detailed consideration of the abstract and concrete model is presented in this chapter. In comparison to chapter 6, where the different models were explained and motivated, this chapter focuses on the contribution of the XSD documents to the modeler. The XSD files are attached in the appendix (Appendix B: XSD).

In section 7.3.1 it was said that the rules of the models were described in the XSD schema. Not only the elements described in chapter 6 are meant in the term “model” in this case. Non-functional properties of the modeled elements were also specified in this rule-set. To give an example: the location of a certain entity on the Scene is said to be non-functional. The model's information will not be influenced if the position of an entity is changed. On the other hand, a clear view is created when unique positions were set. A consistent structure of the model is achieved when the entities are positioned identically each time the underlying model is loaded.

All possible constructions and valid combinations of elements were described in an XSD document. No matter whether visible or not, functional or non-functional, optional or obligatory, each part of a model must have been defined in the schema so that proper models were created. In the following paragraph, all specified elements are named and classified in the context of their defining structure.

With this compendium, a complete and full description of the modeler's ability is engendered.

7.3.3.1 Abstract Model

Abstract models, which were described in chapter 6, were used as the foundation of multiple concrete models. Structures that were modeled in the abstract modeler were instantiated in the concrete modeler. A consistent processing in the concrete model was achieved by consequently observing modeling rules in the abstract modeler. These rules are explained in this subsection.

In the abstract modeler, each model consists of one to infinity entities. An entity is composed of a name, a unique ID, numerous tags, a collection of attributes, and a certain position, which was expressed through X and Y coordinates. An attribute was specified to have a name, a unique ID, and numerous tags as its enclosing entity was already determined to have. In addition, an Aggregation function is required. The structure of the attribute’s CPM (6.2) is described in an Aggregation function. Names of the states, the optional CPM, and one to infinity Priors are attached to an Aggregation function. To combine the CPM's content Adder-, “Exclusive Or”-, Maximum-, Medium-, Minimum-, Negation-, Parametric-, Static-, and Weighted-function are allowed as Aggregation functions.

For each entity, zero to infinity connections to other entities or itself can be specified. Again, these relationships consist of a name, a unique ID, and numerous tags. For a relationship between entities, it is necessary to have a defined origin and target entity; multiplicity information is needed on both ends. Names of origin and target are allowed to be annotated optionally.

Connections between attributes were also implemented. A separate treatment of attribute relationships of a certain attribute compared to attribute connections of two involved entities was realized. Relationships between attributes of a single entity are called internal attribute relationships. They were attributed with ID of origin and target attribute, a flag to show whether a causal relationship was modeled, and optionally dedicated to an entity relationship between the entity and itself.

The external relationships were designed as opposite to the internal ones. Their properties were specified to be the same as the properties of their oppositions. Additionally zero to infinity entity relationships are allowed to be appended. That way, the indirect attribute relationship (6.4) was realized. Each attribute connection was specified to have an optional relationship Aggregation Function. Therefore, Average-, Maximum-, Medium-, and Minimum-function can be chosen as valid relationship Aggregation Functions.

A small example of a possible model can be seen in *Fig. 23*. The entities "Entity 1" and "Entity 2" are connected through an entity relationship. These entities are represented as EntityWidgets. Their attributes, which are wrapped in attribute Widgets, "Attribute 1" and "Attribute 3", are also linked through an attribute relationship (boxed in an Attribute Relationship Widget), which is based on the entity relationship (Entity Relationship Widget). A Maximum relationship Aggregation Function is selected for this attribute relation. "Entity 2" is tagged with the tag "Entity Tag 1", whereas "Attribute 3" is tagged with "Attribute Tag 1".

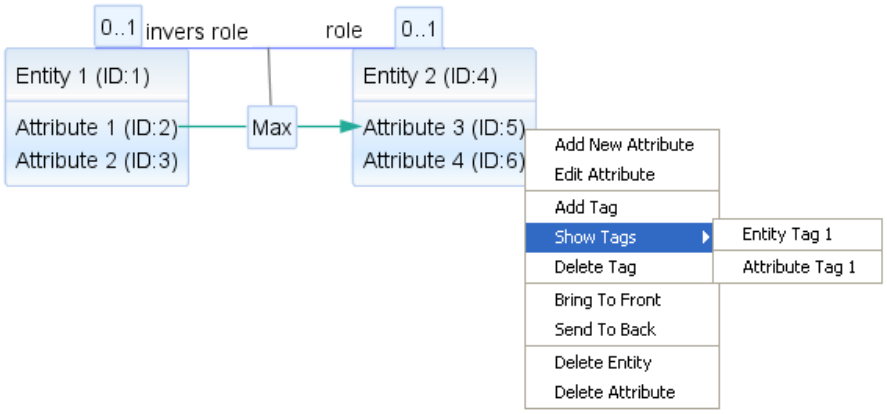


Fig. 23. Abstract Model Example

7.3.3.2 Concrete Model

A certain abstract model gets instantiated in the concrete model. Therefore, many elements of the concrete model are provided with a reference to an element of the abstract modeler.

The concept that a concrete model is based on an abstract model is because the concrete elements either are a reuse of abstract elements or are defined very close to them.

As it has been explained for the abstract model (7.3.3.1), entities in the concrete model are specified as the base of the models. Their properties are the same as the ones in their abstract counterparts. The name of the underlying abstract entity was added, so a connection between them was established.

A change was made, when the attributes of an entity were defined. They were specified to have a name, a unique ID, numerous tags, but additionally the CPM, which is taken from the corresponding abstract attribute during construction. Calculated values and evidences were added as optional properties. Calculated value stands for a mapping between a numerical value (its probability) and a certain state. Thereby a CPM, a state, and a source are stored for evidence.

Entity connections are hold in a shortened way. The ID of the origin and the target, as well as the reference to the abstract entity relationship is stored. A similar approach was chosen for the attribute connections. Besides ID of origin, target, and underlying entity relationships, only the ID of the abstract attribute relationship is saved.

7.4 Architecture

This chapter deals with the main architecture of the application. At first, the boundary conditions are shown and an examination of the concept's evolution is given. Afterwards the final architectural design is presented and illustrated in detail.

7.4.1 Interaction Design

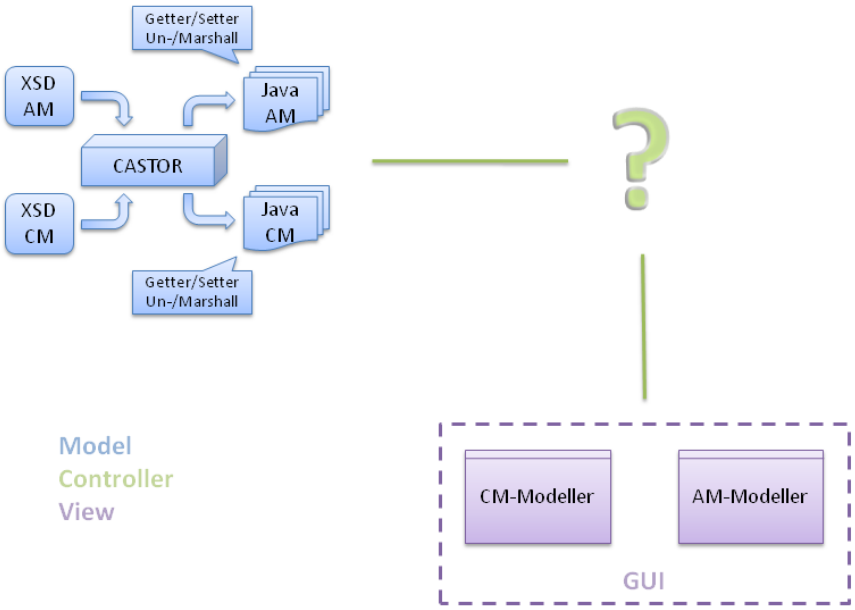


Fig. 24. Open Decision

Following the Model-View-Controller approach, the EA Tool implementation has been divided into three parts. The first part is the Model-Part or Data-Part where the data structure of the model is located. The second part is the View-Part where all visualizations and user interface elements are handled. Finally, the third part is the Controller-Part. In the Controller-Part, a connection between the Data-Part and the View-Part has to be made.

The Data-Part was determined by the usage of XSD files and the Castor tool. The structure View-Part has also been predefined, because it was the goal to implement a Java desktop application. Only the Controller-Part was designed freely (*Fig. 24*).

One main objection while designing the interaction concept was the extendibility of the tool. It was necessary to build an extendible architecture where future additions are feasible and easy to accomplish. Another objection that had to be taken care of was the straightforwardness, and intuitive structure, of the complete construction in a way that it would not take future developers too long to get an overview and start developing. Therefore, a granular functional partition of individual tools was introduced following the single point of truth [104] principle.

7.4.2 Design Alternatives

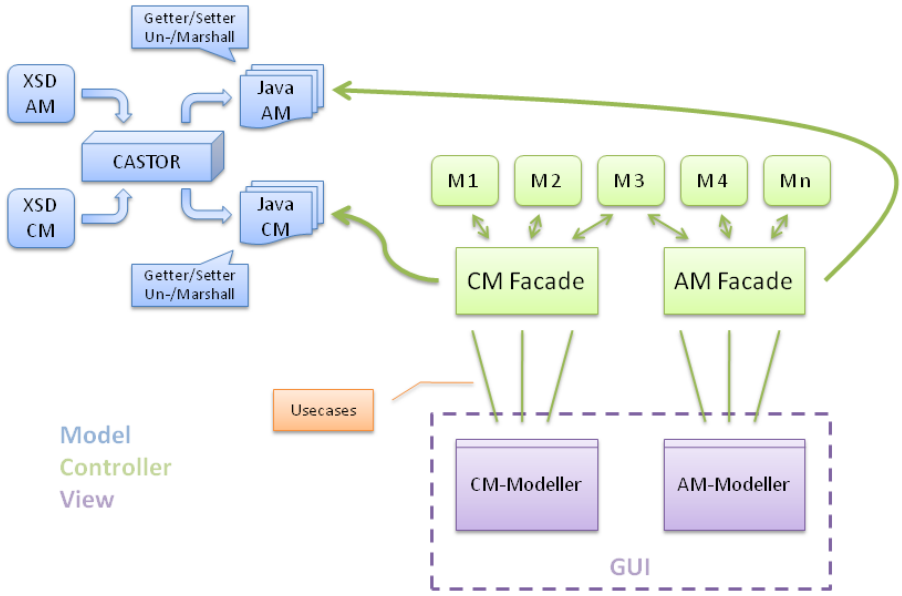


Fig. 25. Design Alternative

While designing an interaction model several alternatives were discussed.

The first alternative was to stick near the existing architectural design of the previous version of the tool. This approach consisted of one main managing class, which held all the information about the model and provided the needed functionality to manipulate the model. A partition of different use,-cases or classes of functionality was provided by the use of interfaces, which were then implemented in wrapper-classes around the main managing class. In this way, only necessary functionality for the defined use case was visible.

It has come to the point where sticking with this approach would increase difficulties while extending the functionality, because it would have been necessary to provide new interfaces with their corresponding implementation in some parts or to build new functionality into existing parts of the code.

The second alternative was to split up the functionality into more atomic parts (*Fig. 25)*. These specialized units would then be ordered thematically and functionally. They would provide all abilities to handle and modify the models. In addition, they would be placed around one main managing module, which holds all information about the model in one place. By using this design approach, it would have been possible to provide slender interfaces to smaller independent parts of the code. Extension of the functionality would have been possible by adding another functional component and its use cases to the managing class.

7.4.3 Final Design

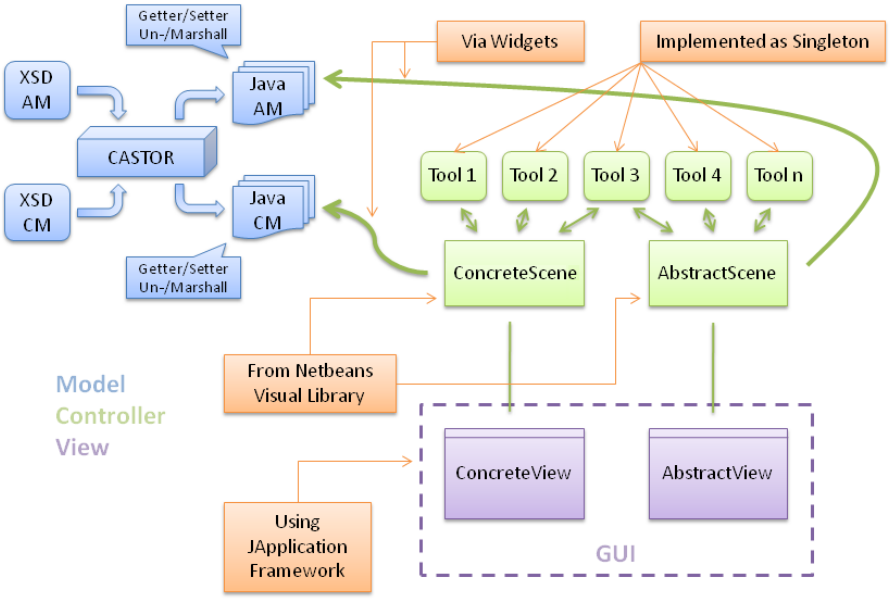


Fig. 26. Final Design

In the end a modular architecture was set up, which consisted of few main classes that handled the visualization and model managing part and several extension modules, called tools (*Fig. 26*).

These tools provide a uniform access to their functionality. They were implemented in a common way using the design pattern of a singleton. By using singletons, it was not necessary to provide a registration and handling part inside the code. Moreover, it was possible to simply add new tools to the code base and use them in the desired part of the managing classes.

The role of the managing classes is taken by the Scene as part of the NetBeans Visual Library (see 7.2). The management of the model elements and the graph structure of the model were also done by the Scene using its widgets to hold references to the model elements. By doing so the Scene and the Widgets become part of two pieces of the program structure. On the one hand they are in charge of managing the model elements, or the controller part, on the other hand they are used to vizualise the models, they belog to the view part. This is possible because Java allows to merge the controller and the view part [81].

7.4.3.1 Data Structure

The class structure of the Model-Part or Data-Part is completely auto generated by the use of Castor. Although a model is designed, no complete model exists in EAT. In fact, all elements and relations are handled by the management class. Only while loading, saving, and calculating a concrete model, a complete model as described in the XSD files is generated.

7.4.3.2 Management Classes

The Scene described in 7.2.2 serves as the main connection and management location at the same time. All model elements are referenced by the Widgets that are organized and held together by the Scene.

The other main connection and management location is the so-called view. A view is part of the Swing Application Framework [99]. The view is in charge of handling all user interactions that are not concerned with the drag-and-drop modeling features of the scene. There are methods that handle button and menu actions. In addition, loading and saving tools are called by the view.

7.4.3.3 Connection of Data Structure and Management Class (Entity & Attribute)

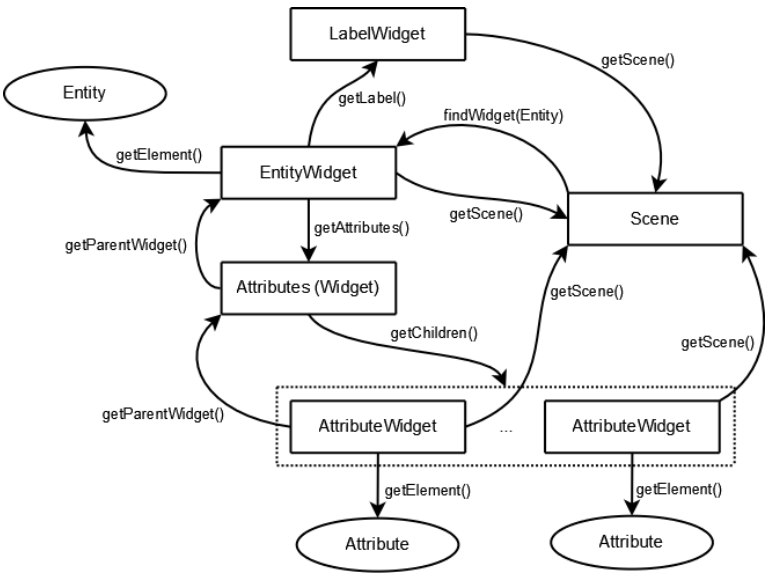


Fig. 27. Connection between Widgets and data elements (focus on main Widgets)

The diagrams in *Fig. 27* and *Fig. 28* show how the data structure of the Castor generated Java classes and the Scene from NetBeans Visual Library with its Widgets is accomplished. Castor generated classes are shown as ellipses and Widgets are shown as boxes. The arrows represent methods of the classes in which the arrows have their origin. The arrowhead points towards a class that is returned by the method annotated on the arrow.

All widgets provide a `getScene`-method, which returns the current Scene. To get hands on an attribute the `getAttributes`-method of the Entity Widget, whose entity contains the attribute, has to be called first. From the returned Widget, the `getChildren`-method returns a vector of attribute Widgets that now have references to the according attributes.

As one can see in *Fig. 27* and *Fig. 28*, the way from the Scene to a data model element such as an entity or an attribute relationship is somewhat longer. For registered objects like entities (GraphPinScene: node) and relationships (GraphPinScene: edge) the according Widgets can be resolved by the Scene's `findWidget`-method.

To find the matching Entity Widget, which contains a given attribute, it is necessary to iterate through all Entity Widgets returned by the Scene's `getChildren`-method (precisely: its mainLayer Widget which contains all Entity Widgets) and, in the lower level, to iterate through all Attribute Widgets of these entities. A for-each iteration is also necessary in order to find all Entity Relationship Widgets that are connected to an entity. The Relationship Widgets can be resolved via the `getChildren` method of the Scene's connectionLayer. Then, for each Entity Relationship Widget, it has to be checked whether the source or target is the particular Entity Widget.

These iterations mark a tradeoff between introducing additional references to the elements and accepting some more computation effort for finding an element. It was decided that no additional references are being introduced to keep the interfaces of the classes slender [99] and to stick with the already defined and well-documented access methods in almost all cases. Especially when it came to information gathering and comparison between elements in the abstract and concrete model these iterations were necessary. Since the iteration purpose and the iteration participants were mainly unique a generalization was not possible (except for using a for-each loop).

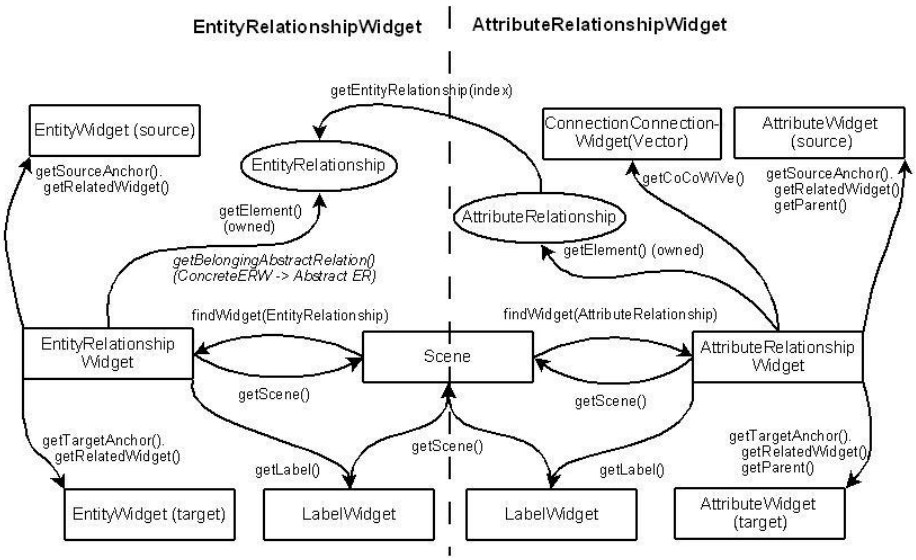


Fig. 28. Connection between Widgets and data elements (focus on connection Widgets)

7.4.3.4 Tools

In this context, the term Tool is used as a specialized piece of software that is used to fulfill predefined tasks. A Tool only offers one relevant function for one defined task, for example to handle a dialog, or to add an element to the model. By implementing the functionality in this way, it is possible to extend the application with new functions by adding an appropriate tool and using this tool in the corresponding place in the management classes. It is also possible to use a tool only from inside a tool.

The Tools should be implemented in a universal way so it is possible to use them in the abstract modeler and in the concrete modeler.

7.4.3.5 Concrete and Abstract Modeler

On one hand, the existence of two modelers is owed to the fact that the concept is based on these two models. On the other hand, the two data structures were incompatible (see 7.3.2). Because there were, only few similarities in the data structure it was necessary to implement a complete second application to draw a concrete model.

The two modelers share the main MainApp class so there is only one main-method in the whole project and it is possible to choose between both modelers by using a command line parameter at launch time.

Other commonalities between both modelers are several tools, which were implemented in a general way by using generalization classes.

7.4.4 Additional Architectural Decisions

The following chapters deal with additional architectural decisions that were made. The construction of the inheritance hierarchy of the Widgets is described. Moreover the composition of an Entity Widgets from different other Widgets is illustrated. These parts are chosen for detailed description because of their complexity and their relevance for further extension and development of the EA Tool.

7.4.4.1 Inheritance Hierarchy of Widgets

The following description is based on the inheritance hierarchy for Widgets as visualized in *Fig. 29*.

The anchor of the complete hierarchy is the Widget class, which is Part of NetBeans Visual Library. To provide a uniform access to the model element of the Widget an interface `EATElementReolver` is used. All Widgets keep a reference to a model element that is stored in an object of a class generated by Castor, e.g. `eat.am.model.Entity` in the case of an EntityWidget. The reference is stored in an attribute of the class `EATWidget`. Because all visual model elements have a name, which has to be displayed, a `LabelWidget`, which is also part of NetBeans Visual Library, is stored in each `EATWidget`. Because these entire abilities match to every Widget, no matter if it is an abstract or concrete element, an attribute, or an entity, the reference to the model element is by now of a generic type `T`.

The next level of inheritance provides a differentiation between EntityWidgets and AttributeWidgets. In consideration of the fact that there is no difference between an abstract entity and a concrete entity concerning the visualization, the construction of the EntityWidget is still done on a generic base. More information about the construction of an EntityWidget can be found in 7.4.4.2.

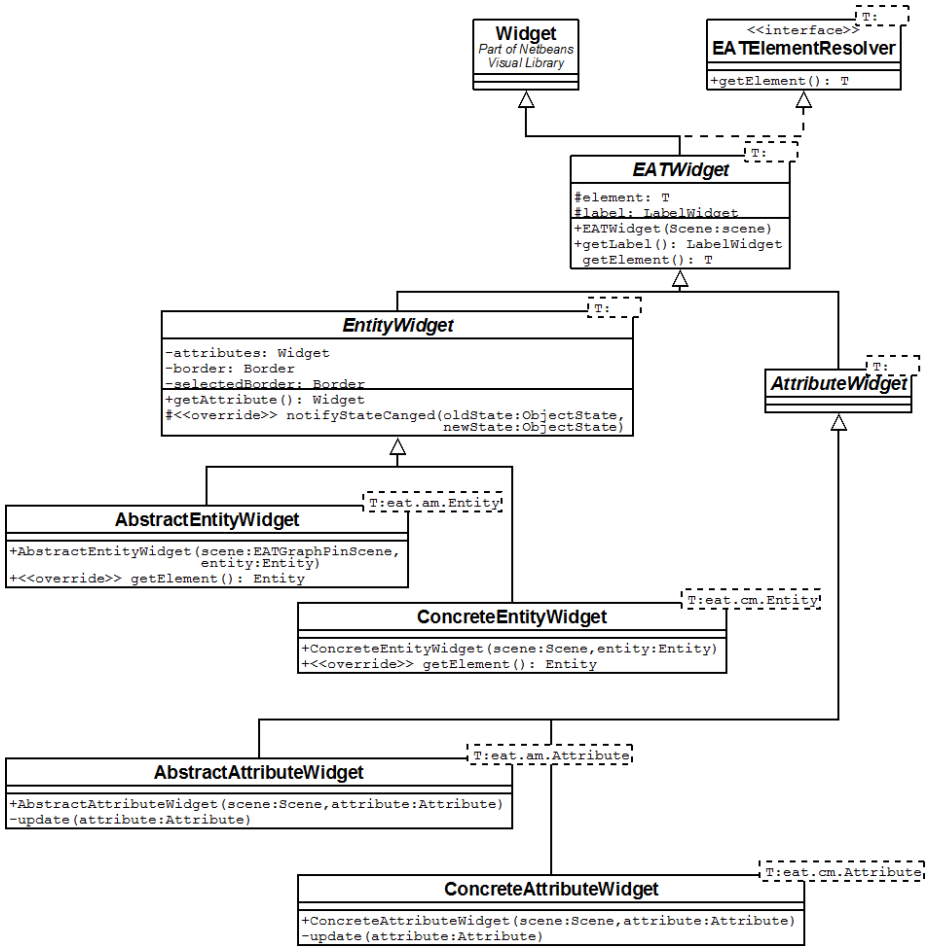


Fig. 29. Hierarchy of Entity and Attribute Widgets

Only in the final level of inheritance, it is necessary to instantiate the generic type of the reference to the model element with the actual type of the model element. This would be `eat.am.model.Entity` for an `AbstractEntityWidget` and `eat.cm.model.Entity` for a `ConcreteEntityWidget`.

Since the same considerations can be made on Attribute Widgets, they were implemented analogously to Entity Widgets, which is also visualized in *Fig. 29*.

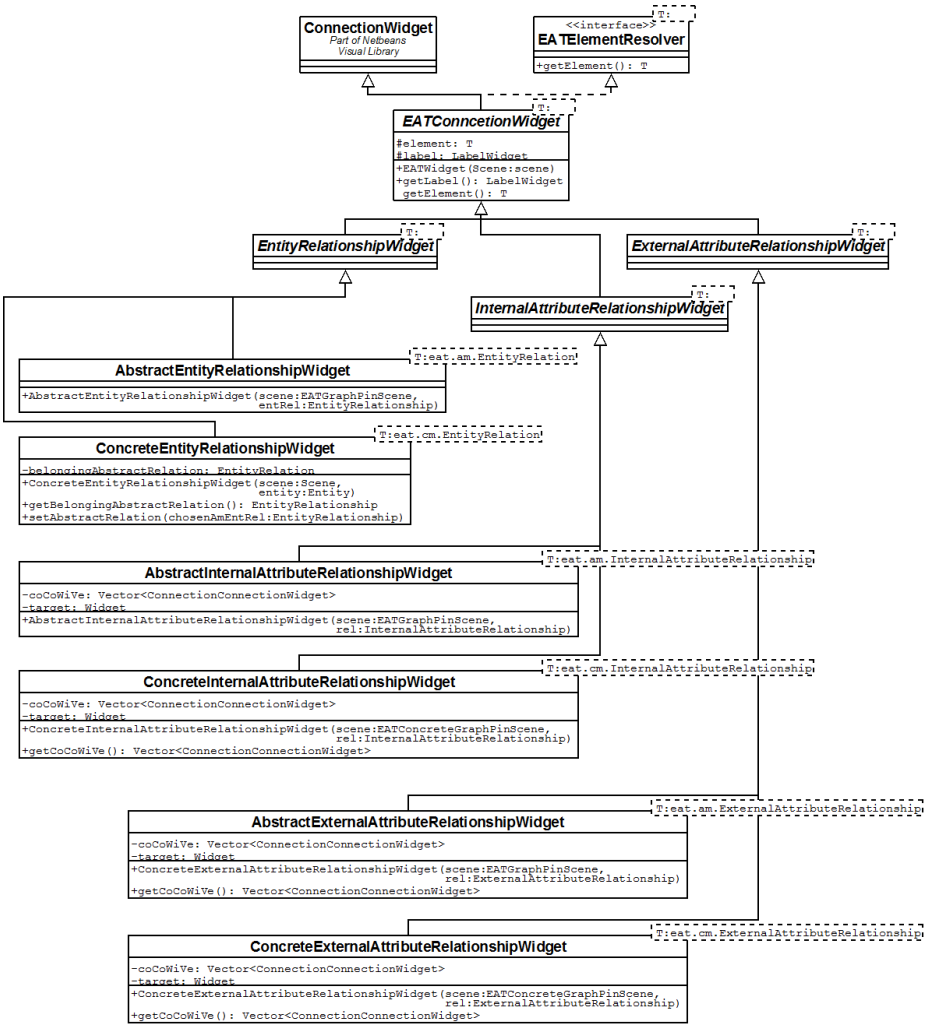


Fig. 30. Inheritance Hierarchy of Relationship Widgets

For the implementation of relationship Widgets, a similar inheritance hierarchy as described for Entity Widgets and Attribute Widgets was designed.

The structure for Relationship Widgets is shown in *Fig. 30*. All Relationship Widgets are subclasses of `ConnectionWidget` from the NetBeans Visual Library. In analogy to the other model elements, Relationship Widgets store a reference to their corresponding model element. A uniform access to this element is provided by the implementation of the same interface `EATElementResolver`, which provides the `getElement`-Method.

Due to the existence of three different relationship Types in each model it was necessary to introduce one class for each Type. There is one class for relations between entities called `EntityRelationshipWidget` and two for relations between

attributes. `InternalAttributeRelatinshipWidget` is used for attribute relationships, which connect two attributes of the same entity. `ExternalAttributeRelationshipWidget` is used for the connection between attributes of different entities.

7.4.4.2 Construction of an Entity Widget

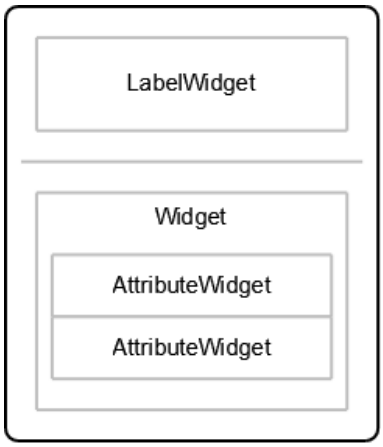


Fig. 31. Construction of an EntityWidget

The following description of an Entity Widget and its different parts suits for both Entity Widgets, abstract and concrete ones. This is because all layout and visualization tasks are handled in the class `EntityWidget` that is a super class of the special Widgets. *Fig. 31* shows a schematic diagram of an `EntityWidget`.

The outer black border is the border of the Widget. An `EntityWidget` has two borders. One default border and another one used for the visualization of a selection. Both borders are generated by the usage of the `BorderTool` (cf. 8.2.2). Inside the main Widget, a `VerticalFlowLayout` is used to arrange the inner elements in a vertical stacked layout.

In the top part, the `LabelWidget` is located. This Widget is provided with an empty border of 5-pixel width to assure a gap between the text and the border of the `EntityWidget`. The next part is the horizontal divider line that is realized by using a `SeperatorWidget` from the NetBeans Visual Library.

In the lower part of the `EntityWidget`, the attributes are shown. For organizational reasons all Attribute Widgets, which are the representation of the model element attributes, are placed inside a so-called attribute compartment. This attribute compartment is just a plain Widget of the NetBeans Visual Library and ensures that there is also a 5-pixel border maintained around all attributes.

Inside the attribute compartment, all AttributeWidgets are arranged in another `VerticalFlowLayout` to stack them vertically.

7.4.4.3 Construction of an EntityRelationshipWidget

The composition of an EntityRelationshipWidget, which will be explained in this paragraph, refers to the architecture of an `AbstractEntityRelationship-Widget`. The counterpart in the concrete modeler only differs in the absence of attached Widgets.

As *Fig. 32* shows, an `AbstractEntityRelationship-Widget` is primarily a simple line leading from a source, an (Abstract) EntityWidget, to a target, an (Abstract) EntityWidget. The relationship is connected to the source and target via appropriate anchors. In addition, the `AbstractEntityRelationshipWidget` is made up of five additional Widgets. It has two children at the source, two children at the target, and one child, a LabelWidget, at the center of the connection. This center label is invisible and used for debugging purposes. The text of the label contains the ID of the relationship. The second and more important purpose of this LabelWidget is that it serves as the target Widget for the `ConnectionConnectionWidget`. This kind of Widget is displayed as a line between an `AbstractAttributeRelationshipWidget` and all of its related `AbstractEntityRelationship-Widgets`.

The `AbstractMultiplicityWidgets` at the source and target end of the connection let the user choose which cardinality should be assigned to the involved elements. The composition of an `AbstractMultiplicityWidget` will be explained later.

The `LabelWidgets` attached to the line hold information about the role names referring to the source and target EntityWidget. It is a simple text and can be modified at design-time.

7.4.4.4 Construction of an AbstractMultiplicityWidget

`AbstractMultiplicityWidgets` can be found at the source and target end of an `AbstractEntityRelationshipWidget`. With the help of this widget, the user can choose between four cardinality choices: “1”; “0..1”, “0..*”, and “1..*”. This widget is an expandable widget. More information about the manner of operation can be found in 8.2.2.

Fig. 33 illustrates that the `AbstractMultiplicityWidget` appears on the scene in two different states. If it is collapsed, only a Widget that contains a `Label-Widget` is displayed. In this `LabelWidget`, the current chosen multiplicity is shown. After double-clicking this Widget, it expands itself. In addition to the previously stated Widgets, it shows a `Widget`, named `detailsWidget`, that contains four `ComponentWidgets`. In this case, these `ComponentWidgets` are Visual Library Wrapper for Swing’s `JRadioButtons`. These buttons provide the four

choices and are grouped in a `ButtonGroup`, which means that only one of them can be selected at a time.

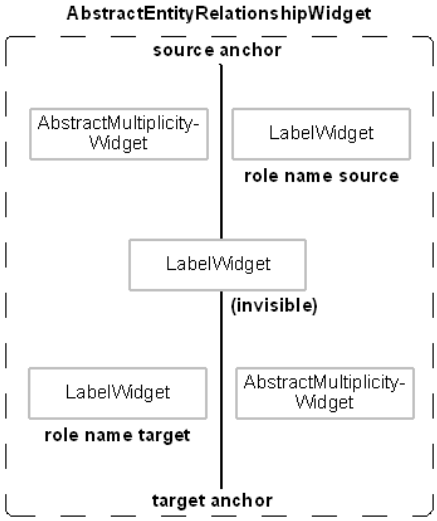


Fig. 32. Composition of an EntityRelationshipWidget in the Abstract Modeler

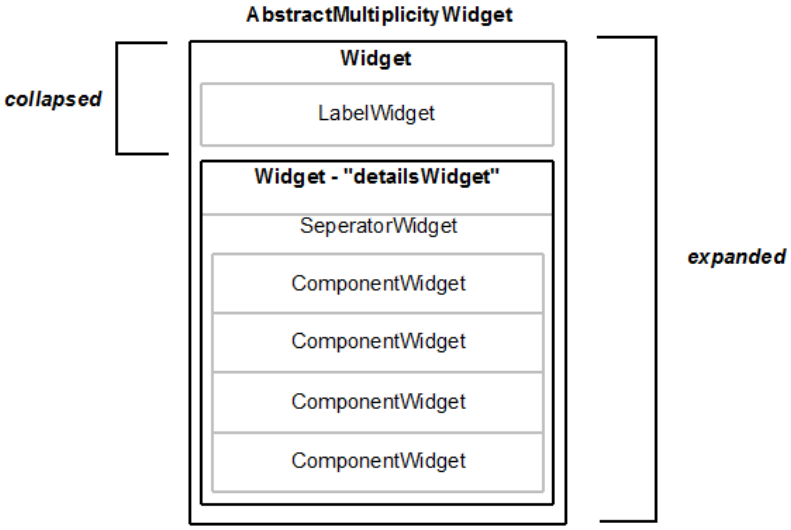


Fig. 33. Composition of an AbstractMultiplicityWidget

7.4.4.5 Construction of an AbstractExternalAttributeRelationshipWidget

Similar to section 7.4.4.3 it is referred to the most complex representative of the AttributeRelationshipWidgets, the `AbstractExternalAttributeRelationshipWidget`, to describe the composition of this kind of widgets. This is because the architecture of all other RelationshipWidgets can be deduced and understood from the explanations of the `AbstractExternalAttributeRelationshipWidget`.

Fig. 34 illustrates that the line between source and target `AttributeWidget` has two centrally attached widgets. The invisible `LabelWidget` serves nearly the same purpose as the invisible central `LabelWidget` of the `AbstractEntityRelationship-Widget`. However, there is one difference: this LabelWidget is the target, not the source, of the ConnectionConnectionWidget.

The other central widget is a `RelationAggregationFunctionWidget`. Like the `AbstractMultiplicityWidget`, this is another expandable widget. The composition of this widget is very similar to the architecture of the `AbstractMultiplicityWidget`. The only difference is that the `RelationAggregationFunctionWidget` provides choices for aggregation function types: “AVG”, “Max”, “Min”, and “Med”.

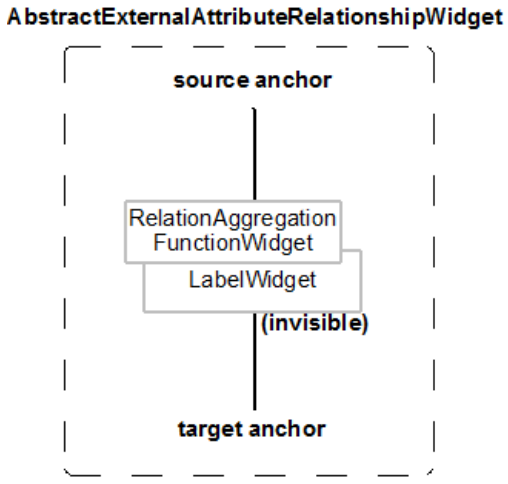


Fig. 34. Composition of an AbstractExternalAttributeRelationshipWidget

PART III – The Enterprise Architecture Tool

This part describes the implementation of a software tool, called Enterprise Architecture Tool, which uses the theoretical concepts and implementation proposals presented in Part II. The foundation of the theoretical concept is the enterprise architecture analysis method. This method includes the creation and visualization of abstract models. Abstract models are instantiated by creating concrete models based on collected evidence. Their computation, by using Bayesian networks, allows an analysis of different scenarios. These scenarios are used to examine different information system goals. The enterprise architecture analysis method could support decision making within enterprise architectures by providing quantitative comparisons between different enterprise information system scenarios. The software supports all steps of this method.

In the following chapters, the source code of the developed tool is explained. The usage of the Enterprise Architecture Tool is depicted in an example. Finally, additional ideas of further extensions are specified.

8 Implementation

In the following the details of the implementation of the abstract and the concrete modeler are explained. Then, the structure of packages and classes are described. In addition, the design of the user interface is presented. The construction of a Bayesian network and its calculation is described systematically. Finally, challenges and solutions that were discovered during the implementation process are described to ease future development.

8.1 Package and Class Structure

This chapter illustrates and explains the package and class structure of the implementation. Therefore, it first provides an overview of the package tree and gives additional information for each package. In the second part, each class of the packages (except the `eat.am` and `eat.cm` package, refer to 8.2 Abstract Modeler and 8.3 Concrete Modeler) are explained by stating their purpose, important methods, and usages. Due to non-unique naming, all package names in this section are referred to by their fully qualified name in dot notation, e.g. `eat.edit.attribute` instead of package `attribute`. Thus, confusion can be avoided.

8.1.1 Package Structure

Fig. 35 gives an overview of EAT's package structure. In general, this structure can be divided into four parts that refer to the Model-View-Controller (MVC) approach as described in 7.4.

This can be illustrated by having a look at the `eat.am` package. This package includes necessary classes that make up the abstract modeler. It contains classes that are only used in the abstract modeler. One can see that there are four subpackages namely `eat.am.model`, `eat.am.tool`, `eat.am.ui`, and `eat.am.widget`. The `eat.am.model` package contains the Castor generated classes that provide the data model. Thus, this package represents the "Model" in the Model-View-Controler approach. In the `eat.am.tool` package, one can find the classes that control the behavior of the attributes and relationships. Tool classes represent the mediator between model and view classes. Hence, this package typifies the "Controller" of the MVC architecture. Consequently and apparent the `eat.am.ui` package is responsible for user interface functionality – the MVC "View". The `eat.am.widget` package holds a special position as its classes forge the link between the "Model" and the "Controller" by embodying the data model elements.

The same subpackage structure can be found in the `eat.cm` package because the architecture of the concrete modeler is identical to the abstract modeler's architecture.

Three of the remaining packages, `eat.tool`, `eat.ui`, and `eat.widget`, serve the same purpose as the namely similar ones in the `eat.am` and `eat.cm` package. The only difference is that all classes contained in these packages can be used in both EA tools, the AM and CM.

`eat.cpm`, `eat.edit`, `eat.filehandling` are made up of classes that only have supplemental characteristics. They assist tool classes by providing their functionality or simply provide fundamental computation routines.

More detailed descriptions of the purpose and contents of each package are provided on the following pages.

Fig. 35. The package tree of the EAT with brief descriptions

`eat`

The top-level package `eat` contains all subpackages.

`eat.am`

The classes in this package make up the abstract modeler. It consists of subpackages that provide model, view, and controller classes that are exclusively used in the abstract modeler. More information about included subpackages and classes can be found in 8.2.

`eat.cm`

Here one can find the subpackages that define the appearance and fundamental behavior of the concrete modeler. For a more detailed description about all classes and packages included in `eat.cm`, refer to 8.3.

`eat.cpm`

This package is one of the supplemental packages and provides basic and extended classes for enabling dynamic creation of conditional probability tables (CPT). It offers multiple types of CPTs. Moreover, it contains probability calculation variations, e.g. "Max" and "Min" functions.

This package contains the following classes:

- `CPT`
- `Max`
- `Min`
- `Frequency`
- `Parametric`
- `Weighted`

`eat.edit`

`eat.edit` is another supplemental package. Here Undo/Redo management classes can be found. Classes providing this functionality are referred to as "Edit"-classes. This package comprises two subpackages, `eat.edit.attribute` and `eat.edit.entity`, and two classes:

- `EATEdit`
- `RenameEdit`

`eat.edit.attribute`

This subpackage of `eat.edit` is made up of four classes that primarily extend the `EATEdit.java` class for providing undoable and redoable actions for attributes. This includes adding, editing, and removing attributes.

This package contains the following classes:

- `AddAttributeEdit`
- `AttributeEdit`
- `EditAttributeEdit`
- `RemoveAttributeEdit`

`eat.edit.entity`

According to the `eat.edit.attribute` subpackage, here one can find classes that facilitate undoable and redoable actions for entities. Currently this can be add, remove, and multi-remove actions.

This package contains the following classes:

- AddEntityEdit
- EntityEdit
- RemoveEntityEdit
- RemoveMultiEntityEdit

eat.filehandling

This is the third supplemental package and simply provides a class for checking and filtering XML files in a FileChooser dialog.

This package contains following class:

- XmlFileFilter

eat.tool

This is the enclosing package for several subpackages that make up a large part of the MVC-"Controller". All classes in the subpackages can be used with both Abstract modeler and concrete modeler. They are all referred to as "tools" and implemented as Singletons. Each tool provides one relevant use case, e.g. create a new AttributeWidget inside an EntityWidget. For further information on tools, please refer to section 7.4.3.4. Since this package consists of various subpackages, they are all listed below:

- attribute
- entity
- popupmenu
- relationship
- scene
- tag_filter

eat.tool.attribute

This is the tool package that contains classes, which provide various attribute handling features. For example, the AttributeHelperTool.java proffers functions for deep-copying and comparing attributes.

This package contains the following classes:

- AddAttributeTool
- AggregationFunctionTool
- AttributeHelperTool
- CustomCPTDialogTool
- EditAttributeDialogTool

`eat.tool.entity`

In this package, entity-relevant classes can be found. The tools of this subpackage cover capabilities reaching from adding `EntityWidgets` and removing `EntityWidgets` to managing connections between `EntityWidgets`. Non-tool classes providing outer appearance of `EntityWidgets` are contained in this package, too.

This package contains the following classes:

- `AddEntityTool`
- `BorderTool`
- `EATBorder`
- `EATBorderSelected`
- `EATColorScheme`
- `EntityConnectTool`
- `RemoveEntityTool`

`eat.tool.popupmenu`

Each class in this package implements `PopupMenuProvider` and `ActionListener` for the createPopupMenuAction assigned to the according object. For each relevant modeling element, a specific popup menu is implemented. The createPopupMenuAction is triggered by right clicking on a widget where the action was assigned. Currently a popup menu is available for EntityWidget, AttributeWidget, EntityRelationshipWidget, ExternalAttributeRelationshipWidget, InternalAttributeRelationshipWidget, and the scene.

This package contains the following classes:

- `AttributePopupMenuTool`
- `CreateTagSubmenuTool`
- `EntityPopupMenuTool`
- `EntityRelationPopupMenuTool`
- `ExternalAttributeRelationPopupMenuTool`
- `InternalAttributeRelationPopupMenuTool`
- `ScenePopupMenuTool`

`eat.tool.relationship`

The tool-classes inside `eat.tool.relationship` provide capabilities for actions on relationships that are not restricted to either abstract modeler or concrete modeler. The tools enable general use cases like removing relationships or renaming the labels of source role and target role. Moreover, the `AttributeConnectionPathFinderTool` contains path-finding algorithms and neighbor-determination, which is necessary for drawing indirect relationships between attributes.

This package contains the following classes:

- AttributeConnectionPathFinderTool
- RemoveAttributeRelationTool
- RemoveEntityRelationTool
- RenameSourceOfConnectionTool
- RenameTargetOfConnectionTool

eat.tool.scene

This package contains tools whose purpose is not dedicated to modeling elements, but to the scene and the form. All provided actions take part on the scene directly (MultiMoveTool), manipulate the scene (NewSceneTool), or need information from the scene (ExportToPNGTool).

This package contains the following classes:

- CreateModelsTool
- ExpandOnSelectTool
- ExportToPNGTool
- FileHandlingTool
- KeyActionsTool
- MultiMoveTool
- NewSceneTool
- OpenAbstractModelTool
- OpenSceneTool
- RenameWidgetTool
- SaveSceneTool
- UndoManagerTool

eat.tool.tag_filter

Classes that enable tagging and filtering can be found here.

This package contains the following classes:

- AddTagDialogTool
- DeleteTagTool
- FilterDialogTool
- FilterTool
- GetAllTagsTool
- GetTagTool
- UnFilterTool

eat.ui

In eat.ui, there are interfaces and abstract classes serving as a unified foundation for the "View" part of the MVC architecture. Moreover, this package contains dialogs that are used in both modelers. It also provides fundamental table handling classes.

There is one subpackage, called resources, inside eat.ui, which contains optional description files of dialogs.

This package contains the following classes:

- ColumnGroup
- EATAddTagDialog
- EATGraphPinScene
- EATView
- GroupableTableHeader
- GroupableTableHeaderUI
- TypedTableModel

eat.widget

Inside this package there are generalized and parameterized classes providing the basics for any kind of widget that appears in the abstract modeler or the concrete modeler. For example, each widget implements the EATElementResolver in order to allow access to the embraced, particular data model element. Here it makes no difference whether the widget represents a relationship or not.

This package contains the following classes:

- AttributeWidget
- ConnectionConnectionWidget
- EATConnectionWidget
- EATElementResolver
- EATWidget
- EntityRelationshipWidget
- EntityWidget
- ExternalAttributeRelationshipWidget
- InternalAttributeRelationshipWidget

8.1.2 Class Structure

eat.cpm.CPT

The class eat.cpm.CPT is an abstract Java class. Its idea is to represent the different types of dynamic CPTs that were used in this implementation. It provides functionalities to calculate matrices and to convert them into Castor complying CPMs.

This class is used if during the network creation process dynamic CPMs are needed. Its specializations differ in the way they create these CPMs.

`eat.cpm.Frequency`
This class is a specialization of `eat.cpm.CPT`. It creates a frequency based CPM

`eat.cpm.Max`
This class is a second specialization of `eat.cpm.CPT`. It filters the maximum value in the parents' states and gives this state in the child node probability of 1.0.

`eat.cpm.Min`
This class is another specialization of `eat.cpm.CPT`. The created CPM is the opposite of the one created by `eat.cpm.Max`. The minimum value in the parents' states is searched and the state of the child node is set to 1.0.

`eat.cpm.ParametricCPT`
This class is also specialization of `eat.cpm.CPT`. Now it has to be considered as a mock-up of a parametric CPM creator. Therefore, parameters should be used to create a CPM based on them.

`eat.cpm.WeightedCPT`
The class `eat.cpm.CPT` is specialized by this class as well. As `eat.cpm.ParametricCPT` this is also a mock up. When it is implemented in the future, dynamic CPMs, based on weights could be created.

`eat.edit.EATEdit`
This class is used as a foundation for all other edit classes used in the Undo-Redo System of EAT. The default `AbstractUndoableEdit` is extended to make it possible to uses an `UndoManager` that handles all necessary queuing of Edits. The `UndoManager` is handled in the `UndoManagerTool`.
The class provides uniform access to necessary values such as an entity, an attribute or the Scene. In the three member variables a reference of the edited object is saved to make it possible to find out if an Undo or Redo request has to be fulfilled.

`eat.edit.RenameEdit`
The `RenameEdit` class provides functionality to undo and to redo renaming actions of Widgets. Because of the unification of the Widgets, (see 7.4.4.1) it is possible to use this edit class for all renaming actions on all widget that are subclasses of `EATWidget`.

`eat.edit.attribute.AddAttributeEdit`
This class is a subclass of `AttributeEdit` and handles undo and redo requests for the addition of attributes. Undo is implemented using the `RemoveAttributeTool` and redo is implemented using the `AddAttributeTool`.

`eat.edit.attribute.AttributeEdit`

This class provides all general attribute related undo- and redo-features and serves as superclass for all other `AttributeEdit` classes, such as `AddAttributeEdit`, `EditAttributeEdit` and `RemoveAttributeEdit`. The reference to the original attribute, entity and the Scene is saved in this class.

The `AttributeEdit` class also provides a `findAttributeWidget` method that enables one to find the matching `AttributeWidget` based on the entity. This method is necessary because by undoing an entity addition the reference to the corresponding `EntityWidget` is lost. To redo the possible following attribute addition it is necessary to identify the new Widget.

`eat.edit.attribute.EditAttributeEdit`

This class is a subclass of `AttributeEdit` and handles undo- and redo-requests for editing attributes. On construction, a reference to a deep copy of the original attribute and a reference to the new attribute are saved.

On undo, the original attribute is restored by using the `update` method of the `AttributeWidget`. The `AttributeWidget` is found using the `findAttributeWidget` method implemented in the `AttributeEdit` class.

In case a previous undo-action has to be made undone by the redo action, the new attribute is restored via the `update` method of the `AttributeWidget`.

`eat.edit.attribute.RemoveAttributeEdit`

This class is a subclass of `AttributeEdit` and handles undo- and redo-requests for the removal of an attribute. In analogy to the `AddAttributeEdit`, an undo-request is fulfilled by using the `AddAttributeTool` and a redo-request by using the `RemoveAttributeTool`.

`eat.edit.entity.AddEntityEdit`

As a subclass of `EntityEdit` this class provides the functionality for handling undo- and redo-requests of the addition of an entity.

In the event of an undo-action, first, the current position of the entity is saved and in a second step, the entity is removed using the `removeNode` method of the Scene. Redo works vice-versa. First, the entity is added to the Scene using the `addNode` method and second the previous position is restored using the `setPreferredLocation` method provided by the Widget.

`eat.edit.entity.EntityEdit`

This class provides all general entity related undo- and redo-features and serves as superclass for all other `EntityEdit` classes such as `AddEntityEdit`, `RemoveEntityEdit` and `RemoveMultyEntityEdit`. The reference to the original entity, its position and the Scene is saved in this class.

`eat.edit.entity.RemoveEntityEdit`

As a subclass of `EntityEdit`, this class provides the functionality for handling undo- and redo-requests of the removal of an entity.

In the event of an undo-action, first, the entity is added to the Scene again using the `addNode` method and in the second step, the previous position is restored using the `setPreferredLocation` method provided by the Widget. The entity is removed using the `removeNode` method of the Scene.

`eat.edit.entity.RemoveMultiEntityEdit`

`RemoveMultiEintityEdit` is also a subclass of `EntityEdit` and deals with the undo and redo requests regarding the removal of multiple entities in one step. Since it is possible to select multiple entities by dragging a rectangle around them in the modeler, and it is possible to remove them afterwards a handling for undo and redo of this case is needed.
The `RemoveMultiEntityEdit` utilizes the existing `RemoveEntityEdit` by storing one `RemoveEnditEdit` Object per removed entity in its own `UndoManager`. Therefore, in the end `RemoveMultiEntityEdit` is mainly a wrapper for multiple single entity removals.

`eat.filehandling.XmlFileFilter`

This filter examines a certain file. It checks if its file type (the last three letters of a filename) are "XML". This filter functionality is used in open- and save-dialogs, to show only valid documents. In addition, the assignment of a filename is eased, as "XML" can be attached automatically.

`eat.tool.attribute.AddAttributeTool`

This tool handles the addition of an attribute to an entity. It provides functionality for the generation of a new attribute. This new attribute is filled with default values to ensure the feasibility of saving the model without changing any values. In addition to this it is also possible to add concrete and abstract attributes, which already exists, to concrete and abstract entities.

`eat.tool.attribute.AggregationFunctionTool`

This tool provides two methods to map an `AggregationFunction` to an Integer value and vice versa. This is used to map indices of `ComboBoxes` in Dialogs to the adequate `AggregationFunction` and the other way around.

`eat.tool.attribute.AttributeHelperTool`

The `AttributeHelperTool` provides two methods that are related to attributes. The first method is the `copy` method, which generates a deep copy of an attribute. The other method is the `isEqual` method that checks if two attributes have the same values.

Since the attribute classes are automatically generated using Castor, it is not convenient to change them manually. Therefore, such an addition to the functionality of an attribute has to be implemented in a Tool.

`eat.tool.attribute.EditAttributeDialogTool`

This Tool handles the Dialog `EATEditAttributeDialog` that provides the ability to edit an attribute. The Tool initializes the dialog and shows it. When the

dialog is closed, the corresponding AttributeWidget is updated to show the new values.

`eat.tool.entity.AddEntityTool`

Every time the `addNode` method calls the `attachNodeWidget` method of the `EATGraphPinScene`, the `AddEntityTool` provides the creation of a new abstract or concrete EntityWidget. Therefore, it expects the entity, the node that has to be encapsulated. It provides two `execute` methods, one for the AbstractEntityWidget and one for the ConcreteEntityWidget. The creation of the EntityWidget includes the calculation of an appropriate point where the widget is placed on the scene. The computation of this so-called *creation point* considers whether the place is already blocked by another widget.

`eat.tool.entity.BorderTool`

This tool is used for the generation of an order for Entity Widgets. It uses the appearance definitions of `EATBoder` and `EATBorderSelected`. In the future, it would be possible to use the Color definition of an `EATColorSceme` to determine the appearance (see 10.3.2).

`eat.tool.entity.EATBorder`

This class defines the colors of an Entity Widget. The visual appearance of a Widget, for example the gradient, is defined in the `paint` method.

`eat.tool.entity.EATBorderSelected`

The appearance of a selected entity is defined in this class. It is used to differentiate a selected entity from an unselected entity by different visualizations. In the current Version, the border of a selected entity is painted darker than the border of an unselected one.

`eat.tool.entity.EATColorScheme`

The `EATColorScheme` class holds information about a combination of different colors, which can be used for the design of entity, attribute and Relationship Widgets. A set of multiple balanced colors is necessary for the gradient in Entity Widgets.

This feature is not implemented completely. Only the foundation is included in the current Version (see 10.3.2).

`eat.tool.entity.EntityConnectTool`

Like the `AbstractAttributeConnectTool`, this tool is also used for the manual drawing of relationships via “drag & drop”. This class provides the functionality of drawing a connection between two EntityWidgets. This class is assigned as a `createExtendedConnectAction` of an EntityWidget in `EntityWidget`. It separates between abstract and concrete `EntityRelationships`.

In order to draw a relationship the `isSourceWidget` and `isTargetWidget` methods try to resolve and verify the correct source and target Widget of the drag and drop action. The `createConnection` method gathers all relevant data, creates the

abstract or concrete EntityRelationship, and initiates the registration and drawing of an abstract or concrete EntityRelationshipWidget.

`eat.tool.entity.RemoveEntityTool`

This tool handles the removal of an entity and its corresponding Widget from the Scene. It is takes care of all in and outgoing relations before the `removeNode` of the Scene is invoked. The rest of the removal action takes part in the `removeNode` method.

The tool is capable of handling abstract entities as well as concrete entities.

`eat.tool.popupmenu.AttributePopupMenuTool`

This popup menu implements the `PopupMenuProvider` for the `createPopupMenuAction` in the `AttributeWidget` class. After right clicking on an `AttributeWidget`, an appropriate menu pops up. This class defines the appearance of the menu, its menu entries, and the according action for each menu item (`actionPerformed` method). Capabilities like “add tag”, “delete attribute”, or “bring to front” can be accessed via this menu.

`eat.tool.popupmenu.CreateTagSubmenuTool`

This tool creates the menu to show tags of a certain Widget in a list representation. In case the Widget is an attribute representation, tags of the underlying Entity and attribute itself are displayed. This tool is called by the `eat.tool.popupmenu.EntityPopupMenuTool`, `eat.tool.popupmenu.EntityRelationPopupMenuTool`, `eat.tool.popupmenu.ExternalAttributeRelationPopupMenuTool`, `eat.tool.popupmenu.InternalAttributeRelationPopupMenuTool`, and `eat.tool.popupmenu.AttributePopupMenuTool`.

`eat.tool.popupmenu.EntityPopupMenuTool`

This popup menu implements the `PopupMenuProvider` for the `createPopupMenuAction` in the `EntityWidget` class. After right clicking on an EntityWidget, an appropriate menu pops up. This class defines the appearance of the menu, its menu entries, and the according action for each menu item (`actionPerformed` method). Capabilities like “add tag”, “delete entity”, or “bring to front” can be accessed via this menu.

`eat.tool.popupmenu.EntityRelationPopupMenuTool`

This popup menu implements the `PopupMenuProvider` for the `createPopupMenuAction` in the `AbstractEntityRelationshipWidget` and `ConcreteEntityRelationshipWidget` class. After right clicking on an EntityRelationshipWidget, an appropriate menu pops up. This class defines the appearance of the menu, its menu entries, and the according action for each menu item (`actionPerformed` method). Capabilities like “add tag” or “delete relationship” can be accessed via this menu.

`eat.tool.popupmenu.ExternalAttributeRelationPopupMenuTool`

This popup menu implements the `PopupMenuProvider` for the `createPopupMenuAction` in the `AbstractExternalAttributeRelationshipWidget` and `ConcreteExternalAttributeRelationshipWidget` class. After right clicking on an ExternalAttributeRelationshipWidget, an appropriate menu pops up. This class defines the appearance of the menu, its menu entries, and the according action for each menu item (`actionPerformed` method). Capabilities like "add tag" or "delete relationship" can be accessed via this menu.

`eat.tool.popupmenu.InternalAttributeRelationPopupMenuTool`

This popup menu implements the `PopupMenuProvider` for the `createPopupMenuAction` in the `AbstractInternalAttributeRelationshipWidget` and `ConcreteInternalAttributeRelationshipWidget` class. After right clicking on an InternalAttributeRelationshipWidget, an appropriate menu pops up. This class defines the appearance of the menu, its menu entries, and the according action for each menu item (`actionPerformed` method). Capabilities like "add tag" or "delete relationship" can be accessed via this menu.

`eat.tool.popupmenu.ScenePopupMenuTool`

This popup menu implements the `PopupMenuProvider` for the `createPopupMenuAction` in the `EATAbstractGraphPinScene` and `EATConcreteGraphPinScene` class. After right clicking on the scene, an appropriate menu pops up. This class defines the appearance of the menu, its menu entries, and the according action for each menu item (`actionPerformed` method). The functionality of adding a new node, EntityWidget, to the scene can be found in this menu. The new EntityWidget then flies in or just appears at the point of right clicking.

`eat.tool.relationship.AttributeConnectionPathFinderTool`

When an indirect `AbstractExternalAttributeRelationshipWidget` needs to be drawn, this happens with the support of the `EATAttributeConnectionPathDialog`. In that dialog, the user chooses which path to take from the source to the target node. The path, the EntityRelationships between the chosen nodes (EntityWidgets), are then associated to the new `AbstractExternalAttributeRelationship`. In order to display correct and relevant neighbor nodes at each point on the path some algorithms are necessary. This tool provides these algorithms.

There are two methods for both the abstract modeler and the concrete modeler. The first method, `pathExists`, determines whether there is a path from a given source to a given target node. It is a recursive depth-first search and considers already visited nodes (entities) and edges (relationships).

The second method is called `getAppropriateNeighbors` and finds for a given source all neighbor nodes that lie on a possible path between source and target node.

The `execute` methods wrap the according (abstract/concrete) `getAppropriateNeighbors` method and make it accessible.

This tool is also used when it has to be determined if there is a (indirect) path between two nodes, e.g. in `AbstractAttributeConnectTool` and `CheckIndirectAttributeConnectionPathTool`.

`eat.tool.relationship.RemoveAttributeRelationTool`

This class provides the functionality of removing Internal or ExternalAttributeRelationshipWidgets from the scene. Therefore the `execute` method expects the AttributeRelationshipWidget that should be deleted as an argument, decides whether it is internal or external and then removes the AttributeRelationshipWidget with its optional ConnectionConnectionWidgets.

This tool is used in popup menu tools, where it is assigned to menu item actions (e.g. `DELETE_RELATIONSHIP` in `ExternalAttributeRelationPopupMenuTool`). Moreover, it is called when an EntityWidget, an AttributeWidget, or an EntityRelationshipWidget is in deletion process.

`eat.tool.relationship.RemoveEntityRelationTool`

This tool executes the unregistration and removal of an EntityRelationship and its Widget. Therefore the execute method receives the EntityRelationship that should be deleted and calls the `removeEdge` method that calls the scene's detachEdgeWidget method. Moreover, it checks whether there are depending AttributeRelationships and -widgets that should be removed, too.

This tool is used in the `EntityRelationPopupMenuTool` as a performed action for the `DELETE_RELATIONSHIP` menu item. Moreover, it is called when an EntityWidget has been removed whose entity was the source or target node of this EntityRelationship.

`eat.tool.relationship.RenameSourceOfRelationTool`

To be able to edit the text of a LabelWidget at an EntityRelationshipWidget's source node end, this tool is needed. The tool implements the `TextFieldInplaceEditor` for the `createInplaceEditorAction` of the appropriate LabelWidget. Currently it is used for the `sourceName` LabelWidget in the `AbstractEntityRelationshipWidget` class. It gets and sets the text of the LabelWidget and sets the `OriginName` of the EntityRelationship data model element.

`eat.tool.relationship.RenameSourceOfRelationTool`

To be able to edit the text of a LabelWidget at an EntityRelationshipWidget's target node end, this tool is needed. The tool implements the `TextFieldInplaceEditor` for the `createInplaceEditorAction` of the appropriate LabelWidget. Currently it is used for the `targetName` LabelWidget in the `AbstractEntityRelationshipWidget` class. It gets and sets the text of the LabelWidget and sets the `TargetName` of the EntityRelationship data model element.

`eat.tool.scene.CreateModelsTool`

This tool creates Castor conform models, based on the scenes content. At first, an empty abstract or concrete model is created. In the next step all widgets, which are on the scene, are visited and their underlying Castor element are added to this model.

Afterwards these models can be saved (`eat.tool.scene.SaveSceneTool`) or processed for the calculation (`eat.cm.tool.scene.CollectValuesTool`).

`eat.tool.scene.ExpandOnSelectTool`

This class implements the `SelectProvider` necessary for the `createSelectAction` that is assigned to the instances of the AbstractMultiplicityWidget and RelationAggregationFunctionWidget in the `AbstractExternalAttributeRelationshipWidget` and `AbstractEntityRelationshipWidget` classes.

This tool decides what happens when these Widgets are clicked. In this case the `select` method calls the `expand` method of the previously verified, expandable Widget.

`eat.tool.scene.ExportToPNGTool`

This tool simply writes the contents of the current viewable area or the whole scene into a PNG-file that was chosen with the help of a `FileChooser` dialog. This functionality can be found in the `exportToPNGAction` or `exportSceneToPNGAction` method in the EATView classes. This feature can be accessed via a menu entry in the "File" menu.

`eat.tool.scene.FileHandlingTool`

This tool extracts the file type of a given file name. As EAT is only designed to work on XML documents, this check is necessary. Whenever it comes to file interactions, a validation of the type has to be performed.

`eat.tool.scene.KeyActionsTool`

This tool provides the ability of handling Key Events on the Scene. It is a subclass of `WidgetAction.Adapter` from the NetBeans Visual Library. The tool overrides the `keyPressed` method. This method gets a `WidgetKeyEvent` and the current Widget as parameters. The keycode can be resolved from the `WidgetKeyEvent` via its `getKeyCode` method.

The tool has to be added to the list of actions in the Scene.

`eat.tool.scene.MultiMoveTool`

The `MultiMoveTool` implements the interface `MoveProvider`. By adding it as `MoveAction` to the `EntityWidget` it is possible to select multiple entities and to move them using drag and drop.

The functionality is implemented in four methods. The first is the `movementStarted` method, which is called in the moment the movement of the selected entities starts. It collects references to all EntityWidgets that are part of the selection. Then the `getOriginalLocation` method is used to determine the position of one anchor Widget from the selected Widgets. The `setNewLocation` method calculates the movement of the anchor Widget and moves all other entities by the same amount. Finally, the `movementFinished` method clears the list of entities.

`eat.tool.scene.NewSceneTool`

This class provides functionality to refresh a scene. All content that has been modeled before is removed. Afterwards the scene is prepared for a new model. Besides the obvious use case to start from scratch, the tool is also used to clean up the scene before an existing model gets loaded (`eat.tool.scene.OpenSceneTool`).

`eat.tool.scene.OpenAbstractModelTool`

This tool is an analogical tool to `eat.cm.tool.scene.OpenConcrete-ModelTool`. It loads an abstract model that can be found at a certain file path in a XML file. Unmarshall functionalities provided by Castor were used. They also result in an instantiation of Castor generated classes.

`eat.tool.scene.OpenSceneTool`

This tool is used to load XML documents, in which previously created models are contained. The `OpenSceneTool` can be used inside the abstract and concrete modeler. This is the reason why the tool has to determine whether the current model is abstract or concrete. After this decision, either `eat.tool.scene.Open-AbstractModelTool` or `eat.cm.tool.scene.OpenConcretemodel-Tool` is executed.

`eat.tool.scene.RenameWidgetTool`

This tool provides a `TextFieldInplaceEditor` for changing an entity or an attribute name. The `TextFieldInplaceEditor` is shown at double-click on a Widget that can be renamed. It materializes at exactly the same position as the name has been shown before.

The tool is capable of editing the names of abstract entities, abstract attributes and concrete entities. Due to modeling conventions, it is not possible to edit the name of a concrete attribute.

`eat.tool.scene.SaveSceneTool`

This class contains the functionality to save a scene's content to an XML file. Therefore, the content of the scene is read, with the help of (`eat.tool.scene.CreateModelsTool`), and Castor-compliant models are created. These models are serialized using the Castor provided functionalities into an XML document.

`eat.tool.scene.UndoManagerTool`

This tool provides access to the undo and redo functionality in EAT. It contains the `UndoManager` Object that handles all queuing activities of edits.

By using the `UndoManagerTool`, it is possible to access the functionality from virtually any point of the code.

`eat.tool.tag_filter.AddTagDialogTool`

This tool displays a dialog to add a new tag (`eat.ui.EatAddTagDialog`) for the different elements that could be created. The dialog is prepared in advance, depending on whether an abstract or concrete element should be tagged.

`eat.tool.tag_filter.DeleteTagTool`

This tool displays a dialog (`eat.am.ui.DeleteTagDialog`) to delete one or more tags of an element. The dialog is prepared with the assigned tags of the underlying EntityWidget, AttributeWidget or RelationshipWidget.

`eat.tool.tag_filter.FilterDialog.Tool`

This tool prepares and displays the dialog to configure the filter functionality (`eat.am.ui.EATConfigureFilterDialog`). For easier use the settings, made on last dialog run, are preset. On first execution, this tool sets a flag, so that the dialog shows an initial default configuration. The tag filtering of the dialog is preset as well. Therefore, all tags, assigned to any model elements are read (using `eat.tool.tag_filter.GetAllTagsTool`) and transferred.

`eat.tool.tag_filter.FilterTool`

This tool filters the scene content based on a filter configuration made in the `eat.am.ui.EATConfigureFilterDialog`. The filter process consists of two sections. In the first step, all elements are set to invisible, therefore the `eat.tool.tag_filter.UnfilterTool` is used. Afterwards the content fitting to the filter criteria are displayed.

`eat.tool.tag_filter.GetAllTagsTool`

A list of all assigned tags is created by the `eat.tool.tag_filter.GetAllTagsTool`. This collection is built by iterating over all elements of the scene. In case they were tagged with a label that is not already known, the tag is added to the collection.

`eat.tool.tag_filter.GetTagTool`

This tool is implemented to deliver tags of a certain scene element, whereas the `eat.tool.tag_filter.GetAllTagsTool` collects tags of all elements (this is done by multiple runs of the `eat.tool.tag_filter.GetTagTool`).

`eat.tool.tag_filter.UnfilterTool`

This tool offers functionality to set the whole scene visible or to set it completely non-visible. This functionality is needed during the filtering process. In addition, it is applied when the filter is switched off and the scene is displayed completely again.

`eat.ui.ColumnGroup`

This class is used to organize multiple columns in a table as a group. It is mainly used in association with a `GroupableTableHeader`. It stores information about all columns that belong to a group.
Original source from [28] has been marginally modified.

`eat.ui.EATAddTagDialog`

This dialog offers functionality to add a new tag to an element. Before a tag is assigned, a check is performed whether the element is already tagged with this label.

The eat.tool.tag_filter.AddTagDialogTool prepares this dialog in advance, to anchor the widget that should be tagged.

eat.ui.EATGraphPinScene

From Visual Library 2.0 - Documentation: "GraphPinScene manages a model with nodes, pins and edges. A pin is always attached to a node. Edge could be connected to a source and a target pin only." [It uses] generics and therefore developers can specify their classes that represents nodes, pins and edges. [...]"

This class is an abstract and parameterized generalization class for the AbstractGraphPinScene and the ConcreteGraphPinScene. It provides functionality that is used in both subclasses, like the counter for the generation of IDs and the handling of the different layers on the Scene. It is a subclass of GraphPinScene (cf. 7.2) and is still implemented as a generic class. The instantiation of the generic types is done in the subclasses of EATGraphPinScene.

eat.ui.EATView

This is a generalization interface for EATAbstractView and EATConcreteView. It determines common functionalities like the checkUndoRedo method.

eat.ui.GroupableTableHeader

The GroupableTableHeader provides the ability to visualize which table columns belong together by grouping them under a spanning table header. To achieve this, a subclass of JTableHeader is generated which holds information about the groups of columns, beside the usual needed information about the standard columns.

The column groups are provided by the class ColumnGroup and the visualization is done by the GroupableTableHeaderUI class.
Original source from [28] has been marginally modified.

eat.ui.GroupableTableHeaderUI

This provides visualization Information and methods for a GroupableTableHeader. All needed paint methods are defined in this class.
Original source from [28] has been marginally modified.

eat.ui.TypedTableModel

This class is a derivation of the TableModel used in JTables. It was necessary to implement an own TableModel because the DefaultTableModel is not capable of handling different data types for different table columns. This was necessary in the Edit-Attribute-Dialogs where the table should contain the name of a state as string and the value as double.
Main extensions are an array of classes, which contains the corresponding data type of a column, and overriding the getColumnClass method, which provides access to the type array.

`eat.widget.AttributeWidget`

This is a generalization class for the `AbstractAttributeWidget` and the `ConcretAttributeWidget`. More information about the class model of the Widgets can be found in chapter 7.4.4.1.

`eat.widget.ConnectionConnectionWidget`

This extension of a `FreeConnectionWidget` is a special kind of Widget whose main purpose is to illustrate the dependency between EntityRelationships and AttributeRelationships. It is displayed as simple lines between an AttributeRelationshipWidget and all of its associated EntityRelationshipWidgets. This tool defines the appearance, sets the source and target of the line, and assigns actions to the Widget. Finally, it adds the new Widget to the `connectionLayer` and updates the scene.

`eat.widget.EATConnectionWidget`

This Widget class serves as the foundation for all EntityRelationshipWidgets and AttributeRelationshipWidgets. It is a generalization class of Visual Library's ConnectionWidget. Its purpose is to provide common attributes and methods for managing the data model element.

`eat.widget.EATElementResolver`

This interface is part of the Widget inheritance hierarchy. It provides the `getElement` method. See 7.4.4.1 for more information.

`eat.widget.EATWidget`

This is the entry point of the Widget hierarchy. It marks the connection between the NetBeans Visual Library and the self-designed parts. See 7.4.4.1 for more information.

`eat.widget.EntityRelationshipWidget`

This class is used for generalization of the `AbstractEntityRelationshipWidget` and the `ConcreteEntityRelationshipWidget`. See 7.4.4.1 for more information.

`eat.widget.EntityWidget`

This class is used for generalization of the `AbstractEntityWidget` and the `ConcreteWidget`. See 7.4.4.1 for more information

`eat.widget.ExternalAttributeRelationshipWidget`

This class is used for generalization of the `AbstractExternalAttributeRelationshipWidget` and the `ConcreteExternalAttributeRelationshipWidget`. See 7.4.4.1 for more information.

`eat.widget.InternalAttributeRelationshipWidget`

This class is used for generalization of the `AbstractInternalAttributeRelationshipWidget` and the `ConcreteInternalAttributeRelationshipWidget`. See 7.4.4.1 for more information.

8.2 Abstract Modeler

The abstract modeler provides the ability to build abstract models as described in 7.3.3.1. It is possible to add, edit and remove entities and attributes of entities, as well as the definition of relationships between them.

The ability to save and load models has also been implemented.

8.2.1 Package Structure

This chapter deals with all subpackages of the `eat.am.*` package. As described in 8.1 and in 7.4 EAT incorporates the Model-View-Controller approach.

The package `eat.am` contains classes that are only used in the abstract modeler and not in the concrete modeler. In several cases the abstract modeler uses also classes of the more general `eat.*` package. These are described in 8.1.1.

All packages of the abstract modeler can be matched to the MVC approach. The `eat.am.model.*` package contains all classes needed for the model or data part. For visualization, mostly the classes of the `eat.am.ui.*` package and in some parts the classes of the `eat.am.widget` package are used. Finally, the controller part is mainly represented in the classes of the package `eat.am.tools` and in part by the classes of the `eat.am.widget` package.

`eat.am.model.*`

All classes, which can be found in the eat.am.model and its subpackage, are automatically generated by Castor (7.1.3). All information about an abstract model is stored by using these classes.

The `eat.am.model.decriptors` package contains additional information of the model elements that are also generated by Castor.

`eat.am.tool`

The package `eat.am.tool` contains the tools that are only used in the implementation of the abstract modeler. It contains two subpackages `eat.am.tool.attribute` and `eat.am.tool.relationship`.

`eat.am.tool.attribute`

In the package `eat.am.tool.attribute`, two tools are stored which deal with attributes in abstract models.

This package contains the following classes:

- `AbstractAttributeConnectTool`
- `RemoveAttributeTool`

`eat.am.tool.relationship`

Tools that are related to relationships in abstract models can be found in the package `eat.am.tool.relationship`. Here the contained classes call dialogs that are relevant for relationship handling.

This package contains the following classes:

- `ChooseRelationshipForIntAttRelDialogTool`
- `ConnectionPathDialogTool`

`eat.am.ui`

The package `eat.am.ui` contains all classes needed for displaying the user interface of the abstract modeler with all its components and dialogs. Key-classes are the `EATAbstractView` as the main perspective of the modeler and the `EATAbstractGraphPinScene` as part of the NetBeans Visual Library (cf. 7.2), which provides the modeling surface.

Another very important component of the EA Tool can be found in this package. The `EATApp` class contains the `main`-method of the project and serves as entry point on program launch.

The subpackage `eat.am.ui.resources` contains additional property-files for information like button labels or menu entries. These files are automatically generated while using the NetBeans User Interface Designer (cf. 7.1.5). In addition, bitmap images for the toolbar buttons are stored in this package.

This package contains the following classes:

- `EATAboutBox`
- `EATAbstractGraphPinScene`
- `EATAbstractView`
- `EATApp`
- `EATAttributeConnectionPathDialog`
- `EATChooseRelationshipForIntAttRelDialog`
- `EATConfigureFilterDialog`
- `EATCustomCPTDialog`
- `EATDeleteTagDialog`
- `EATEditAttributeDialog`

`eat.am.widget`

Widget classes used to visualize model information on the Scene are stored in the package `eat.am.widget`. More information on the role of Widgets can be found in 7.2.2.

This package contains the following classes:

- `AbstractAttributeWidget`
- `AbstractEntityRelationshipWidget`
- `AbstractEntityWidget`
- `AbstractExternalAttributeRelationshipWidget`
- `AbstractInternalAttributeRelationshipWidget`
- `AbstractMultiplicityWidget`
- `RelationAggregationFunctionWidget`

8.2.2 Class Structure

`eat.am.tool.attribute.AbstractAttributeConnectTool`

This tool provides the functionality of drawing a connection between two abstract AttributeWidgets. In order to register and draw a relationship, first the source and target Widgets have to be resolved. If none of them is null, it is checked whether the connection is going to be internal or external, i.e. whether the source and target Widgets are the same or not. Related EntityRelationshipWidgets are resolved, too. By now, it is also possible to draw attribute connections between AttributeWidgets that are indirectly connected. This class is assigned as an `ExtendedConnectAction` of an AttributeWidget in AbstractAttributeWidget.java. The methods in this tool are called automatically by the action.

`eat.am.tool.attribute.RemoveAttributeTool`

This tool provides the necessary functionality for removing attributes from entities. In the first step, the entity containing the attribute is resolved from the model structure. After removing the attribute from the according entity, it is also removed from the Entity Widget and by this from the scene. In the last step, it has to be taken care of the attribute relationships that are connected to the deleted attribute. In this step, the incoming and outgoing attribute relations have to be considered as well as internal and external ones, as they are represented by different Object-Types.

`eat.am.tool.relationship.ChooseRelationshipForIntAttRel-DialogTool`

This tool calls a dialog that lets the user choose whether a recently drawn AbstractInternalAttributeRelationshipWidget should be associated to an according AbstractEntityRelationshipWidget. Therefore, the tool computes all AbstractEntityRelationshipWidgets that have the same source and target EntityWidget as the latterly drawn AbstractInternalAttributeRelationshipWidget. The result list is then passed to the tool that displays the results in a ComboBox to the user. The chosen AbstractEntityRelationshipWidget, that can be “NONE” also, is then set as the result and passed back to the tool.

`eat.am.tool.relationship.ConnectionPathDialogTool`

Each time a user wants to draw an AbstractAttributeRelationshipWidget between AbstractAttributeWidgets whose parent AbstractEntityWidgets are not directly connected, this tool is called (by the `AbstractAttributeConnectTool`). It subsequently calls the `EATAttributeConnectionPathDialog` that lets the user choose which path along AbstractEntityRelationshipWidget to take until the target is reached. More information about this procedure can be found in the `EATAttributeConnectionPathDialog` class description below. The result of the dialog (EntityRelationshipWidgets on the selected path) that was passed back to this tool is subsequently passed to the AbstractAttributeConnectTool. Here the AbstractEntityRelationshipWidgets on the selected path are essential information for the creation of an AbstractAttributeRelationship(Widget). The AbstractEntityRelationshipWidgets become associated with the newly created AbstractAttributeRelationship, visualized in the AM by a ConnectionConnectionWidgets between both types of relationships.

`eat.am.ui.EATAbstractGraphPinScene`

From Visual Library 2.0 - Documentation: "GraphPinScene manages a model with nodes, pins and edges. A pin is always attached to a node. Edge could be connected to a source and a target pin only." [It uses] generics and therefore developers can specify their classes that represents nodes, pins and edges. [...]" (note: maybe move to EATGraphPinScene)
EATAbstractGraphPinScene provides all methods for managing the scene in the abstract modeler

`eat.am.ui.EATAbstractView`

This class represents the main Application Frame of the abstract modeler. It contains information about all UI elements such as Buttons and Menus. In addition, various Actions for different Interaction Events as Menu selection or Mouse Clicks are handled in this class.

`eat.am.ui.EATApp`

This class is the main class of the Application. It contains the main-Method that initializes all UI and Application elements and serves as entry point on startup.
The differentiation between the concrete modeler and the abstract modeler is also handled in this Class. It is implemented as a command-line-parameter, which has to be given on startup. If the application is launched without parameter, the abstract modeler starts. If the concrete modeler is the desired Application Part, one has to specify the command line parameter "`-concrete`".

`eat.am.ui.EATAttributeConnectionPathDialog`

This class provides a dialog in the abstract modeler that appears when the user draws an indirect AbstractAttributeRelationshipWidget. Here a user can choose which path to take from the source EntityWidget to the target EntityWidget via various EntityRelationshipWidgets. This dialog is called by the `ConnectionPathDialogTool`. At first, the `setup` method prepares the dialog with the first choices. Therefore, it depends on the path calculation results of the `AttributeConnectionPathFinderTool`. This tool computes all possible neighbors of the source Enti-

tyWidget that are located on a path from source to target. These relevant neighbors are then displayed in the ComboBox. After a user selection, the `prepareNext-Choice` method is called which prepares the next step. This is repeated until the user selects a node that represents the target EntityWidget. All EntityRelationshipWidgets that lie on the selected path between source and target are stored in a Vector and passed back to the `ConnectionPathDialogTool`.

`eat.am.ui.EATChooseRelationshipForIntAttRelDialog`

This dialog is called by the `ChooseRelationshipForIntAttRelDialog-Tool`. This dialog displays all relevant AbstractEntityRelationshipWidgets that can be chosen as associated relationships to the recently drawn AbstractInternalAttribute-RelationshipWidgets. Thereby it is also possible to choose "NONE", which means that the AbstractInternalAttributeRelationshipWidget remains without a reference to an AbstractEntityRelationship(Widget). In the case the "Cancel"-button is pressed the whole registration and drawing process will be aborted. The "Apply"-button passes the chosen AbstractEntityRelationshipWidget back to the tool.

`eat.am.ui.EATConfigureFilterDialog`

This Dialog is used to configure the filter in the abstract modeler. A selection of the different Scene-elements is possible. The decision can be made between entities (EntityWidgets), attributes (AttributeWidgets), entity relationships (EntityRelation-shipWidgets), and attribute Relationships (AttributeRelationshipWidgets). The dialog takes care of the model hierarchy, so only useful combinations of elements are allowed. When a certain component should be shown, its underlying also gets selected if it is not already. This means that AttributeWidgets only are displayed, when their EntityWidgets are selected, too. In case of de-selection of an element, its children are deselected as well.

The consideration of tags can be activated too. This means that only scene elements that have one of the relevant tags are displayed. If the use of tags is selected, a choice between the assigned tags gets enabled.

A filter that has been configured within this dialog can be activated in the main view (`eat.am.ui.EATAbstractView`) or changed, when the Dialog is opened again. For this use case, the functionality that is provided by `eat.tool.tag_filter.FilterDialogTool` is necessary.

`eat.am.ui.EditAttributeDialog`

This class provides a Dialog for editing attributes. It is possible to change the attributes name and different characteristics of its CPM such as the dynamic CPM type or the states and their values. The appearance of the displayed (conditional probability) table depends on the model structure.

`eat.am.ui.DeleteTagDialog`

This Dialog shows the assigned tags of a certain taggable element (viz. EntityWidgets, AttributeWidgets, and the different kind of relationship widgets). These tags can be selected. When the Delete button is pushed, the selected tags are removed. In case the Cancel button is selected, nothing happens, except the dialog disappears.

`eat.am.widget.AbstractAttributeWidget`

This class provides the visual container for an attribute element in the abstract modeler. It extends the AttributeWidget class. In the tree-hierarchy of Widgets these AbstractAttributeWidgets are children of a Widget (`compartmentWidget`) which itself is a child of an EntityWidget. An AbstractAttributeWidget provides a field for the data model element (attribute), a LabelWidget for displaying the name, and actions, e.g. for offering a popup menu.

`eat.am.widget.AbstractEntityRelationshipWidget`

This is the abstract specialization of an EntityRelationshipWidget. It is the user interface container for the EntityRelationship data model element in the abstract modeler. It lies in the connection layer of the `EATAbstractGraphPinScene`. `AbstractEntityRelationshipWidget` is made up of two AbstractMultiplicityWidgets, a central name LabelWidget, and two role-name LabelWidgets. Moreover, it holds an action list for providing a popup menu and control points.

`eat.am.widget.AbstractEntityWidget`

The graphical representation of a node on the `EATAbstactGraphPinScene` is provided by an AbstractEntityWidget object. It contains a field for the data model element Entity. Furthermore, it holds a LabelWidget and zero or more AbstractAttributeWidgets. This class extends the `EntityWidget` class.

`eat.am.widget.AbstractExternalAttributeRelationshipWidget`

This widget class is the abstract specialization of a relationship between two AttributWidgets whose parent EntityWidgets are different. As this is a Widget its main functionality is to provide a field for the data model element, which here is an ExternalAttributeRelationship. Moreover, actions for popup menu and control point capabilities are assigned. Because an AttributeRelationship holds information about its associated EntityRelationships, this association needs to be displayed, too. Therefore, an `AbstractExternalAttributeRelationshipWidget` computes the targets of its related EntityRelationshipWidgets (precisely the central LabelWidget of the EntityRelationshipWidget). Then for each associated EntityRelationship it creates a `ConnectionConnectionWidget` between the source (LabelWidget of `AbstractExternalAttributeRelationshipWidget`) and the target. The created ConnectionConnectionWidget is then stored in a Vector.

`eat.am.widget.AbstractInternalAttributeRelationshipWidget`

This widget class is very similar to the `AbstractExternalAttributeRelationshipWidget`. It provides exactly the same functionality, but is used when source and target AbstractAttributeWidgets of the abstract AttributeRelationshipWidget belong to the same parent AbstractEntityWidget. A special characteristic of this relationship widget is that it has at most one related EntityRelationship.

`eat.am.widget.AbstractMultiplicityWidget`

The `AbstractMultiplicityWidget` is located at the source and target EntityWidget of an AbstractEntityRelationshipWidget. Here the user can choose be-

tween four multiplicities - "1", "0..1", "0..*", and "1..*". It consists of a LabelWidget showing the current selection and a detailsWidget, which contains the JRadioButtons that provide the choices. Each button is linked to an ActionListener that processes the selection. An `AbstractMultiplicityWidget` knows whether it is at the source or target end. It is able to expand and collapse itself. In the collapsed state, only the LabelWidget with the current chosen multiplicity is shown. On double-clicking, the widget expands itself and shows the four choices. After selection, it collapses again. The `expand-` and `collapse`-methods control the according behavior. The moment of expand is controlled by the `ExpandOnSelectTool` which implements a `SelectProvider`. This provider is a parameter for the `createSelectAction` that is assigned to the `AbstractMultiplicityWidget` in the `AbstractEntityRelationshipWidget` class.

`eat.am.widget.RelationAggregationFunctionWidget`

This widget works in the same way and provides the same methods as the `AbstractMultiplicityWidget`. The difference is that the `RelationAggregationFunctionWidget` is assigned to an `AbstractExternalAttributeRelationshipWidget`.

8.2.3 User Interface

This paragraph is meant to explain the graphical user interface (GUI) of EAT. Therefore, all menus and dialogs that might appear, during a program run will be considered, depending on the use-case. The illustration below is structured as follows: At first, the main perspective and the abilities offered to the user are explained. This main perspective is the scene on which the models are drawn. From there on the different options for the user are considered, according to their underlying user interface. In this way, all dialogs and options, displayed to the user, are considered. It has to be mentioned that, compared to the usage example (chapter 9), no continuous scenario is considered. The advantage of this proceeding are that a certain dialog is explained on its own and therefore understandable without knowledge of the rest of the model. A consideration of a complete model would also lead to large redundancies and imply that there is only a single way of using, which is not the case. It should be remarked that the GUI description is restricted to the explanation of the dialogs and perspectives presented to the user. Further, it is not intended to show here why a certain value could be found at a certain place or which configuration has advantages over another one.

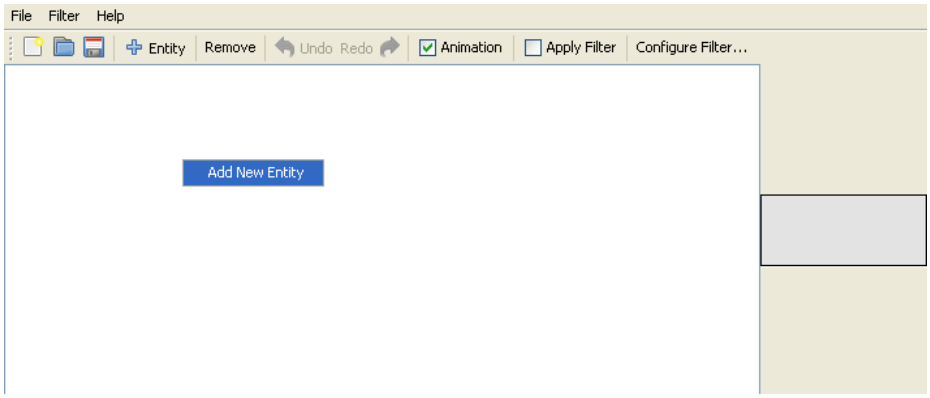


Fig. 36. Empty-Model Perspective of the Abstract Modeler

In *Fig. 36* the initial perspective of the abstract modeler is visualized. This perspective is displayed each time the drawing of a new model is started. The largest share of this perspective is the (white) space on which the models are drawn. A left-click on this space displays a popup-menu that offers to add a new entity to a model (the corresponding menu-entry is labeled: "Add New Entity").

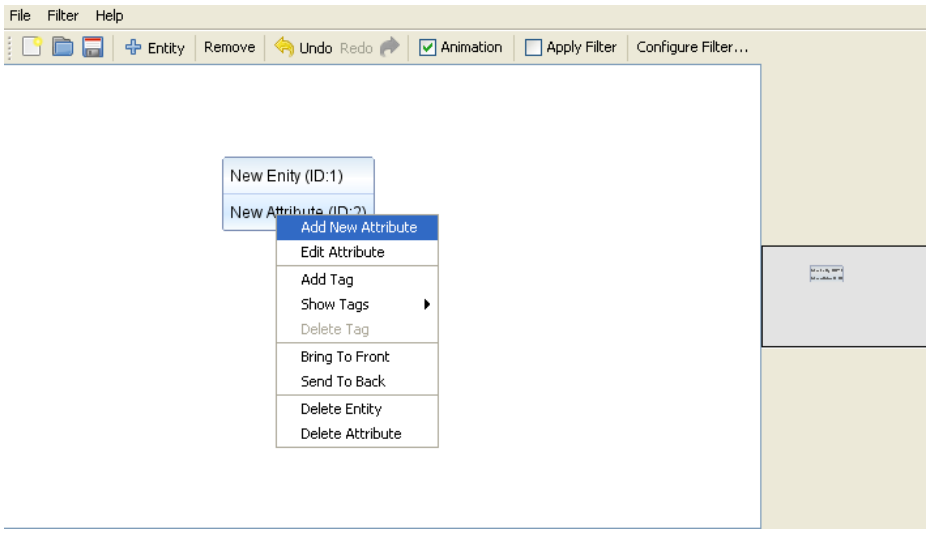


Fig. 37. Popup-menu of an Abstract Entity

The heading of this model-perspective is separated into two parts: a textual-menu part and a bar, which contains graphical and text-based icons. The first section of the textual part is the "File"-menu. It provides the functionality to start with a clean, new

model (shortcut is CTRL+N), to open an existing model (shortcut is CTRL+O), to save a model, with use of the well-established “Save”- and “Save As”-functionalities (shortcuts are CTRL+S and CTRL+Shift+S). The possibilities to create a screenshot of the model or the section which is currently displayed, is also offered by the “File”-menu. Its last menu-entry is the exit-functionality, which also can be accessed with the shortcut CTRL+Q. The “Filter”-menu offers two functionalities: configuration and application of a filter. By now, the “Help” menu provides only information about the vendor of the EA tool.

The buttons of the graphical part of the heading are explained starting from left to right (in *Fig. 37*). At first, the opportunity to reset the model is offered again. The next button offers functionality to open a previously modeled scenario. “Saving functionality”- is provided by the third button. The next button causes an addition of a new entity to the scene, whereas its neighbor to the right removes a selected scene element. The sixth button provides “undo”-functionalities. This means that a modification of the model is reverted. Button number seven provides “redo”-functionalities, causing the reversion of a previous executed “undo”-functionality. The switch labeled “Animation” switches on or off whether the entities, added by the popup-menu “fly” into the scene or appear straight ahead. In case a defined filter should get applied this can be switched on via the “Apply Filter”-switch. The last button displays configuration options for the filter, which is presented below.

Besides menus and the model-scene, a so-called “Satellite-view” visualizes the complete modeled structure. This view, which virtually films the scene from a satellite, is embedded in the right part of the GUI.

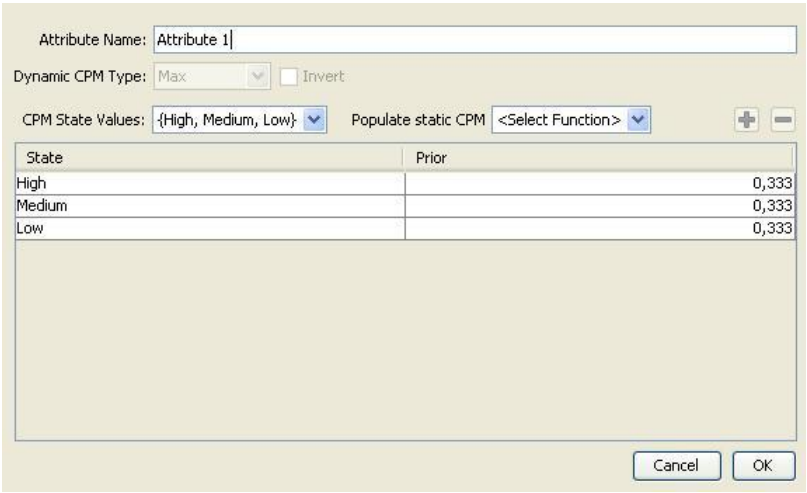
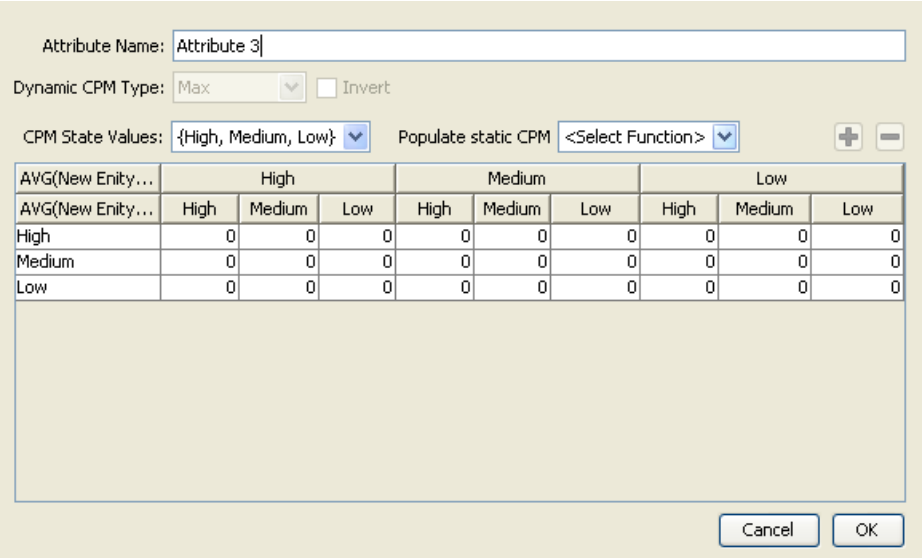


Fig. 38. Dialog of an Attribute, with zero multiplicity

In the illustration presented above, besides the already explained GUI, the popup-menu of an entity is visualized. It offers the possibility to add a new attribute and edit the currently selected one. In addition, several options for tag-handling, namely addition and removal of a tag as well as showing of the attached tags, are offered. The

EntityWidget, symbolizing an Entity, can also be brought to the front and send to the back of the current scene. This helps to reorganize the models, especially when they are very large. At last, the option to delete the current entity is offered.
Depending on the model structure, two different dialogs to edit an attribute are shown. This depends on whether the multiplicities allow zero connections or not.

In *Fig. 38* the dialog, which provides the edit-functionality of an attribute, is presented. First, the name of the attribute can be changed. Below that, the dynamic CPM of the attribute can be chosen out of a predefined selection. In case the attribute is a Prior, the CPM States can be selected again from a predefined collection. The table at the bottom of the dialog allows mapping numerical values to the respective states. In addition, the possibility to select a static CPM, which should be populated, is offered. A configuration is saved, when the "OK"-button is pushed, whereas the execution of the "Cancel"-button discards the changes which were made.



AVG(New Enity…	High			Medium			Low		
AVG(New Enity…	High	Medium	Low	High	Medium	Low	High	Medium	Low
High	0	0	0	0	0	0	0	0	0
Medium	0	0	0	0	0	0	0	0	0
Low	0	0	0	0	0	0	0	0	0

Fig. 39. Edit Dialog of an Attribute, without zero multiplicity

In the dialog, which is presented below (*Fig. 39*), the configuration of an attribute, which has no multiplicity that could be zero, is displayed. The dialog is derived from the first attribute-dialog (*Fig. 38)*; however, in this case the CPM can be modified. This CPM depends on the structure of the model and allows the modification of a certain probability by the user.
This consideration of the GUI is continued with an explanation of the dialogs, which are relevant for the different tagging functionalities.

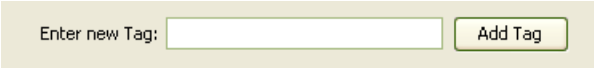


Fig. 40. Add Tag Dialog of a scene element

The dialog displayed above (*Fig. 40*) is shown, when a new Tag should be added. It is a simple input mask, to enter a tag, which is attached to the currently considered element.

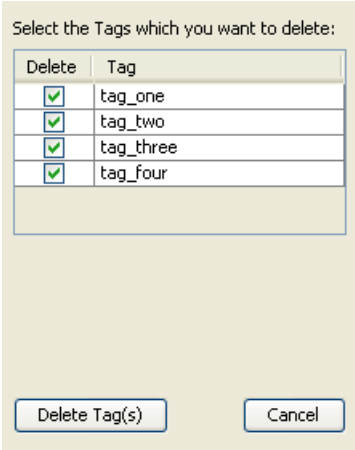


Fig. 41. Delete Tag Dialog of a Scene‘s element

In *Fig. 41*, the dialog, which is displayed when a tag should be removed, is presented. A list of all assigned tags of the selected scene element is shown. The user can select any combination of tags that should be removed.

In *Fig. 42*, the dialog for configuring the filtering of the model is illustrated. The dialog is separated into two parts. At first the elements, which should be considered and therefore displayed on the scene, can be selected. At the bottom of the dialog a selection of tags can be made, depending on the state of the “use Tags”-switch. In case the tags should be used, they are presented in a list and a subset can be configured. When this configuration is applied all scene elements that have at least one tag out of this subset, will be displayed on the canvas.

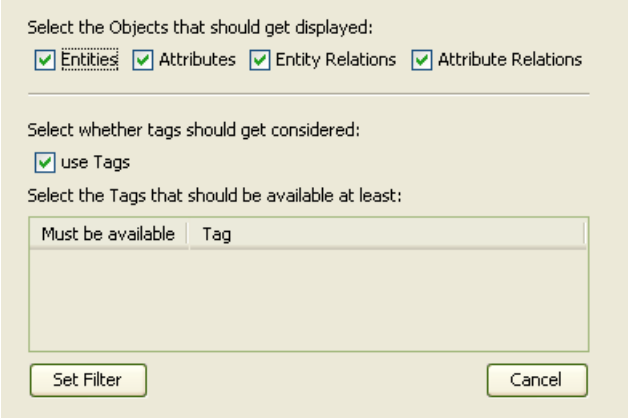


Fig. 42. Configure Filter Dialog

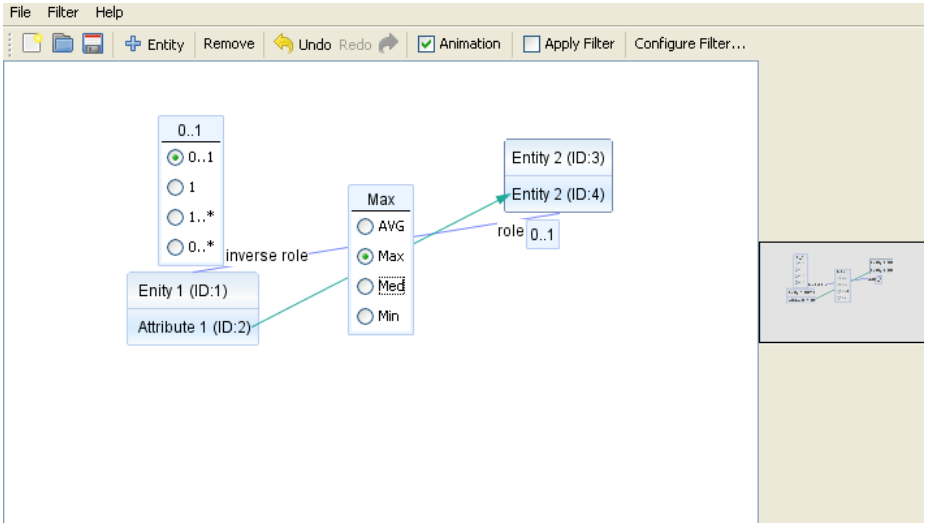


Fig. 43. Configuration of Multiplicity and Aggregation Function

Finally, the dialogs that occur during the creation of relationships are explained. A relationship can be created with the use of "Drag&Drop", while the CTRL-key is pressed. An abstract model, which has been created in this way, is visible in *Fig. 43*. For begin and end of an entity relationship a configuration of the multiplicities is possible, by choosing one out of four possible candidates. In addition, the roles can be named individually. The Aggregation Functions of the attribute relationships are also allowed to be one out of four different kinds, to be selected by the user.

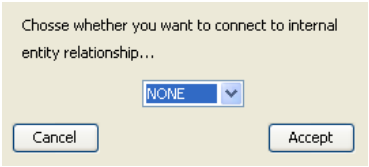


Fig. 44. Configuration of internal Relationship Dialog

The dialog presented in *Fig. 44* is displayed automatically when an internal attribute relationship is created. This dialog allows connecting the internal attribute relationship to an entity relationship, which was introduced earlier into the model. This mapping is optional.

An indirect attribute relationship between a pair of source and target nodes is configured within a separate dialog (*Fig. 45*). In case of more than one valid entity relationships, a path has to be chosen manually. Therefore, on the way from source to target, each currently considered node and its descendant has to be selected. The path that has already been accepted is displayed in a box at the bottom of the dialog.

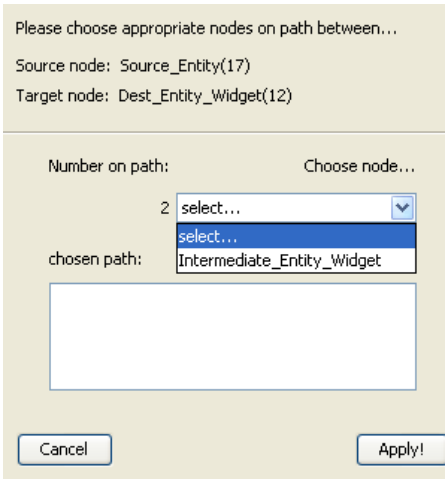


Fig. 45. Configuration of indirect Attribute Relationship

8.3 Concrete Modeler

In the concrete modeler, the abstract models created with the abstract modeler are processed. The model elements, which were defined in the abstract modeler, are instantiated. Therefore, a certain abstract model has to be loaded. Concrete entities, which have their root in abstract entities, can be connected based on the previous defined relationships. To a given attribute of an entity, numerous evidences can be attached. When the modeling process is finished, the calculation (8.3.4) can be executed. Afterwards the calculated values to each state of each attribute are set. A reconsideration of the attributes shows them to the user.

8.3.1 Package Structure

As explained in 8.1 the EAT consists of a multitude of packages. This paragraph takes a closer look on the `eat.cm.*` packages. They have in common that only Java classes with relevance for the concrete modeler are provided here. Four different kinds of packages can be found: `eat.cm.model.*`, `eat.cm.tool.*`, `eat.cm.ui.*`, and `eat.cm.widget`.

`eat.cm.model.*`

The classes that can be found in the `eat.cm.model.*` were automatically generated by Castor (cf. 7.1.3), based on XSD documents (cf. 7.1.2). In these classes, the representation of the model elements can be found. For each node element in the XSD files (Appendix B: XSD) a separate class was introduced. The package `eat.cm.model.descriptors` contains additional information about the model elements "...to dramatically enhance performance" [7].

`eat.cm.tool.*`

The package `eat.cm.tool` organizes the tools for the concrete models. It is again subdivided into three parts. The subpackage `eat.cm.tool.attribute` bundles classes with relevance to the concrete attributes. The functionalities to connect and edit them were implemented inside this package. In addition, a tool to manage the creation of dynamic CPMs is found here (`CreateDynamicCPMTool`).

In `eat.cm.tool.relationship` tools with relation to the connections of concrete attributes or entities are held.

The third subpackage `eat.cm.tool.scene` bundles all tools that offer functionalities for a certain model. On the one hand, tools that are used during the calculation process are found here. On the other hand, a tool to load a concrete model is defined in this package (`OpenConcreteModelTool`).

The subpackage `eat.cm.tool.attribute` contains the following classes:

- `ConcreteAttributeConnectTool`
- `CreateDynamicCPTTool`
- `EditConcreteAttributeDialogTool`
- `UpdateAttributeCPMTool`

The subpackage `eat.cm.tool.relationship` contains the following classes:

- `CheckIndirectAttributeConnectionPathTool`
- `ChooseRelationshipTypeDialogTool`

The subpackage `eat.cm.tool.model` contains the following classes:

- `CalculateTool`
- `CollectValuesTool`
- `DisplayResultsTool`
- `OpenConcreteModelTool`

`eat.cm.ui.*`

Dialogs and components which are relevant for the user interface are found in the `eat.cm.ui.*` package. Especially `EATConcreteGraphPinScene`, the scene of the concrete modeler, and `EATConcreteView`, the default perspective of the modeler, should already be mentioned here. They both build the foundation of the concrete modeler.

This package also has a subpackage `eat.cm.ui.resources`. Inside, multiple property-files can be found. They provide information about labeling of the user interface and other configuration details. These properties are automatically generated, when the NetBeans dialog-generation-wizard is used.

This package contains the following classes:

- `EATChooseRelationshipTypeDialog`
- `EATConcreteGraphPinScene`
- `EATConcreteView`
- `EATEditConcreteAttributeDialog`

`eat.cm.widget`.

The package `eat.cm.widget` contains the Widgets that are displayed on the scene. EntityWidgets and AttributeWidgets are found here, above all the relationships between these components.

This package contains the following classes:

- `ConcreteAttributeWidget`
- `ConcreteEntityRelationshipWidget`
- `ConcreteEntityWidget`
- `ConcreteExternalAttributeRelationshipWidget`
- `ConcreteInternalAttributeRelationshipWidget`
- `EATConcreteGraphPinScene`
- `EATConcreteView`

8.3.2 Class Structure

`eat.cm.tool.attribute.CreateDynamicCPTTool`

This tool creates dynamic CPMs, based on a given aggregation function. For the dynamic CPM creation the classes in `eat.cpm.*` were used. These were mostly taken from the old version of EAT. This tool checks the aggregation function, which is one of its execution parameters. Depending on the type of this function, a corresponding dynamic CPM is created.

`eat.cm.tool.attribute.ConcreteAttributeConnectTool`

Unlike the manual creation of abstract attribute relationships, their counterparts in the concrete modeler are drawn automatically. Therefore, this tool is called every time a ConcreteEntityWidget is created or when a concrete entity relationship is drawn. This tool provides three instead of one `execute` method because it has to decide whether there is a need for a `ConcreteInternalAttributeRelationshipWidgets`, a direct ConcreteExternalAttributeRelationshipWidget, or an indirect ConcreteExternalAttributeRelationshipWidget. The usual process is that the `execute` method provides parameters on which the method decides whether there should be a new attribute relation. The case of an indirect ConcreteExternalAttributeRelationshipWidget makes an exception, because the need for this relationship is already found and verified by the `CheckIndirectAttributeConnectionPathTool`. The necessity-check is done based on the information of the abstract model. The last step is to call the `add-` and `set-` methods of the scene in order to register and initiate the drawing of a necessary attribute relationship.

`eat.cm.tool.attribute.EditConcreteAttributeDialogTool`

This tool provides functionality to handle the dialog for editing attributes in the concrete modeler. The core functionality provided is the setup of the dialog and the retrieval of the edited attribute when the dialog is closed.

`eat.cm.tool.relationship.CheckIndirectAttributeConnectionPathTool`

As mentioned before, this tool determines whether an indirect ConcreteExternalAttributeRelationshipWidget needs to be drawn. This determination process starts when a ConcreteEntityRelationshipWidget was created. Obviously, it is a complex and expensive routine because this check must not only consider the source and target ConcreteEntityWidget of the newly drawn ConcreteEntityRelationshipWidget, but any ConcreteEntityWidget on the scene. For example, a new entity relationship could merge parts of the graph where a node in part one could be the source and a node in part two could be the target of an indirect attribute relationship in the abstract modeler (which would enforce the drawing of an according relationship in the concrete modeler).

To minimize the effort of collecting all necessary information, several `HashMaps` were introduced. Thus, the collection can be done only once and the results can be associated in these maps for quicker access.

The `execute` method is divided into two parts. First, it collects all indirect abstract external attribute relationships including their source and target attributes and entities. After that, it checks for each concrete entity if its according abstract entity is part of those abstract external attribute relationships. Further validations ensure that it is necessary to draw a new indirect concrete external attribute relationship between a source and a target ConcreteEntityWidget.

`eat.cm.tool.relationship.ChooseRelationshipTypeDialogTool`

Concrete entity relationships contain a reference to their associated abstract counterparts that represent the (abstract) type of a concrete relationship. They are related to an abstract entity relationship. As there could be multiple AbstractEntityRelationshipWidgets between two AbstractEntityWidgets, there could also be more than one possible association candidate for a concrete entity relationship. This tool has the task to find these candidates and to send them to a dialog that displays them and waits for the user's choice. The subsequently called dialog is the `EATChooseRelationsipTypeDialog`.

The determination of possible abstract entity relationships that are going to be referenced by the new concrete entity relationship presumes another important task: a multiplicity check. Therefore the `execute` method first gets all abstract entity relationships between the abstract representation of the source ConcreteEntityWidget and the target ConcreteEntityWidget. Then it collects belonging abstract entity relationships from existing concrete entity relationships between source ConcreteEntityWidget and target ConcreteEntityWidget in order to know how many abstract entity relationships are "in use". Finally, with this information, a multiplicity check is performed to know whether it is possible to draw a new concrete entity relationship of an abstract type. With the result (filtered choices), the dialog is called. The tool with its multiplicity check has also impacts on the `EntityConnectTool` as it decides whether it is possible to draw a ConcreteEntityRelationshipWidget between the current source and target at all. It could be the case that the multiplicity-limit is reached or that there is not an according abstract entity relationship to refer to. The drawing will then be canceled.

`eat.cm.tool.scene.CalculateTool`

This tool controls the calculation of concrete models. First, the values are collected, which form the basis for the smile net. Relevant values in this case are CPMs, priors and evidences. The structure of the model (what type of relationships were used, and which aggregation functions were selected) is important for the links inside the network that has to be created. Afterwards, a Bayesian network is built, based on this information. In the next step, the net is calculated. In the last step, the calculated values are noted at the corresponding attributes.

`eat.cm.tool.scene.CollectValuesTool`

This tool collects the values out of the scene and builds a Bayesian network based on these beliefs. The SMILE API (see 7.1.1.1) is used for the building process. For an optimal use of this API, it is recommended to proceed in a defined way (see 8.3.4) [23], which is followed by this tool.

`eat.cm.tool.scene.DisplayResultsTool`

Inside an execution of `eat.cm.tool.scene.CalculateTool`, this tool writes the calculated values back into the attribute. Therefore, the calculated network and the scene are aligned. With the use of the SMILE API, the network can be questioned for the calculated values of a certain AttributeWidget and underlying attribute.

`eat.cm.tool.scene.OpenConcreteModelTool`

This tool loads a concrete model that can be found at a certain file path in an XML file. The unmarshall functionalities provided by Castor (cf. 7.1.3) are. They led to an instantiation of Castor generated classes, which can be processed.

`eat.cm.ui.EATChooseRelationsipTypeDialog`

This dialog appears when a user draws a concrete entity relationship. The user has to choose an according abstract entity relationship that will then own the drawn concrete entity relationship. As explained before this dialog is called by the `ChooseRelationshipTypeDialogTool`. The `setup` method expects a `HashMap` containing all abstract entity relationship candidates calculated by the tool. It displays theses candidates in a `ComboBox` and maps the user's choice to an abstract entity relationship. The chosen abstract entity relationship is then passed back to the `ChooseRelationshipTypeDialogTool`, which returns it as the result to the `EntityConnectTool`.

`eat.cm.ui.EATConcreteGraphPinScene`

This class provides the GraphPinScene for the concrete modeler. Methods for adding and removing elements to the model are provided along with references to necessary additional management data. See EATAbstractGraphPinScene (in 8.2.2) for additional information about fundamental modus operandi.

`eat.cm.ui.EATConcreteView`

This is the main view class of the concrete modeler. The view is shown as main frame inside the Swing Application Framework. Central purpose of this class is to

provide all necessary UI elements like buttons and or menus and their corresponding actions.

Due to a bug in the Swing Application Framework [100], it is necessary for additional views like this to add its own window listener for terminating the application after the window is closed.

`eat.cm.ui.EATEditConcreteAttributeDialog`

This class provides a Dialog for showing information about an attribute of an entity in a concrete model. Additional functionality is the addition of evidence to the attribute. The current CPM and the results of a calculation process can be viewed in this dialog.

`eat.cm.widget.ConcreteAttributeWidget`

This Widget class is used to visualize attributes in entities in concrete models. It holds a reference to the corresponding model element and takes care of connection events by using the `AbstractAttributeConnectTool`, right-click events (`AttributePopupMenuTool`) and in-place editing (`RenameWidgetTool`).

`eat.cm.widget. ConcreteEntityRelationshipWidget`

This is the concrete specialization of an EntityRelationshipWidget. It is the user interface container for the entity relationship data model element in the concrete modeler. It lies in the connection layer of the `EATConcreteGraphPinScene`. In comparison to an `AbstractEntityRelationshipWidget` the `ConcreteEntityRelationshipWidget` does neither contain multiplicity widgets nor role name LabelWidgets, but a central name LabelWidget. Moreover, it holds an action list for providing popup menu and control points. As `ConcreteEntityRelationshipWidgets` have a reference to their belonging abstract entity relationship this class provides getter and setter methods for this variable, too.

`eat.cm.widget.ConcreteEntityWidget`

This class provides a representation of a concrete Entity. It is a subclass of `EntityWidget` and handles mainly the initialization of the Widget. For more information about EntityWidgets see 7.4.4.2.

`eat.cm.widget.ConcreteExternalAttributeRelationshipWidget`

This widget class is the concrete specialization of a relationship between two AttributWidgets whose parent EntityWidgets are different. It is very similar to the AbstractExternalAttributeRelationshipWidget. Because this is a Widget, its main functionality is to provide a field for the data model element, which here is an external attribute relationship. Moreover, actions for popup menu and control point capabilities are assigned. Because an attribute relationship holds information about its associated entity relationships, this association needs to be displayed, too. Therefore, a `ConcreteExternalAttributeRelationshipWidget` computes the targets of its related EntityRelationshipWidgets (precisely the central LabelWidget of the EntityRelationshipWidget). Then, for each associated entity relationship, it creates a `ConnectionConnectionWidget` between the source (LabelWidget of `Con-`

`creteExternalAttributeRelationshipWidget`) and the target. The created ConnectionConnectionWidget is then stored in a `Vector`. In comparison to its abstract peer, this widget is drawn automatically whenever necessary.

`eat.cm.widget.ConcreteInternalAttributeRelationshipWidget`

This widget class is very similar to the `ConcreteExternalAttributeRelationshipWidget` and `AbstractInternalAttributeRelationshipWidget`. It provides exactly the same functionality, but is used when source and target ConcreteAttributeWidgets of the concrete AttributeRelationshipWidget belong to the same parent ConcreteEntityWidget. A special characteristic of this relationship widget is that it has at most one related entity relationship.

8.3.3 User Interface

This paragraph explains the user interface of the concrete modeler, in an analogous way, as it was already done for the abstract modeler (see 8.2.3). Again, the illustration starts with a consideration of the main perspective, which is presented in *Fig. 46*.
In this perspective, a separation into two parts is visible. At the top of the modeler, the

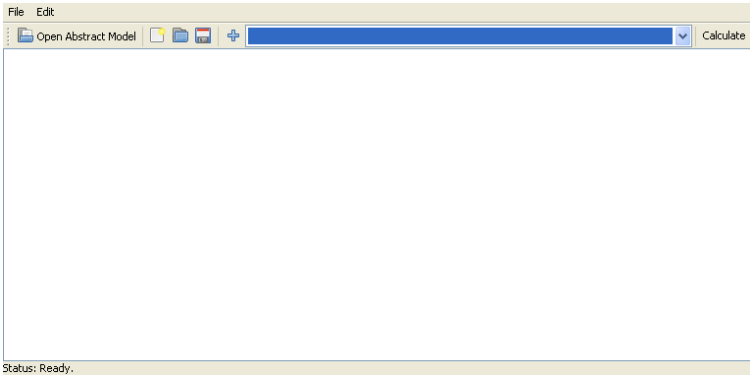


Fig. 46. Empty-Model Perspective of the Concrete Modeler

configuration options are arranged, whereas the center offers space to draw a concrete model. The “File”-menu offers functionalities for starting with a new model, for opening an existing concrete model; also saving is provided. The underlying abstract model can be opened, which means loaded, as well. The “Edit”- menu offers the possibility to set a filter.

With the use of buttons, several tasks can be performed additionally. The discussion of the menu entries starts with the button, which is at the left and continues from left to right. At first is also possible to load an abstract model using a button. The functionalities to create a new concrete model, to load one that was made previously, and to save a concrete model is also provided. The fourth and fifth buttons have to be considered together, as essential functionality is provided by these two. Depending on

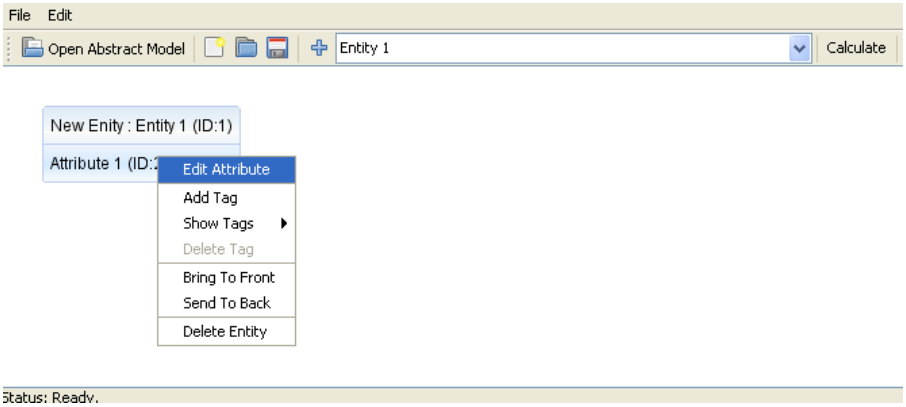


Fig. 48. Popup-menu of a Concrete Entity

the opened abstract model, its entities are displayed in the pull-down menu at the fifth position. Each entity can be instantiated, and in this way added to the current concrete model, with use of the fourth button. Finally, the button, which is on the left, starts the calculation process of the concrete model.

The popup-menu of a concrete entity is very similar to the menu of an abstract entity. The identical opportunities are offered (*Fig. 48*), except for the possibility to add attributes, which is done exclusively in the abstract modeler. Therefore, these tasks are not explained again, as they can be found in the description of the graphical user interface of the abstract modeler. The "Edit Attribute"- functionality is an exception with differing methods of operation, therefore it is considered in the following. On selection of this item, a menu with three different tabs is displayed.

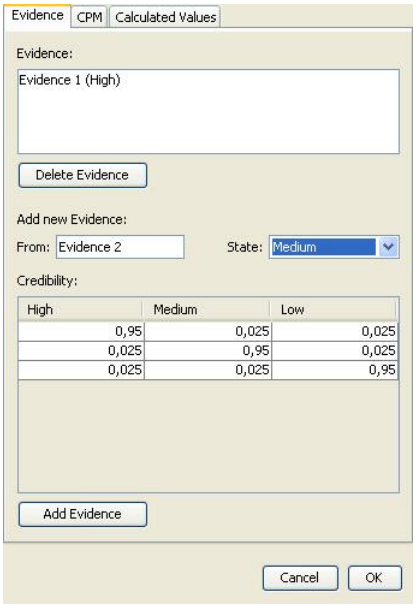


Fig. 47. Add Evidence Dialog of an Attribute

The first one allows the user to add evidences to the current attribute. This is visualized in *Fig. 47*. Evidence for the attribute is stored as CPMs. For each evidence a source and the state, the evidence is corresponding to, can be saved, too.

The second tab displays the CPM, which is configured in the "Edit-Attribute" dialog of the abstract modeler. This CPM cannot be changed anymore at this time.

The third tab that is named "Calculated Values" is used to visualize the probabilities after the execution of the calculation in the concrete modeler. It is visualized in *Fig. 49*. For each state of the considered attribute, a numerical value is indicated.

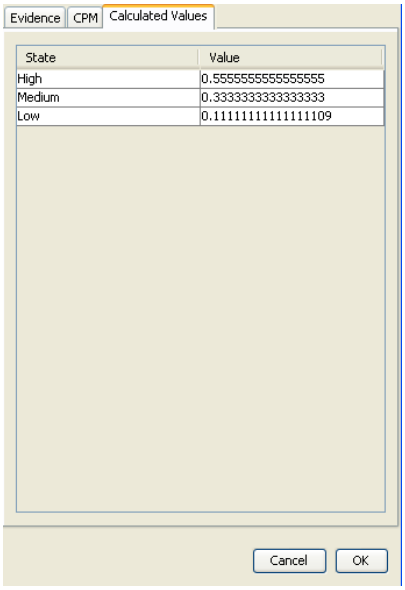


Fig. 49. Report of the calculated values

When a relationship between concrete entities is made in the modeler, the underlying abstract relationship has to be selected. Therefore, each time two entities are connected, a dialog is shown, which offers the abstract relationship candidates. This dialog is visualized below in *Fig. 50:*

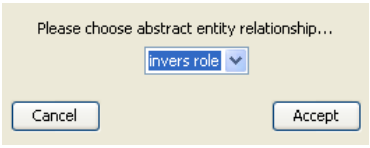


Fig. 50. Selection of the Abstract Entity Relationship

8.3.4 Calculation

In the following, the calculation process is explained. The way from a concrete model over the creation and calculation of a Bayesian network finally to the calculated values is presented. Each important step is visualized with screenshots from the modeler and GeNIe (cf. 7.1.1.2). In this way an illustration of the mathematical process is made possible, which normally remains in the background. The underlying abstract and concrete models as well as a mapping between components of them and the network, which is considered in the following, are available in the Appendix (Appendix A: Calculation).

The network-creation and -calculation algorithm is an adoption of the process described in [24]. This is a bottom-up approach, in which nodes were created first and their connection introduced afterwards. In this scenario, the adjusted method consists of eleven steps, including preparation in advance and calculation after a successful network creation.

The network is created and calculated with the help of the SMILE API (cf. 7.1.1.1).

Before the network creation starts, the scene content is collected and, based on the modeled elements a Castor compliant concrete model is created. To have a valid and complete model from the beginning helps to translate this structure into a Bayesian network. An easy lookup of the model's context can be performed. In this way, it is ensured that the created network remains consistent to the model. Another advantage in this proceeding is that a direct access on the model's components can be performed and the long way from the scene to the underlying Castor compliant elements does not need to be taken.

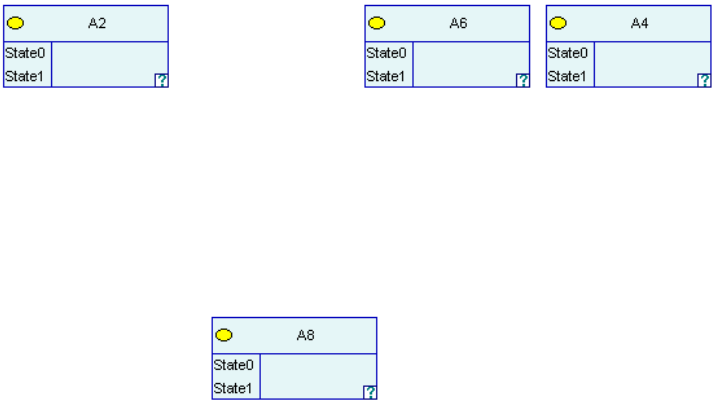


Fig. 51. Step 1: Translation of Attributes to nodes

After the preparatory stage in the first step shown in *Fig. 51*, the AttributeWidgets of all EntityWidgets are read out of the previously generated concrete model. For each

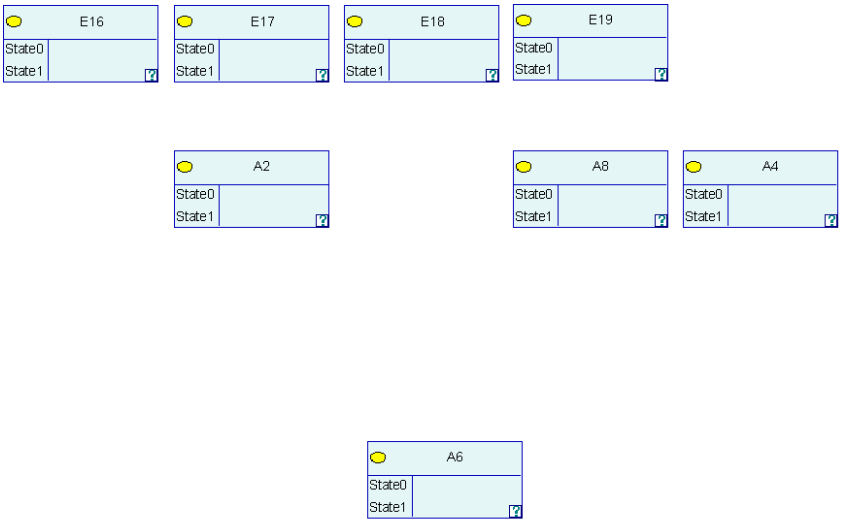


Fig. 52. Step 2: Addition of Evidence nodes

attribute (extracted out of the AttributeWidgets), a node in the network is added. These nodes are named with A (for attribute) and a unique ID, which match with the ID of the underlying attribute. Initially these nodes have two States (State0 and State1), which are adopted in one of the following steps.

The addition to the network that is made in step 2 (*Fig. 52*), is that for each attribute all evidences are collected. These evidences were again added to the net. Their name at each time is set to "E" (for evidence) combined with the unique ID of the corresponding evidence in the concrete model.

In the third step (*Fig. 53*) the States of the nodes, which were created during first and second step, are adjusted. Therefore, all attributes and their attached evidences are considered. Their states are read and used to replace the default states.

The aim of the fourth step is to connect the nodes, which are derived from the attributes. This is shown in (*Fig. 54*). An important aspect of the creation of the attribute connections is that the abstract model, which is the basis of the concrete model to be calculated, has to be considered. It has to be checked, whether an aggregation function node is necessary or not. This needs some computation, as it cannot be read straight from the concrete model. If the abstract model requires an aggregation function node (see 6.4), this node has to be added before the already present nodes can be connected.

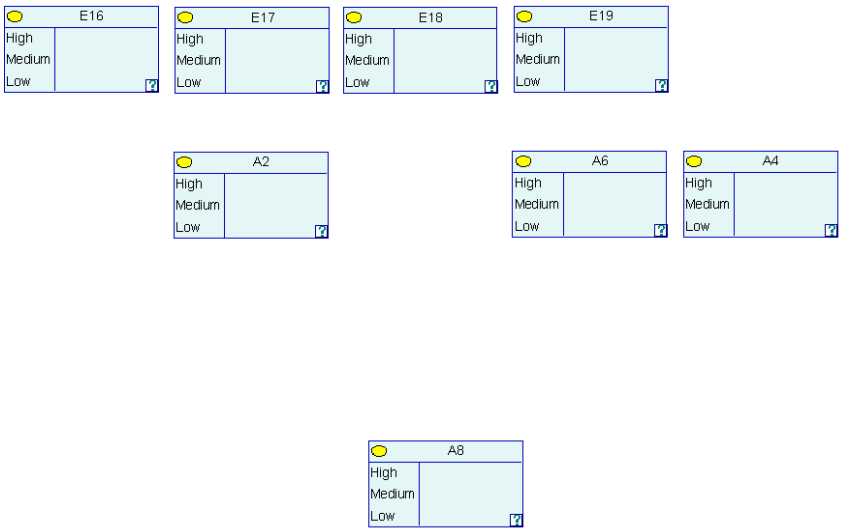


Fig. 53. Step 3: Setting of States

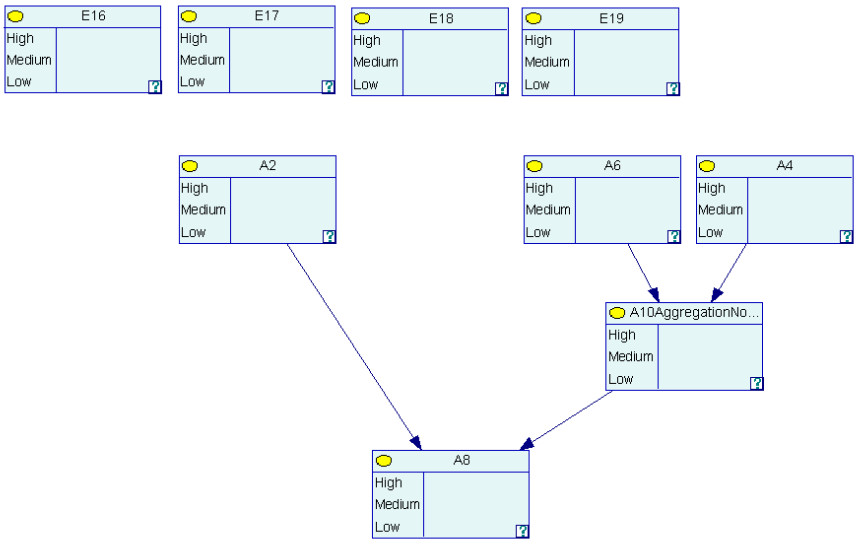


Fig. 54. **Step 4: Connection of Attributes**

This node's states need to be changed, as already done for the other nodes. The name is set to A (again for attribute) followed by the name of the underlying attribute relationship in the abstract model and the remark that this node represents an aggregation function node.

Afterwards, the dependencies, which are represented through directed arcs, can be added. The absence of an aggregation function node implies a direct connection. Otherwise, a connection to the newly created node is introduced. As last sub-step, this aggregation function node is connected, as it was modeled for the ingoing attribute connections in the concrete model.

Step 5, which is not visualized, also deals with the aggregation function nodes. Their CPM's have to be set dynamically, as it is not predictable during the modeling process (in the abstract modeler) how their input will look like, especially how many nodes will connect to them. Therefore, the type of CPM, which is remarked in the abstract model, needs to be read. Based on this information and the knowledge about the number of parents, which is available after the fourth step, a dynamic CPM creation can be done.

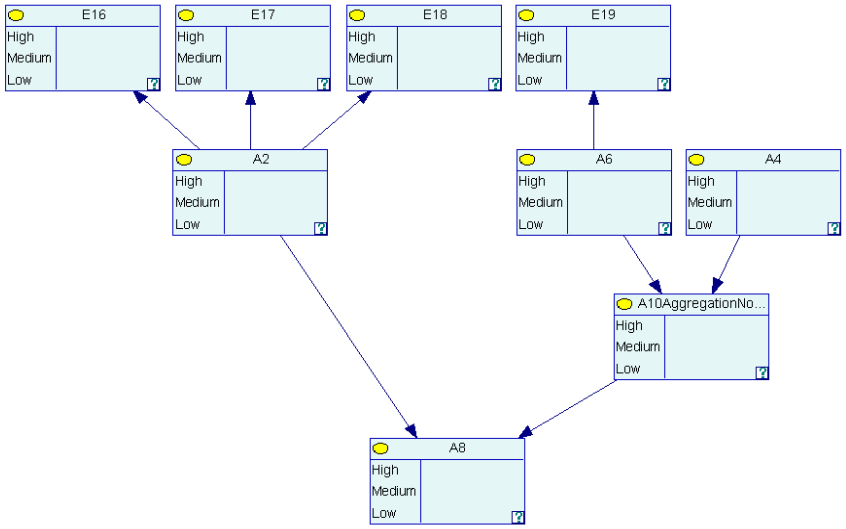


Fig. 55. Step 6: Addition of Dependencies between Attributes and Evidences

In the sixth step (*Fig. 55*), the missing connections between attribute nodes and evidence nodes are added. Therefore, all attributes have to be reconsidered and linked to their respective evidences. Afterwards all relations between nodes are set.

In order that the Bayesian network can be calculated, the information about the nodes CPMs have to be set. This is done in three sequent steps, which again are not visualized:

During step eight the CPMs, which were set in the abstract model, are set for the attributes. Iteration over all attributes is performed. For each attribute, it is decided whether the static CPM, which is defined in the abstract model is used. As described in 6.4 dynamic CPMs were only used, when a zero lower-bound multiplicity at the corresponding attribute of the attribute relationship is allowed. In case of a dynamic CPM is needed, the type of CPM is taken from the abstract model and used to create an appropriate one.

In the seventh step, the CPMs of the evidences are set. They were defined in the concrete model and can easily be looked up through the Castor compliant model created in the beginning.

The eighth step is already a preparation for further implementations. It adds evidences to evidence nodes in the Bayesian network. Currently, these evidence-evidences cannot be set in the concrete modeler, but in the future, this possibility should be offered. However, the (empty) evidences of attributes are read and attached to the evidence nodes in the Bayesian network.

With this last step, the network is complete. The concrete model has been translated into a Bayesian network. All information for inference is available.

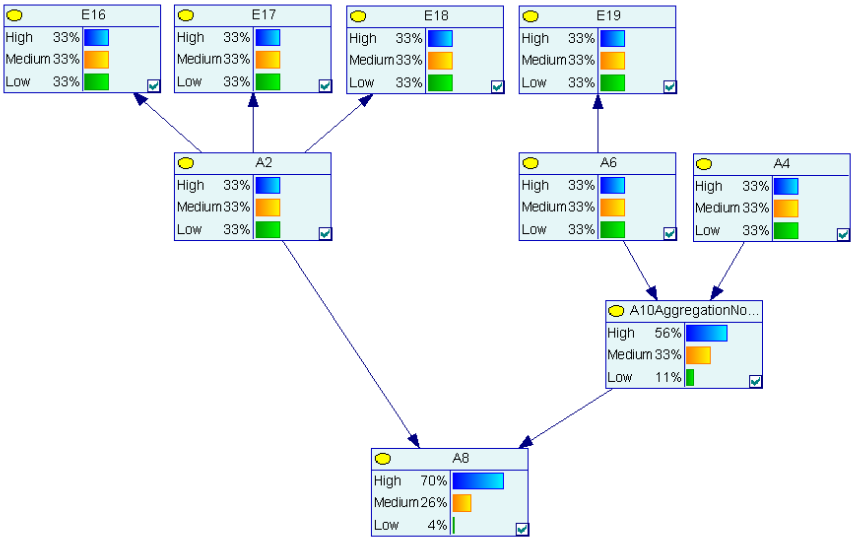


Fig. 56. Step 9: Updated Beliefs

The following figure (*Fig. 56*) shows the networks after its beliefs have been updated. The values of the states have been propagated according to the dependencies of the network.

With this last step, the creation of a Bayesian network and its calculation process is described completely. Now a benchmark on different scenarios can be performed and recommendations based on a mathematical calculation can be made.

8.4 Challenges

In this section, challenges, which were discovered during the implementation process, will be explained. Issues related to the chosen components that are used in the tool are considered on the one hand. On the other hand, complicated, and therefore remarkable, algorithms, which were implemented, are presented.

This section can be considered as a reference book in which solutions for hard nuts are provided. The explanations, which are presented, are for understandable reasons only an extract of all the solutions that were found as the program was created. The criteria, which the elements of the compilation listed below have in common, is that the provided solutions are considered not intuitive, that the only reasonable way was implemented, or that they are very special for a certain case. The remedies, which are provided, are expected to get relevant for further additions again.

8.4.1 Cyclic Dependencies in XSD

Castor is not able to handle XSD elements whose type is equal to its name. Therefore, it is not possible to define an element “Foo” whose type is also “Foo”. The names have to be different like “Foo” as name and “FooType” as type.

Since Castor generates one class for the type, with the name of the type and one class for the element, with the name of the element, the naming and the definition order has to be well considered.

The class of the element is always a subclass of the class generated for the type. Therefore, if an inheritance hierarchy of the type exists, it is necessary to choose the name of the element the same as the super-class of all classes, which should be compliant with the value of the element. The definition of the super-class also has to be made before the definition of the given element.

An example would be the complex type “FooType” which is extended by the complex type “SpecialFooType”. Now an element should be capable of holding values that are compliant with “FooType” and all its subtypes. Because of this, it is necessary to call this element “FooType” as well. This way the complex type that holds the element “FooType” is also able to hold a value of “SpecialFooType”.

More information on XDS can be found in 7.1.2. The real world difficulty can be seen in the XSD (Appendix B: XSD) for an abstract model, specifically in the definition of attributes and the “AggregationFunctionType”.

8.4.2 Property File Management

In its default configuration the NetBeans graphical user interface designer (NetBeans GUI designer) is configured to use property files, to save information about the appearance of the created dialogs. Compared to a user interface creation by hand in which e.g. the label of a certain button is defined within the Java code, the information management of the IDE was found to be different. For each dialog, a properties-file is automatically created once the NetBeans GUI designer is used. These properties-files are stored in a "resources"-subpackage of the package the dialog is created in. These properties-files are structured as maps in which each button has a label assigned. Optionally additional information is stored as short descriptions (displayed when the mouse is moved over this button) as well. When a dialog is created or changed, these properties are irrelevant for the programmer, as the NetBeans GUI designer builds and updates them in case of changes.

During the use of the IDE, it sporadically turned out that these properties-files are handled in a different way than expected. As the code of the involved programmers is sheared with the use of the NetBeans SVN plug-in [76], all the files are administrated by a defined authority. It was expected that the properties-files would be managed in the same way. In some situations, which were found not to be reproducible, the properties-files were not exchanged. It turned out that changes, which were made by a programmer, were not transferred to another programmer, whereas the exchange of Java-files was found to be working. The exchange finally was made possible, as the first user changes were committed again. Sometimes, multiple iteration of this procedure was necessary. In the worst case, a complete project-checkout of the second programmer was needed.

These problems were stopped as the "Automatic Resource Management" of the IDE was switched off. This possibility is offered in the dialog properties. Afterwards the properties-files were not used any longer. The dialog now is realized completely in Java-code.

8.4.3 Wrong JFrame When Using Multiple Views

Because of a bug in the Swing Application Framework, which is used in EAT, the `getMainFrame` method of the application returns a wrong `JFrame`. The so-called main frame is necessary for changing the window title of the application. It also handles the quitting of the application if the window is closed (usually by clicking on the x in the top-right corner). Because a reference to a wrong `JFrame` is returned, it was not possible to terminate the application. It continued to run in the background after the window was closed.

Some research revealed that this is a known bug [100]. The chosen solution consisted of a reference to the necessary frame in the `EATAbstractView` and `EATConcreteView` classes to make it possible to change the title and the implementation of a custom `WindowListener` in the `EATConcreteView` class, which recognizes the close-Event and terminates the application afterwards.

The bug only appeared after the second view for the concrete modeler was introduced. While the application consisted of only one view, the reference was usable for all actions. Only in the case the view is set manually on startup and is not equal to the default one, the reference is unusable. It still references the default view.

8.4.4 Order of Actions

It was found out that the order of assigning actions to a Widget is relevant. Taking for example the instantiation of an `AbstractMultiplicityWidget` in `AbstractEntityRelationshipWidget.java`; in line 59 and 60 two actions were added to the action queue of the `expWidSource`, an instance of an `AbstractMultiplicityWidget`. The first action is a `createMoveAction`; the second action is a `createSelectAction` that provides the `ExpandOnSelectTool` functionality. When the actions are added in this order the `AbstractMultiplicityWidget` on the scene can be moved while holding down the left mouse button on this Widget. When double-clicking on this Widget it expands itself and shows the four multiplicity JRadioButtons. If these actions were added in the reverse order, a drag and drop movement would not be possible because the widget would have expanded itself on the first click (the click needed for holding down the mouse button). Thus, the second action (`createSelectAction`) would be shadowed by the first action.

8.4.5 Deselecting an Expandable Widget

Currently it is not possible to collapse an expandable Widget on deselect. Expandable Widgets are `AbstractMultiplicityWidget` and `RelationAggregationFunctionWidget`. The action that provides the expand capability is the `createSelectAction` of the `ActionFactory`. This action is assigned to these Widgets after instantiation in the `AbstractEntityRelationshipWidget` respectively `AbstractExternalAttributeRelationshipWidget` classes. The `createSelectAction` needs a `SelectProvider` as an argument, which controls the behavior in case of selection. In this case, the `ExpandOnSelectTool` implements the `SelectProvider`. The tool implements the `select` method and defines that the `expand` method is called after selecting these kind of widgets. Unfortunately, the `SelectProvider` does not offer a `deselect` method for determining the behavior in case of deselecting. Using the Visual Library, this is usually handled by the call of a scene's `createSelectAction` without arguments. This method is final and thus cannot be overridden in order to collapse all expanded Widgets. The `notifyStateChanged` method of the expandable Widgets does not notice a deselecting, too. A workaround could be done be overriding the `notifyStateChanged` method in the `EATAbstractGraphPinScene`, `AbstractEntityWidget`, `AbstractEntityRelationshipWidget`, `AbstractEx-`

ternalAttributeRelationshipWidget, and AbstractInternalAttributeRelationshipWidget. In the case of a state change, the method should then collect all expandable Widgets from the EATAbstractGraphPinScene and call their collapse method.

8.4.6 Router Issue

As pointed out in 7.2 there is a performance issue with the OrthogonalSearchRouter in NetBeans Visual Library. This means if there is a sufficient amount of orthogonal routes on the Scene, a movement of an EntityWidget, leading to the recalculation of several paths, will slow down the overall performance significantly. The critical number of routes depends on the machine, but is often reached when five or more connected EntityWidgets on the Scene exist. This problem is a known bug and will be fixed with NetBeans 6.5 [74]. Referring to this API review the problem is that the search router calculation is not free enough, i.e. the anchor should allow arbitrary routing. In the future, the router will use the center of an anchor to calculate the best route first and then makes a fine adjustment of the ends. This will be achieved by introducing two additional methods to the API. The API itself will remain fully backward compatible. Currently, there is a workaround by avoiding the use of the OrthogonalSearchRouter and using the DirectRouter or FreeRouter instead.

8.4.7 Path Algorithms

In order to improve the correctness of the model at design-time, the user is sometimes provided only with possible options. Especially when it comes to relationships between attributes, several preconditions have to be fulfilled. Moreover, the model needs to be consistent after the manual or automatic drawing of such relationships. As attribute relationships can be drawn between indirect connected EntityWidgets, algorithms are necessary to invoke an automatic drawing (in concrete modeler) or to provide only relevant and valid options (in abstract modeler). Following tools offer capabilities to check and ensure connection-consistency: the AttributeConnectionPathFinderTool provides (depth-first search) algorithms for the abstract and concrete modeler, which determine whether there is a path between two Entity Widgets. If there is a path, it also offers methods for finding appropriate neighbors of an entity on a path (from source entity to target Entity). These neighbors are then passed to a dialog that lets the user choose which one to take. The CheckIndirectAttributeConnectionPathTool is called when a concrete entity relationship (Widget) was drawn. It checks if an attribute connection between non-directly connected entities should be drawn, too. Moreover, the Abstract and ConcreteAttributeConnectTool provide routines for determining which relationship should be drawn.

All these tools have in common that the algorithms are very costly. Most of their runtime they collect and compare Widgets and its attributes from the scene. Thus, it was very important to reduce the effort by introducing `HashMaps`, as they store and link related and necessary objects.

9 Usage Example

In this chapter substantial capabilities and typical procedures during the creation of an enterprise architecture model with EAT are explained. It gives an example of how to use the Enterprise Architecture Tool for modeling. The order of actions is not obligatory as EAT is able to retain model consistency throughout the modeling process. Thus, swapping, repeating, or undoing steps, as well as deleting elements will not result in negative formal impacts on the model.

As the usage example closely relates to the example written in [46] it presumes that an analysis and formal creation of an enterprise information system scenario has already been done. This means that this usage example is not about identifying entities, attributes, and relationships in a relevant enterprise environment, but transferring an existing model (which already contains the elements) into a digitally processable enterprise architecture model.

This chapter shows how to use EAT. A detailed explanation of the Abstract and Concrete Modeler's GUI elements can be found in chapters 8.2.3 and 8.3.3. Following the first subsection illustrates what happens in the Abstract Modeler, whereas the second subsection shows the instantiation of the abstract model - the creation of the concrete model.

9.1 Abstract Modeler

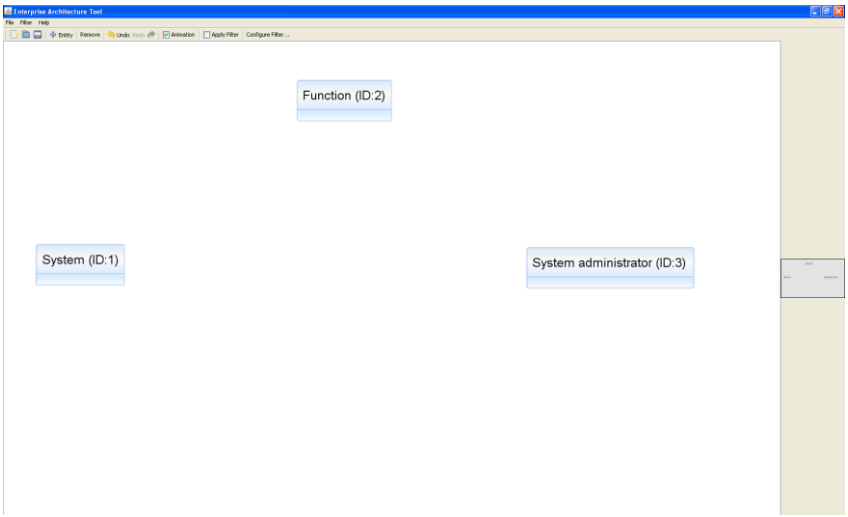


Fig. 57. The Abstract Modeler UI with three created entities

In *Fig. 57* one can see the user interface of the Abstract Modeler. Below the menu bar that is populated with the menu items “File”, “Filter”, and “Help”, one can find the toolbar containing the “new”, “open”, “save”, “add entity”, and more buttons which are explained in 8.2.3. The large, white pane is the drawing surface where all actions take place. Located at the right border of the application is a so-called satellite view, which provides a navigation window that displays the model’s complete surface area.

The first step towards an expedient abstract model is to create the needed entities. For this example, three entities have to be created. In *Fig. 57* there are three corresponding EntityWidgets generated with the help of the “add entity” button and the context menu entry “Add New Entity”, that appears by right-clicking on the empty drawing surface. The entities can be renamed by double-clicking on their label.

The second step, which recurs after each step, is to rearrange the elements via drag and drop.

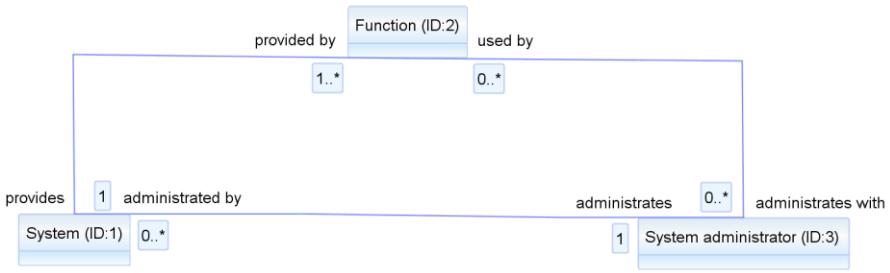


Fig. 58. Entities with entity relationships

The third step is to draw entity relationships between entities whose future attributes will be causally related. This can be done by moving the cursor from a start entity to a target entity while holding down the left mouse button and the CTRL key. Note that the drawing must start in the label area of the entity, not in the attribute area. In addition to the blue line between the source and target entity, there appears a label and a multiplicity box at either side of the connection. The labels represent the role names and they can be renamed after double-clicking. The multiplicity boxes extend themselves after having double-clicked on them. There the user can choose between multiplicities for the connection, as they are "1", "0..1", "0..*", and "1..*". Referring to *Fig. 58* this means, for example, that zero to infinity Systems are administrated by a System administrator. The location of multiplicity boxes and labels can be changed by drag & drop. Their position is relative to the connection. Thus, they move accordingly to the movement of the connection line in cases of rearrangements.

An optional fourth step would be to set, move, or remove control points on the connection lines in order to manipulate the routing.

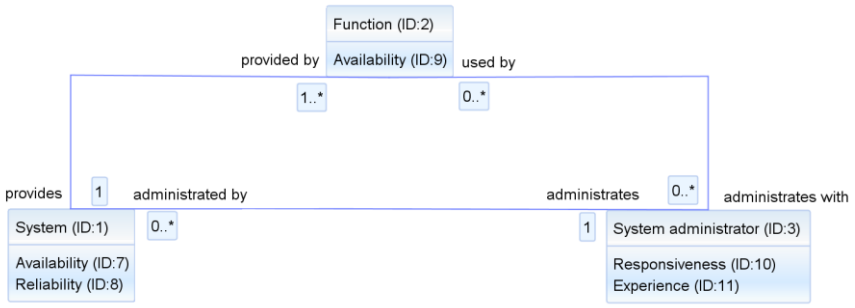


Fig. 59. Abstract model with attributes added

The fifth step is to add attributes to the entities (*Fig. 59*). The attributes represent the property assessment criteria. This means that the quantitative rating of a scenario refers to one of these attributes. More detailed information about the role of attributes, especially with respect to Bayesian networks, can be found in 6.3. Attributes can be

added by right clicking on an entity. The context menu that opens contains a menu item named “Add New Attribute”. It is the first entry in the context menu. Selecting this menu item adds a default attribute to the selected entity. The default attribute “New Attribute” can be renamed either by double-clicking or by opening the “Edit Attribute” dialog. This dialog can be accessed via the context menu of the selected attribute. The dialog is explained elaborately in 8.2.3. It provides possibilities for changing the attribute’s name, configuring the dynamic CPM type, the state values, priors, or static CPM values. More information about CPMs is located in 6.2.

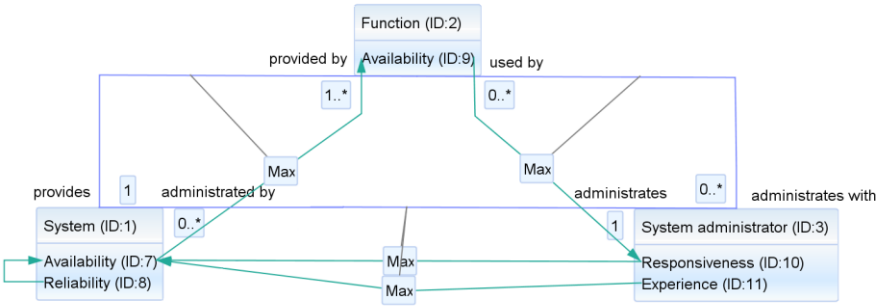


Fig. 60. Added attribute relationships to the abstract model

Drawing relationships between attributes is the sixth step. These relationships can be drawn in the same way as entity relationship. There is only one difference: the mouse cursor must be initially located above the source attribute within the entity. Then, while holding down the left mouse button and the CTRL key, the connection can be dragged to the target attribute. In comparison to entity relationships, which are undirected, the attribute connections are directed. If there is no direct or indirect connection between the entities that the source and target attributes belong to, it will not be possible to draw an attribute relationship, except the source and target entity is the same one (internal attribute relationship). Depending on whether there is a self-looped entity relationship, a dialog pops up after drawing an internal attribute relationship, which wants the user to choose whether he wants to relate the attribute relationship to any of the according entity relationships. In the case of an indirect attribute relationship between different, indirectly connected entities (not in the usage example) another dialog wants the user to choose the path to the target entity (containing the target attribute) in order to determine the related entity relationships (see 8.2.3 and 7.3.3.1). Having drawn the attribute relationship, the aggregation function needs to be configured. Like the multiplicity box, this is another expandable box that provides aggregation function types. These types influence the way the CPM information of parent attributes are aggregated before processing (see 6.3.3 and 7.3.3.1). The small, gray line that appears between an attribute connection and its related entity connection just illustrates their relation. On attribute relationships it is also possible to add, move, or delete control points.

In addition to filtering capabilities for the ease of viewing the model and exporting capabilities that can be used, the last step is to save the abstract model. The example

model in *Fig. 60* is complete now. After having saved this model, it can be loaded and processed in the Concrete Modeler.

9.2 Concrete Modeler

The Concrete Modeler can be started by adding the –concrete parameter on the command line. Its GUI is shown in *Fig. 61*. It is made up similarly to the Abstract Modeler. Below the menu bar, the toolbar can be found. The "Open Abstract Model" button is very important, as it loads the abstract model previously saved. After having loaded the abstract model it will not be displayed, but managed internally to ensure consistency of the instantiated elements – the concrete model. The long combo box on the right of the "save" button contains all entity types. At the right end, there is the "calculate" button that starts the inference engine in order to calculate the attributes' CPM values. Below the toolbar is again the drawing surface. The status bar at the bottom shows the current state of the model. More information about the user interface can be found in 8.3.3.

In order to create a concrete model based on the abstract model built before, the first step is to load the abstract model. This is done by using the "Open Abstract Model" button in the toolbar. Having done this, the combo box shows all entities that have been created in the Abstract Modeler; here: "Function", "System", and "System" administrator. When there is an abstract entity selected, the button with the plus icon adds an instance of this entity type to the scene. It can be renamed on double-click. *Fig. 61* shows that there are two instantiations for each abstract entity, e.g. "Juliet" and "Joseph" as "System administrator". Belonging attributes, as well as internal attribute relationships that are not related to a self-looped entity relationship, appear automatically after the instantiation of an abstract entity. Again, these concrete entities can be rearranged, renamed, filtered, and the view exported. Attributes should not be renamed to maintain consistency with the abstract model. Moreover, it is not possible to add more attributes to an entity. Characteristics of any attribute (e.g. CPM states or priors) are transferred from the abstract to the concrete counterparts, which also remain fixed.

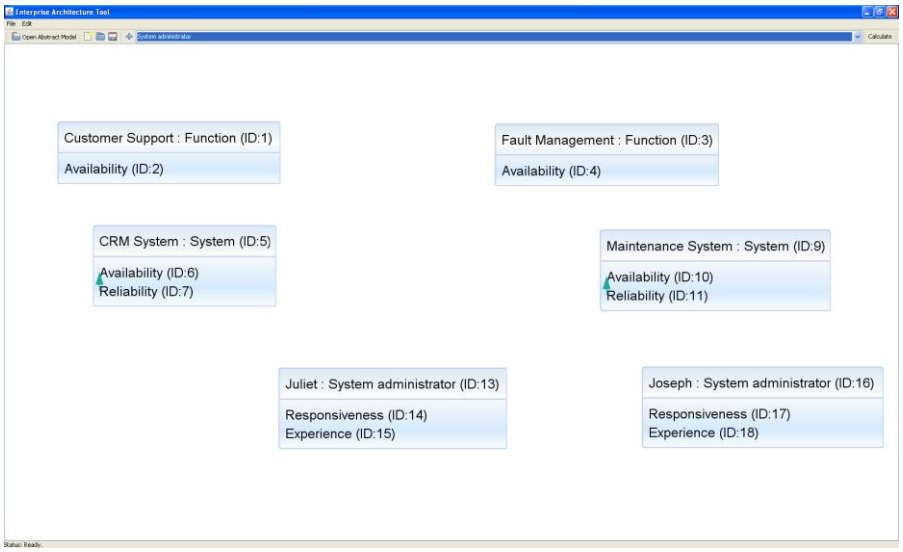


Fig. 61. Concrete Modeler GUI with instantiated entities

The next step is to draw entity connections again. This happens in the same way as in the Abstract Modeler. Several routines are capable of checking whether it is allowed to draw a relationship or not. This depends on available entity relationships in the abstract model and on multiplicity restrictions. Drawing a concrete entity relationship launches several algorithms, which check whether related attribute relationships should be drawn automatically. This is a difference to the Abstract Modeler: attribute relationships cannot be drawn manually, as they have a determined relation in the abstract model on which they are build upon. *Fig. 62* shows the complete concrete model in reference to the example from the paper mentioned at the beginning of this chapter.

Now the last step (besides saving the concrete model) is to push the “calculate” button that initiates the Bayesian network and starts the inference. The result of the calculation can be found in the attributes’ “edit attribute” dialog. This dialog can be accessed by right clicking on an attribute and choosing the “Edit Attribute” menu item in the context menu. More detailed information about the whole calculation process is listed in 8.3.4.

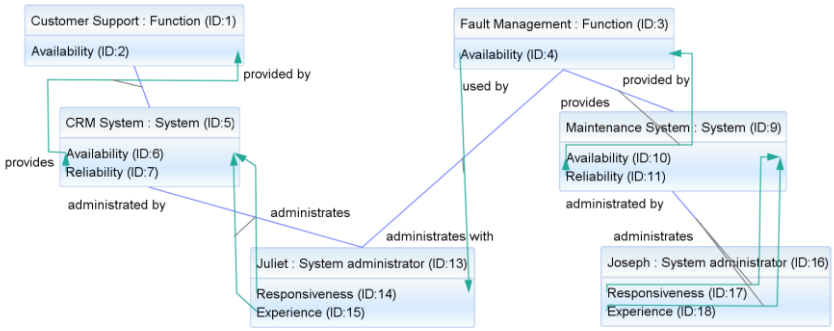


Fig. 62. Complete concrete model

This usage example gave an introduction into how EAT could be used. As it is dedicated to users, it focused on the description of how to create a concrete model starting with a blank drawing surface in the Abstract Modeler. Detailed information about how things work internally can be found in the linked sections and literatures. Moreover, not all capabilities of EAT could be shown in the usage example, as it was chosen due to its pragmatic characteristic. Special cases, e.g. indirect external attribute relationships, were briefly explained and referenced.

10 Extension Guide

This chapter gives additional information about how to extend the Enterprise Architecture Tool. Therefore, it first provides detailed descriptions of the most typical use cases in the EA Tool. These descriptions examine the interaction of participating classes and the call flow. Wherever necessary a figure introduces the use case by illustrating the central thread of its data flow. Pay attention to the fact that the modeling notation is lightweight. The circles denote classes and the arrows indicate the direction of call. Thereby the arrowhead points to the class that is called. Double-bordered circles stand for entry points – the start of the use case whereas thickly bordered circles symbolize the end of the use case.

The second subsection conveys approaches for extending the tool. This is done by explaining exemplary extension scenarios. In doing so, the scenario is divided into as-is, to-be, and transition descriptions.

The last subsection gives further ideas that came up during the implementation. The benefit of this subsection is gained by the categorization and assessment of future extensions.

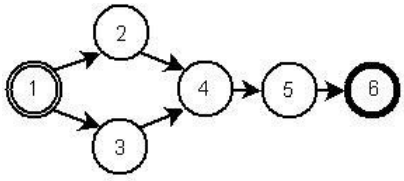
10.1 Use Cases and Their Used Classes

In this section, a mapping between the different use cases and the classes they are using is presented. This is done for three reasons: The first motivation for this section is to document the data flow inside the different use cases. This way, the order of the used components gets clear. If the Java classes, which are described in 8.1.2, 8.2.2 and 8.3.2, are considered in parallel, an understanding of the whole architecture 7.4 will be achieved very easy.

The second motivation is that this section should sensitize developers to parts they have to consider on the way to implement new features. They can find a template for further extensions, considering already existing classes and their underlying data flow. This way, it becomes clear at which place which functionality can already be expected or where the right entry point is, if an addition needs to be made.

Finally, this section helps if debugging is needed. It might happen that the tool will not work as expected, after new features have been added. Therefore, the following sections help to locate the affected classes and to speed up the error correction. The order in which classes are considered for debugging should be derived from the order they are executed during the faulty use case.

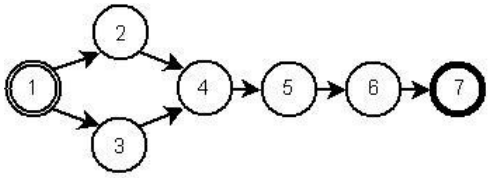
10.1.1 Add Tag



1: eat.am.ui.EATAbstractGraphPinScene
2: eat.tool.popupmenu.Entity.PopupMenuTool
3: eat.tool.popupmenu.Attribute.PopupMenuTool
4: eat.tool.tag_filter.AddTagDialogTool
5: eat.ui.EatAddTagDialog
6: eat.am.ui.EATAbstractGraphPinScene

In case a new tag should be added to an element of the scene, this action is started with a right-click on the scene element to open the popup menu. Depending on the element, the corresponding `eat.tool.popupmenu.`(element)`PopupMenuTool` is executed (e.g. if it is an entity the `eat.tool.popupmenu.EntityPopupMenuTool` is executed). When the "Add Tag" entry is selected, the tool `eat.tool.tag_filter.AddTagDialogTool` is started. This tool displays an `eat.ui.EatAddTagDialog`, which allows entering the name of the label that should be added. Afterwards the scene is shown again.

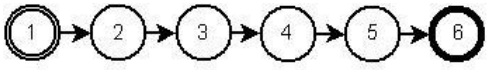
10.1.2 Delete Tag



1: eat.am.ui.EATAbstractGraphPinScene
2: eat.tool.popupmenu.Entity.PopupMenuTool
3: eat.tool.popupmenu.Attribute.PopupMenuTool
4: eat.tool.tag_filter.DeleteTagTool
5: eat.tool.tag_filter.GetTagTool
6: eat.am.ui.DeleteTagDialog
7: eat.am.ui.EATAbstractGraphPinScene

If a label that has been assigned to a scene element should be removed, again this task has to be initiated via the popup menu (as explained in the Add Tag use case `eat.tool.popupmenu.`(element)`PopupMenuTool` is executed). From the menu the `eat.tool.tag_filter.DeleteTagTool` is started. This tool prepares and shows the `eat.am.ui.DeleteTagDialog`, where any tag of the element can be removed. The element's tags were collected with use of the `eat.tool.tag_filter.GetTagTool`. Finally the modeling process can be continued.

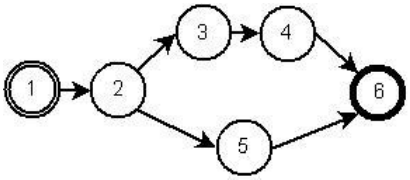
10.1.3 Filter (Configuration)



1: eat.am.ui.EATAbstractView
2: eat.tool.tag_filter.FilterDialogTool
3: eat.tool.tag_filter.GetAllTagsTool
4: eat.am.ui.EATConfigureFilterDialog
5: eat.tool.tag_filter.FilterDialogTool
6: eat.am.ui.EATAbstractView

The composition of new filter criteria is started when the "Configure Filter" button in the `eat.am.ui.EATAbstractView` is pushed. This leads to a start of the `eat.tool.tag_filter.FilterDialog.Tool`. The task of this tool is to prepare and display the `eat.am.ui.EATConfigureFilterDialog`, which allows configuring the filter criteria. In case of tag-based filtering all assigned tags were accumulated from the `eat.tool.tag_filter.GetAllTagsTool`, so that an individual selection of considered tags can be made. When the filter has been configured, it can be applied, which is explained in the following. Therefore, the configuration is saved at the `eat.tool.tag_filter.FilterDialog.Tool`.

10.1.4 Filter (Execution)

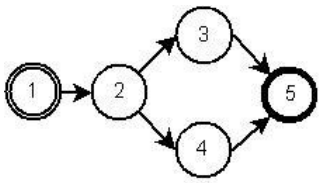


1: eat.am.ui.EATAbstractView
2: eat.tool.tag_filter.FilterTool
3: eat.tool.tag_filter.FilterDialogTool
4: eat.tool.tag_filter.UnfilterTool
5: eat.tool.tag_filter.UnfilterTool
6: eat.am.ui.EATAbstractView

When a configured filter should be utilized, the "Apply Filter" box, in the main (`eat.am.ui.EATAbstractView`) perspective has to be checked. Hereupon the `eat.tool.tag_filter.FilterTool` is executed. This tool reads the configuration, made previously, which is stored in the `eat.tool.tag_filter.FilterDialog.Tool`. Afterwards it uses the `eat.tool.tag_filter.UnfilterTool` to set the whole scene content to be not visible. In the next step the elements that should be displayed, depending on the configuration, are searched and set visible again.

In case the filter should be switched off, an un-checking of the "Apply Filter" box again starts the `eat.tool.tag_filter.FilterTool`. The only operation that is performed in this scenario is to execute the eat.tool.tag_filter.UnfilterTool, which brings back all model elements to the scene.

10.1.5 Open (Load) of an Existing Model



1: eat.tool.scene.OpenSceneTool
2: eat.tool.scene.NewSceneTool
3: eat.tool.scene.OpenAbstractModelTool
4: eat.cm.tool.scene.OpenConcreteModelTool
5: eat.tool.scene.OpenSceneTool

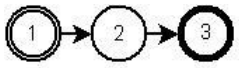
A previously modeled scenario could be loaded using the main menu. A click on the "Open" button in the file menu starts the loading process and displays an open-dialog. The way presented here is also taken when the "Open abstract model"-button in the concrete modeler is pushed. In all cases, the `eat.tool.scene.OpenScene-`

Tool is executed, parameterized with the file-path configured in the open dialog. At first, the current scene content is removed and the scene is reinitialized by the eat.tool.scene.NewSceneTool. Depending on the model to be opened either the eat.tool.scene.OpenAbstractModelTool or the eat.cm.tool.scene.OpenConcreteModelTool is started next. The used tool returns the respective model. Out of this model information, the scene is created by the eat.tool.scene.OpenSceneTool, so that the modeling can be continued.

10.1.6 New Model

A selection of the "New"-button, the icon in the modeling perspective, or the entry in the file menu equally, runs the eat.tool.scene.NewSceneTool. This cleans the scene and reinitializes the model.

10.1.7 Save Model



1: eat.tool.scene.SaveSceneTool
2: eat.tool.scene.CreateModelsTool
3: eat.tool.scene.SaveSceneTool

Saving can be achived in multiple ways. This includes, pushing the "Save" Button as icon or menu entry and saving via shortcut (Ctrl + S). In all situations the eat.tool.scene.SaveSceneTool is started. Internally the eat.tool.scene.CreateModelsTool is used, to transform the scene's content into a Castor compliant model. At the end of this activity, an XML document describing the current model is available.

10.1.8 Save Model ("Save As" Usage)

When a "Save As" action is executed inside the tool, the same tasks are performed as already explained for the Save model use case. The only difference is, that a dialog is shown to configure the path, were the resulting XML document should be saved (the use of eat.filehandling.XmlFileFilter ensures that only valid filenames are assigned to the output). This path is remembered at the scene, so that this configuration is not necessary in the future anymore.

10.1.9 Calculate Concrete Models

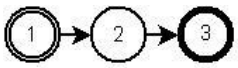
1: eat.cm.ui.EATConcreteView
2: eat.cm.tool.scene.CalculateTool
3: eat.cm.tool.scene.CollectValuesTool
4: eat.tool.scene.CreateModelsTool
5: eat.cm.tool.attribute.CreateDynamicCPTTool
6: eat.cm.tool.scene.CalculateTool
7: eat.cm.tool.scene.DisplayResultsTool
8: eat.cm.ui.EATConcreteView

The calculation of a concrete model is caused by pushing the "Calculate" button in the `eat.cm.ui.EATConcreteView`. This executes the `eat.cm.tool.scene.CalculateTool`, which administrates the calculation. At first, a Bayesian network is created which is done by the `eat.cm.tool.scene.CollectValuesTool`. Inside the `eat.cm.tool.scene.CollectValuesTool` a concrete model, is built with use of the `eat.tool.scene.CreateModelsTool`. This is the basis for the net. Depending on the structure the `eat.cm.tool.attribute.CreateDynamicCPTTool` is applied to create dynamic CPMs, when they are needed. These dynamic CPMs are built with use of `eat.cpm.CPT` and all of its specializations.

When the network has been created, the `eat.cm.tool.scene.CalculateTool` executes the `updateBeliefs` method of the Bayesian network to calculate it.

The last step is to execute the `eat.cm.tool.scene.DisplayResultsTool`. This tool propagates the calculated values back to the scene, so that they are visible for the user of the modeler.

10.1.10 Add EntityRelationshipWidget



1: eat.am.ui.EAT*GraphPinScene
2: eat.tool.entity.EntityConnectTool
3: eat.am.ui.EAT*GraphPinScene

When the user wants to add a new EntityRelationshipWidget to the scene, he starts this task by pressing the CTRL key and left clicking on an EntityWidget. This is called an extended action because an additional key has to be pressed.

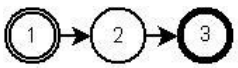
This action invokes the drag handling tool `eat.tool.entity.EntityConnectTool`. This happens because the tool is added to an EntityWidget's action queue as a `createExtendedConnectAction` in `eat.widget.EntityWidget`.

The `EntityConnectTool` implements methods of the `ConnectProvider` interface in order to determine the source and target Widget that is positioned under the mouse pointer during the mouse-dragging process. If there is a valid target Widget, the `createConnection` method prepares the new connection depending on whether the action takes place in the abstract modeler or concrete modeler. In this method, the data model element `EntityRelationship` is created and filled with necessary information. Finally, the method indirectly calls the scene's methods by calling the following library functions:

- `addEdge`: calls `attachEdgeWidget` which creates an according Widget
- `addPin`: (twice, for source and target); calls `attachPinWidget` that adds PinWidgets to the appropriate source and target (serve as connection points)
- `setEdgeSource`: calls `attachEdgeSourceAnchor` that sets the start of the connection
- `setEdgeTarget`: calls `attachEdgeTargetAnchor` that sets the end of the connection

Having done these calls the scene is validated and the new relationship is registered and drawn. This workflow is the same for the AbstractEntityRelationshipWidget and the ConcreteEntityRelationshipWidget.

10.1.11 Add AttributeRelationshipWidget in the Abstract Modeler

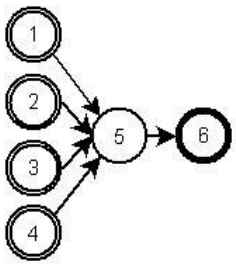


1: eat.am.ui.EAT*GraphPinScene
2: eat.am.tool.attribute.AbstractAttributeConnectTool
3: eat.am.ui.EAT*GraphPinScene

In principle, the workflow for adding an AttributeRelationshipWidget to the abstract scene is the same as adding an EntityRelationshipWidget (see 10.1.10). Because in the concrete modeler attribute relationships are added automatically, instead of manually, the `createExtendedConnectAction`, that executes the `eat.am.tool.attribute.AbstractAttributeConnectTool`, is not located in the `eat.widget.AttributeWidget`, but in the `eat.am.widget.AbstractAttributeWidget`. Apart from that, the `AbstractAttributeConnectTool` provides the same methods with the same purpose as the `EntityConnectTool`. It also calls the library functions that invoke the scene methods for registering and drawing the new connection. The reason for more lines of code in the `Ab-`

stractAttributeConnectTool is the fact that it has to distinguish between three types of AttributeRelationshipWidget: direct ExternalAttributeRelationship-Widgets, indirect ExternalAttributeRelationshipWidgets, and InternalAttributeRelationshipWidgets.

10.1.12 Add AttributeRelationshipWidget in the Concrete Modeler

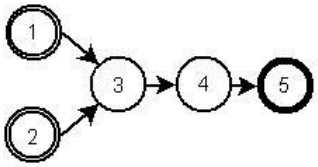


1: eat.cm.ui.EATConcreteView
2: eat.tool.entity.EntityConnectTool
3: eat.tool.popupmenu.ScenePopupMenuTool
4: eat.cm.tool.relationship.CheckIndirectAttributeConnectionPathTool
5: eat.cm.tool.attribute.ConcreteAttributeConnectTool
6: eat.cm.ui.EATConcreteGraphPinScene

AttributeRelationshipWidget in the concrete modeler are added automatically whenever necessary. Therefore, the `eat.cm.tool.attribute.ConcreteAttributeConnectTool` is not assigned as a `createExtendedConnectAction` in a Widget, but is called at several points in the code. It is called every time a ConcreteEntityWidget was created and subsequent checks diagnosed a need for an AttributeRelationship. Currently, this happens in the `addEntityAction` method in `eat.cm.ui.EATConcreteView`, in the `createConnection` method in `eat.tool.entity.EntityConnectTool`, in the `actionPerformed` method in `eat.tool.popupmenu.ScenePopupMenuTool`, and in the `execute` method of the `eat.cm.tool.relationship.CheckIndirectAttributeConnectionPathTool`.

Having called the `ConcreteAttributeConnectTool`, the flow is the same as in 10.1.10 and 10.1.11.

10.1.13 Add Entity in the Abstract Modeler

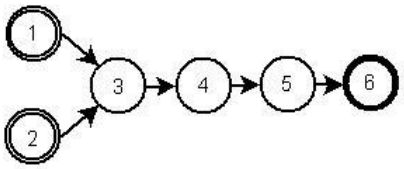


1: eat.am.ui.EATAbstractView
2: eat.tool.entity.AddEntityTool
3: eat.tool.popupmenu.Attribute.PopupMenuTool
4: eat.tool.scene.UndoManagerTool
5: eat.am.ui.EATAbstractView

Entities can be added to an abstract model in two ways. One is the "+ Entity" Button in `eat.am.ui.EATAbstractView`. When the button is clicked, the `addEntityWidgetAction` method is invoked. In this method, the `eat.tool.scene.UndoManagerTool` is used to store the according edit for undo and redo. In addition, the `addNode` method of the Scene is called, which utilized the `eat.tool.entity.AddEntityTool` to place a new Entity Widget on the scene.

The second way is to right click the scene and to choose "Add Entity" from the context menu. On right click, the `eat.tool.popupmenu.ScenePopupMenuTool` is invoked. The `actionPerformed` method afterwards uses the `addNode` method of the Scene to place the entity on the Scene. Thereafter the Widgets preferred location is set to the current mouse position.

10.1.14 Add Entity in the Concrete Modeler



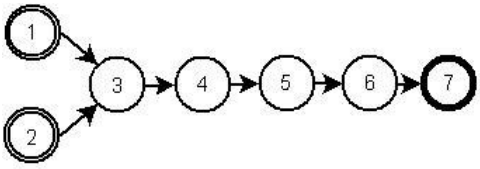
1: eat.cm.ui.EATConcreteView
2: eat.tool.popupmenu.ScenePopupMenuTool
3: eat.cm.ui.EATConcreteView.chooseAbstractEntityComboBox
4: eat.tool.entity.AddEntityTool
5: eat.cm.tool.attribute.ConcreteAttributeConnectTool
6: eat.cm.ui.EATConcreteView

In the concrete modeler, entities can be added using the "+" button in the toolbar of `eat.cm.ui.EATConcreteView`. The invoked `addEntityAction` creates a

new concrete entity. The according abstract entity is read from the `chooseAbstractEntityComboBox`. Then the `addNode` method form the Scene handles the creation of the EntityWidget by using the `eat.tool.entity.AddEntityTool`. Finally, the `eat.cm.tool.attribute.ConcreteAttributeConnectTool` is invoked to handle all necessary relationships.

The second way is to right click the scene and to choose "Add New Entity" from the context menu. The invoked `eat.tool.popupmenu.ScenePopupMenuTool` creates an entity in the way described above. Finally, the preferred location of the new Entity Widget is set to the current mouse-pointer position.

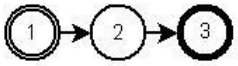
10.1.15 Delete Entity



1: eat.am.ui.EATAbstractView
2: eat.tool.popupmenu.EnityPopupMenuTool
3: eat.tool.scene.UndoManagerTool
4: eat.tool.RemoveEnitiyTool
5: eat.relationship.RemoveEntityRelationTool
6: eat.relationship.RemoveAttributeRelationshipTool
7: eat.am.ui.AbstractGraphPinScene

To delete, one could use the "Remove" Button in `eat.am.ui.EATAbstractView`, which invokes the `removeAction` method to remove all selected entities, or it is possible to right-click an entity and select "Delete Entity" from the popup menu, which is displayed by the `eat.tool.popupmenu.EnityPopupMenuTool`. Both ways perform the following action: At first, the Edit is stored in the `eat.tool.scene.UndoManagerTool`. Then the `eat.tool.RemoveEnitiyTool` is used to remove the entity and the matching EntityWidget. This is done by removing all in- and out-going Relations using the `eat.relationship.RemoveEntityRelationTool` and the `eat.relationship.RemoveAttributeRelationshipTool`. In the end, the entity is removed by calling the `removeNode` method in the Scene.

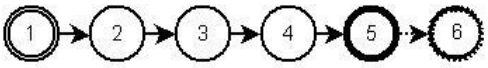
10.1.16 Add Attribute



1: eat.tool.popupmenu.EntityPopupMenuTool
2: eat.tool.scene.UndoMangerTool
3: eat.tool.attribute.AddAttributeTool

Whenever an attribute has to be added to an entity, this could be done by right clicking on an EntityWidget and selecting "Add attribute". In the `actionPerformed`-method of the `eat.tool.popupmenu.EntityPopupMenuTool` the `eat.tool.scene.UndoMangerTool` is used to store the extension and its reverse action. Afterwards the `eat.tool.attribute.AddAttributeTool` is invoked and the attribute is created and added to the entity and the matching EntityWidget.

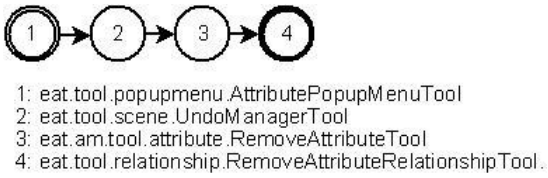
10.1.17 Edit Attribute



1: eat.tool.attribute.AttributePopupMenuTool
2: eat.tool.attribute.AttributeHelperTool
3: eat.tool.attribute.EditAttributeDialogTool
4: eat.am.ui.EditAttributeDialog
5: eat.tool.attribute.AttributeHelperTool
6: eat.tool.scene.UndoManagerTool

If the `actionPerformed`-method of the `eat.tool.attribute.AttributePopupMenuTool` is invoked by right clicking on an Attribute Widget and selecting "Edit Attribute" the `eat.tool.attribute.AttributeHelperTool` is invoked in the first step to save a deep copy of the attribute, which should be edited. In the second step, the `eat.tool.attribute.EditAttributeDialogTool` is used to initialize and display the `eat.am.ui.EditAttributeDialog`. When the dialog is finally closed, the `eat.tool.attribute.AttributeHelperTool` is again used to check whether the attribute was really altered. If so, the `eat.tool.scene.UndoManagerTool` is used to store the original and the changed attribute for future restoration.

10.1.18 Delete Attribute



The `actionPerformed`-method of the `eat.tool.popupmenu.AttributePopupMenuTool` is invoked in the case an attribute should be removed from an entity. The method uses the `eat.tool.scene.UndoManagerTool` to store the attribute for future restoration. After that, the `eat.am.tool.attribute.RemoveAttributeTool` is used to remove the attribute. It also takes care of the in- and out-going attribute relations and deletes them via the `eat.tool.relationship.RemoveAttributeRelationshipTool`.

10.2 How to extend X

In order to convey a sense of how to handle the EA Tool when it comes to adding features, this paragraph provides exemplary extension scenarios. These scenarios have been chosen so that they affect at least two parts of the Model-View-Controller approach and thus are sufficiently complex. They are not explained down to the last detail but they should offer an approach of how to add complex features to the EA Tool by means of finding the right points in the code to hook into.

10.2.1 Saving the Position of an Abstract Entity Relationship Widget's Children

The situation:
Referring to section 7.4.4.3, an AbstractEntityRelationshipWidget has four visible children: two LabelWidgets and two AbstractMultiplicityWidgets. As the action queue of these widgets is added with a `createMoveAction` in `eat.am.widget.AbstractEntityRelationshipWidget`, these children can be selected and moved anywhere on the scene. Nevertheless, they still belong to their AbstractEntityRelationshipWidget, i.e. if the position of the relationship changes, the position of these children changes proportionally.

The problem:
When the abstract model is saved, the saving process ignores the current positions of the relationship's children Widgets. Hence, the positions of the four Widgets are set back to default when an abstract model is loaded.

The idea:
Consider the current positions of a relationship's four children Widgets when the abstract model is saved.

An approach to a solution:
As the position information should survive a shutdown of the tool, it is obvious that they cannot be managed at runtime. The fact that this information should be stored in the abstract model leads to the point that a data model element must take up the position data. As these data belongs to an AbstractEntityRelationshipWidget's children, the according data model element `eat.am.model.EntityRelationshipType` has to store this information. Thus, this element needs to be extended with four pairs of X- and Y-coordinate attributes and according getter and setter methods. How this can be done via the XSD file is explained 7.1.3.

Having done this, a closer look at the affected classes and tools is necessary. Naturally, the `SaveSceneTool` must be involved in this process. This tool manages the save workflow by reading scene contents, creating an abstract model and serializing it. However, this tool calls another tool. This leads us to the `collectValues` method in the `CreateModelsTool`. Here all scene elements including the EntityRelationshipWidgets are collected. This is an ideal point to store all current positions of the children Widgets into the EntityRelationshipWidget's element. Doing this before the entity relationship is added to the abstract model ensures that the positions will be saved, too.

Now, because the `OpenSceneTool` simply invokes a creation of a new AbstractEntityRelationshipWidget (with the help of the stored entity relationship), it has to be ensured that the saved positions are considered during the creation. Finally, this means that the positions brought with the EntityRelationship data model element should be set to the children Widgets in `eat.am.widget.AbstractEntityRelationshipWidget` (because this tool is called in the scene's `attachEdgeWidget` method that is invoked by the `addEdge` call in the `OpenSceneTool`).

10.2.2 Extending the Undo-Redo Functionality

The situation:
Several editing actions performed in the abstract modeler can be reversed by clicking the Undo Button. The actions are stored as edits in the `UndoManagerTool`, which uses a simple queue to store these Edits. Each action requires implementing a specific Edit as subclass of `EATEdit` to enable its cancelation.

These Edits have to be supplied with the needed data and stored in the `UndoManagerTool` at the right point.

The problem:
The most challenging part is to define the right inverse action for undoing the modification. It has to be kept in mind that more undo and redo actions could have taken place before and after the current action. So it is clear that there are only very few reliable parts in the surrounding environment. The most reliable information is about the model elements, which could be identified by their ID. All other objects like Widgets could be removed and restored again as completely new object with different references.
Another challenge is finding the right position to store the action in the `UndoManagerTool`. It is not possible to store the undo information in the according tools themselves because in this case they are not usable for the inverse action anymore.

An approach to a solution:
The best position for inserting undo information is possibly the point where a tool is used. Therefore, the usage of a tool is covered by the edit necessary for undoing.

Finding the correct inverse action depends on the action, which should be enabled for undo and redo. Best results were created if the model elements themselves were stored for restoration. In addition, the usage of the tools is positive since the technique of the action is already implemented.

10.3 Further Ideas

During the implementation of EAT, several possibilities for future extensions were discovered. The primary reason they were not implemented was that their implementation was considered too time-consuming. The aim was to have a complete tool-based support of the modeling process, rather than having an imbalance of features on the one hand and missing essential functionalities on the other hand. Even though some features besides the core-functionalities were already realized.

The tool as it is now provides powerful abilities to create models and to deal with them afterwards. Consequently, extensions will normally not provide any features that are impossible with the current version. The benefit of these add-ons is that they will offer an easier or faster solution compared to the one that is already implemented. Usually, future additions will complement the automatic functionalities of the tool and reduce manual tasks.

Upgrades implemented in the future will enlarge either the abilities of the model or the functionalities of the graphical user interface (GUI). In this way, an improvement for the abstract or concrete model is provided on the one hand or the scene is somehow equipped with more features on the other hand. In a few special cases, benefits for the models and the GUI at the same time could be achieved.

The suggestions, which are presented below, are organized in the following structure. First, the implementation ideas, which add benefit to the models, are discussed.

Next, extensions with influence on models and GUI are presented. In the third subparagraph look and feel upgrades of EAT provided by future add-ons are outlined. Finally, an outlook concerning further additions is presented.

10.3.1 Abstract Modeler and Concrete Modeler Additions

As the tool was implemented, it turned out that there were multiple ways to create models which were not valid (e.g. cyclic dependencies of attributes or attribute dependency chains, build on attributes, which have various different states). Models based on such structures cannot be calculated, and therefore these compositions have to be discovered. This can be done by the implementation of numerous validation functions, which either checks the model on creation time or before the calculation is performed. Thus, it is recommended to implement such functionality. As a result, the modeling process would be eased and the occurrence of errors would be reduced.

In an abstract model, it sometimes happens that a certain entity has one or more specializations; an addition of attributes is provided by them. On the one hand, these specializations share some common features; on the other hand, they have unique properties as well. The occurrences of multiple specializations leading to a granular level of entities are also expected. This pattern can be found in the software development as well, where it is called inheritance. With the introduction of inheritance, the model structure would be eased and complexity of diagrams would be reduced. This would be achieved through the fact that fewer relationships were necessary as relationships of the parent were automatically taken over for their specializations. The number of displayed attributes would be reduced, too. This would lead to more space on the scene. In case this feature would be displayed analog to inheritance defined for UML Class Diagram [77], which is very common, understanding of numerous experts is ensured.

During the modeling process, several decisions have to be made. These determinations of the models have to be documented, in order to understand them afterwards. There is also a good possibility explanations of certain aspects inside the model structure will be required. Therefore, the ability to make comments on the different model elements (e.g. entities, attributes and their relations) should be implemented.

There are already some use cases implemented concerning tagging and the deletion of those tags. Improving these use cases in order to speed up the modeling process is suggested. As a result, it is recommend to extend the addition of tags, so that during this process the already assigned tags (no matter to which element they have been attached) are displayed. If a user-defined selection is made, the concerned tags should be assigned to the current element. Multiple additions of tags simultaneously and a reduction of manual entries are expected to be the benefit of this feature. To select multiple scene elements at one time and to add a common tag to all of them is also suggested as a use case to be implemented. Deletion of tags from more than one element is suggested to be possible in an analog way.

10.3.2 Model and GUI Enhancements

The additions, which were explained so far, deal with the abstract and concrete model. Now two features are presented whose implementation would improve the models and the GUI.

To clean up the scene on the one hand and to bundle elements with common properties on the other hand, the possibility of grouping the scene content would create large benefit. An easier processing of multiple scene elements would speed up the model creation as changes made on the group were propagated to all of its members. The scene's content would be more structured as a minimization of the group's members visualization to a single icon seems to be possible. If a certain group element needs to be edited, as a box, this group could be expanded to provide access to this element. In case the model should be rearranged, a movement of the group icon would end up in an adjustment of all of the group's elements positions.

Another add-on enlarging model and GUI functionalities, is suggested by the implementation of a wizard to create models. As a result, the model creation would be supplemented by a dialog-based extension of the modeling as it is already made possible. With help of the wizard, all of the information, necessary for abstract or concrete model, would be collected. Afterwards a mapping of these values into a corresponding model would result. This guide would ensure that all data that is necessary (e.g. for the calculation functionality) is available. As the models were created automatically and their elements were arranged without manual influence, the resulting models would have a common structure. Thus, they would be easy to understand and well arranged. The implementation of this wizard is considered complex and time-consuming; on the other hand, the use of the tool would be much more guided. This is seen as a large benefit, especially for users withlimited or no experience, or no well-versed contact person.

10.3.3 Interface improvements

This last subsection explains features of the user interface that are suggested to be implemented in the future. The appearance of the models could be improved in several ways. A decision has to be made between improvements of the whole model and improvements of a certain element of the model.

The whole model could be visualized in a better way through the implementation of an auto-layout feature. With help of this functionality, the entities on the scene would be arranged in an optimal way. Therefore, the best distances between the entities would be calculated. Afterwards the elements would be moved, compliant to these values. The resulting model is expected to look very heterogeneously standardized and structured.

The next optional feature that should be discussed is to implement a grid, on which the model would be built. This grid would define placeholders to add elements on the scene. This way, it would be ensured that the scene structure has a certain organization, defined by the structure of the grid. A functionality to switch off the grid should also be implemented, to allow the individual positioning of the scene's content. The

use of the grid would move the positions of the elements so that they were aligned compliant to it.

The grid would be implemented to be switched on and off as required. This approach could also be transferred for other helpers, like ledger lines and spacer. They could be displayed on demand, to help the user. With the use of these features, it would be easier to structure a scene and improve its appearance.

The image of a model would also benefit if the content could be colored more individually. A certain section could be colored apart from the rest of the scene to remark that its content has something in common and diverges from the other modeled elements. Similarities could be expressed by the use of similar colors whereas contrasts of colors could be used to express borders.

Different colors of sections lead to more individualism of a certain aspect or entity. A future addition could offer functionality to change the appearance of a certain entity. This could be done with the use of preconfigured shapes or icons. In addition, the possibility to add an image or define an individual color seems to be possible but more complex. Especially for the storage of user-defined icons, the loading and saving as it is implemented now needs to be adopted.

The connections implemented so far, could also be improved within further extensions: For example the user could have the choice to select between different routing algorithms, e.g. direct router or orthogonal router. Moreover, there could be an option, which controls the rerouting after an EntityWidget has been moved. Here the user could select between option one: calculate complete new route, or option two: only update the last control point of the connection. Another aspect concerning the connections is that it could be discussed whether each entity should be equipped with a fixed number of connectors for relationships to connectors of other entities. These connectors would be evenly spread so that the appearance of the resulting model would be improved, as all relationships would be well regulated.

The modeling process itself could be made better, with implementation of the usual Copy & Paste functionalities. Already modeled sections could be duplicated and be inserted into the model a second time. This feature again reduces the time needed to create a model as redundant work is avoided. It has to be noted that this functionality would work based on simple copies. Changes on the underlying IDs, with consideration of the assigned IDs to all model elements, would be mandatory.

The perspective of the user of the modeler could also be improved, if tabs, similar to the tab-functionalities of popular web browsers would be added. Especially in the concrete modeler, it seems to be helpful to consider more than one model at the same time. With this possibility, a chronological workflow would be encouraged and comparison between different models would be simplified.

When the calculation process is finished, the results are displayed numerically in the properties of their underlying attributes. This could be changed to diagrams, which would ease the understanding of the values, especially when different colors were used. On the other hand, all calculated values could be presented at one certain place so that a manual (collected-) value collection is not necessary anymore.

The filter functionality as it is implemented now provides only basic configuration possibilities. Complex inquiries could only be realized with the help of additional tags, only added for filtering. Therefore, these functionalities could be enlarged. A consideration of more model aspects (e.g. cardinalities or certain model structures)

could be added in the future. Benefit would also be created if sentential connectives were introduced. This would allow the creation of complex inquiries in a catchy and intuitive way.

At this time, the creation of exports is realized with two simple buttons. They could be replaced by a dialog. This dialog could provide functionalities to select the image section to be considered. Quality and type of the output could be influenced at that place, as well.

Analog to this export dialog a print dialog could be implemented as well. This dialog should provide typical configuration possibilities, such as format and alignment of the printout.

The last suggestion that should be discussed is a configuration or properties dialog. This dialog should collect all the different configuration possibilities at one place. On the one hand, this would clean up the user interface, on the other hand all options were offered at one single place. There would be no need for the user to search a certain switch, as they all were present at the same time and place.

The extensions of EAT described above have to be considered as a selection of possible add-ons. It should be noted that this list is not exhaustive. The features presented can be considered as possible candidates, which are not too complex to be implemented, based on the current code basis. These suggestions have in common that the tool, as it is now, is expected to have a large benefit from every proposal that is accepted. This does not mean that additional features are urgently needed to be implemented in order to make the tool work; it is already applicable now.

On the other hand, it is not recommended to implement each idea. This would lead to redundancies and the use of EAT would get more complicated, as several approaches could be followed to get to a certain result. The clearness of the user interface of the tool in the current version would get lost. This means that more or less the opposite of the intent would be achieved.

The implementation of a subset of the additions described above increases the value of the tool. Functionalities improving the displayed abstract or concrete model help the user during model creation and the viewers, who have a look at a certain scenario afterwards. Extensions on the models seem to speed up the modeling process and avoid that the same task has to be done a second time.

Nevertheless, before adding a feature it has to be evaluated which benefit is expected. Add-ons, which provide functionalities that are not, used very often mainly increase the code complexity and decrease the maintainability of the tool. The trade off whether an additional feature is worth implementation has to be made. This is because implementations often are considered more time-consuming than expected at the beginning of their development.

The features presented can be considered as an inspiration for individual ideas. They show, which aspects are worthy of consideration, and explain the different scopes that could be considered during an implementation.

As the tool is fast-growing software whose code is changing very quickly and often, the recommendation refer to the current version. It is possible that additions made in the future will make some of the suggestions needless or not possible to be realized. On the other hand, the need of a certain feature might increase as more and more use cases (which might be implemented in the future) might profit from it.

11 Discussion and Conclusion

The main objective of this diploma thesis was to implement a prototypic software tool for supporting architectural decisions by providing routines for enterprise architecture analysis. This goal was achieved by implementing the abstract modeler and the concrete modeler, which facilitate scenario definition, instantiation, evidence handling, and quantization calculation. The tool - EAT - enables comparative assessments of system qualities of an enterprise information system scenario (and its environment) and gives an estimate of its credibility.

The development process was conducted in a 4-phase-loop based on the spiral model [3].

- Plan next phase;
- Determine objectives, alternatives, and constraints through prioritized requirements list;
- Develop and test;
- Feedback in daily and weekly meetings.

A clear and prioritized requirements list especially leveraged the progress in these phases.

The evaluation of IDE's at the beginning of the project proved beneficial, as the selected IDE, NetBeans IDE, in conjunction with the Visual Library, made a very uncomplicated and fluent development process possible. The only disadvantage in using this highly capable IDE was a noticeable performance lag in comparison to Eclipse. The design of the tool's architecture, including preliminary considerations for module segregation based on the MVC architecture, eased the process of distributed programming, too. By using the commonly used XML Schema Definition (XSD) for the description of the structure of the models, it was possible to store the models in XML files. Due to the high interoperability of XML processing it was possible to separate the data structure from the modeling tool. This way, a guaranteed future for the modeled scenarios was achieved. The use of jSMILE eased the mathematical processing of the models. This way GeNIe-compatible Bayesian networks were created and calculated with a minimum amount of preparation on the part of the EAT implementation.

The Enterprise Architecture Tool in its current state and this thesis mainly serve the purpose for providing a solid, modularized and well-documented base for further development. Nevertheless, it can already be used for educational purposes in architectural modeling, as an as-is modeling and documentation tool, or as an enterprise architecture analysis software.

The probabilistic approach taken in the enterprise architecture analysis method, with the Bayesian network serving as the mathematical backbone for the assessment of a scenario's qualities, has the advantage that it states a clear value of a decision. The prediction and diagnosis abilities of Bayesian networks enable a non-collaborative decision support that is independent of historic decisions (cf. [114]). Nevertheless, this can also be seen as a disadvantage, as there is nearly no verification (at least the credibility and structural correctness can be considered) and thus confidence in the calculation results is necessary. On the other hand, the tool makes a quantitative assessment, although many states of the scenarios could be uncertain.

Moreover, it considers the environment of an enterprise information system and thus comprises human factors into the decision, too.

The problems of large CPTs resulting from extensive concrete models (and thus Bayesian networks), were met by introducing intermediary attributes with the help of aggregation functions and by utilizing PRM slot chains through enabling indirect attribute connections. The solution for the problem of gathering huge amounts of evidences – their prioritization – has been described, too. Nevertheless the complexity of computation and presentation of large Bayesian networks still needs research efforts, e.g. the reduction of complexity by removing arcs with a weak influence [49].

Appendix A: Calculation

In this appendix, the abstract and concrete model is pictured on which the description of the calculation process in section 8.3.4 is based. Concluding, a mapping between these Modes and the final Bayesian network is presented.

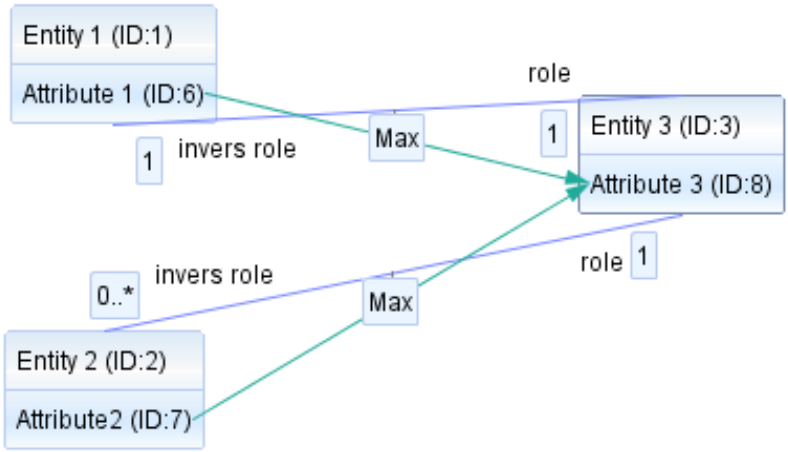


Fig. 63. Underlying Abstract Model

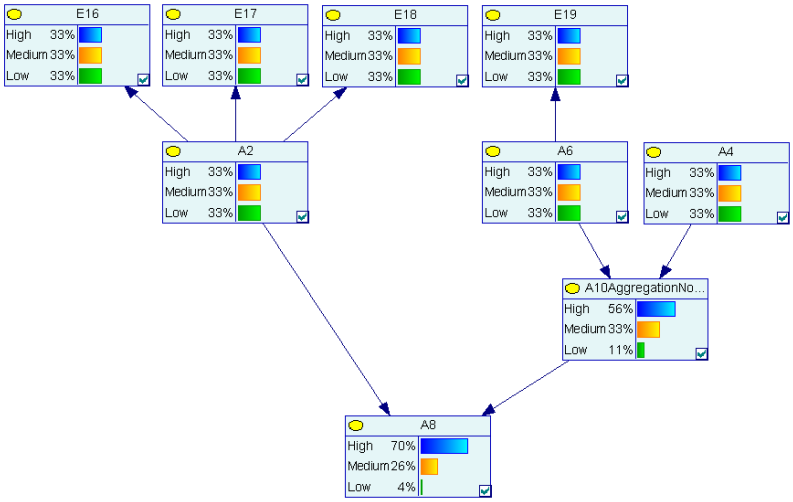


Fig. 64. Resulting Bayesian network

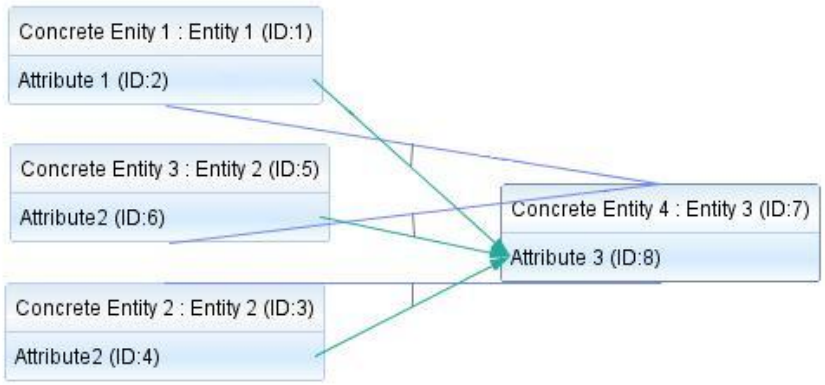


Fig. 65. Concrete Model (created using above Abstract Model)

Table 6. Mapping Bayesian network and models

Element in Bayesian Network	Corresponding Element in abstract (AM) or concrete model (CM)
Node A2	Attribute 1 (ID:2) (CM)
Node A6	Attribute 2 (ID:6) (CM)
Node A4	Attribute 2 (ID:4) (CM)
Node A8	Attribute 3 (ID:8) (CM)
Node A10AggregationNode	Maximum Aggregation Function on relation between Attribute 2(ID:7) and Attribute 3(ID:8) (AM)
Evidence E16	Evidence for Attribute 1 (ID:2) (CM) (not visible)
Evidence E17	Evidence for Attribute 1 (ID:2) (CM) (not visible)
Evidence E18	Evidence for Attribute 1 (ID:2) (CM) (not visible)
Evidence E19	Evidence for Attribute 2 (ID:6) (CM) (not visible)
Connection A2 to E16	Assignment of Evidence E16 to Attribute 1 (ID:2) (CM) (not visible)
Connection A2 to E17	Assignment of Evidence E17 to Attribute 1 (ID:2) (CM) (not visible)
Connection A2 to E18	Assignment of Evidence E18 to Attribute 1 (ID:2) (CM) (not visible)
Connection A2 to A8	Relationship between Attribute 1 (ID:2) and Attribute 3 (ID:8) (CM) build on Relationship between Attribute 1 (ID:6) and Attribute 3 (ID:8) (AM)
Connection A6 to E19	Assignment of Evidence E19 to Attribute 2 (ID:6) (CM) (not visible)
Connection A6 to A10AggregationNode	Relationship between Attribute 2 (ID:4) and Attribute 3 (ID:8) (CM) build on Relationship between Attribute 2 (ID:7) and Attribute 3 (ID:8) (AM)
Connection A4 to A10AggregationNode	Relationship between Attribute 2 (ID:6) and Attribute 3 (ID:8) (CM) build on Relationship between Attribute 2 (ID:7) and Attribute 3 (ID:8) (AM)
Connection A10AggregationNode to A8	Relationship between Attribute 2 (ID:7) and Attribute 3 (ID:8) (AM)

Appendix B: XSD

In this appendix, the XSD documents, the underlying of the models, are explained. At first the relevant elements (precise: their type definitions) are visualized. Therefore, Altova XMLSpy was used[9]. Afterwards the entire XSD files are presented.

Abstract Model

At the top level (*Fig. 66)* of the XSD document a split-up into “Entity”, “EntityRelationship” and “ExternalAttributeRelationship” was made. The fields “elementName”, “elementId”, “Tag(s)” and “Comment” were defined as their common attributes.

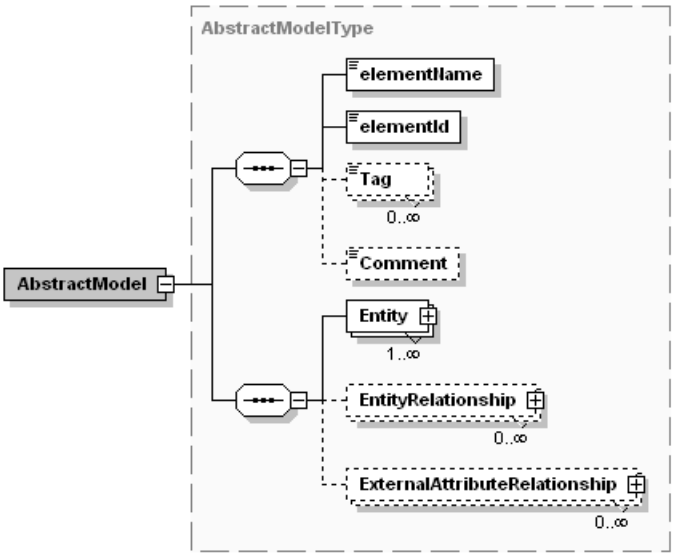


Fig. 66. Abstract Model Overview

9 Altova XMLSpy, which is developed by Altova (http://www.altova.com), is considered to be one of the most popular XML editors [109]. It has multiple capabilities to modify XSD documents and to visualize them.

Each “Entity” (*Fig. 67*) was allowed to have numerous “Attributes”. Between those attributes, “InternalAttributeRelationships” were made possible. Moreover the position of the “Entity”, with X and Y coordinates, was stored (“XPos”, “YPos”).

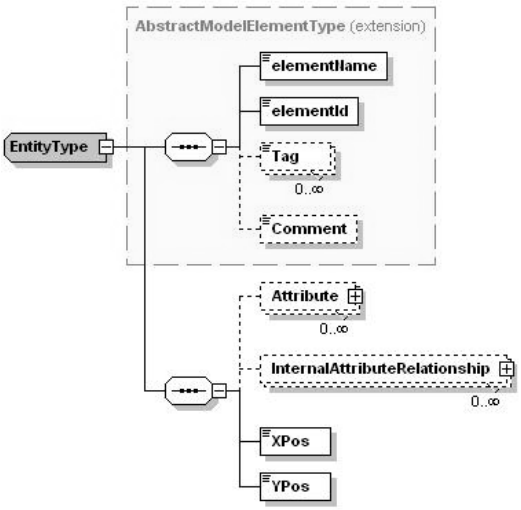


Fig. 67. Abstract Model Entity

The attributes “elementName”, “elementId”, “Tag” and “Comment” were saved for an “Attribute” (*Fig. 68*) likewise. Supplemental the “AggregationFunctionType” was added.

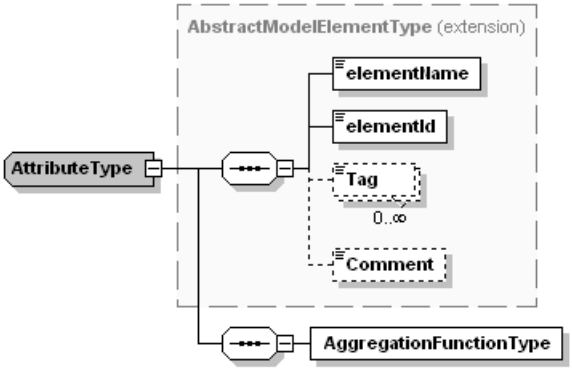


Fig. 68. Abstract Model Attribute

Besides the common properties that were also made available for the scene's other elements, more connection details were stored for an "EntityRelationship" (*Fig. 69*). The IDs and names of origin- and target-entity were kept. In addition, the respective multiplicity was saved. Finally, the "RelationshipAggregationFunction" was stored too. The conventional properties of an element can also be found in an "ExternalAttributeRelationship" (*Fig. 70*). IDs of origin and target were stored too. The capability

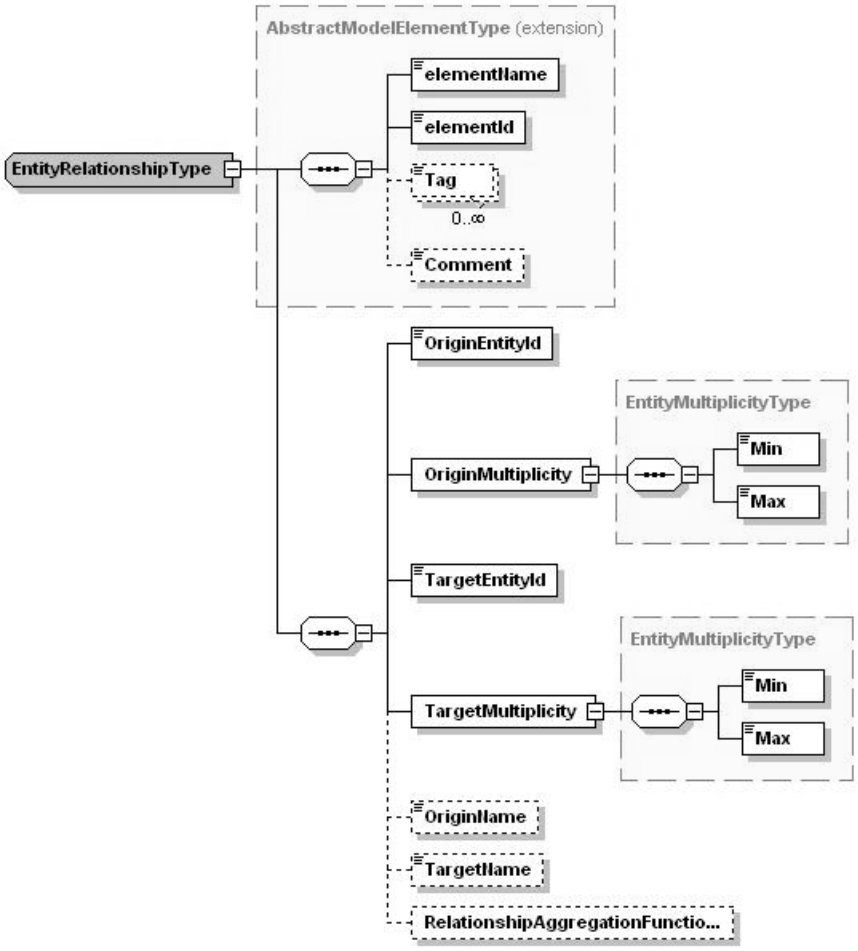


Fig. 69. Abstract Model Entity Relationship

for saving multiple "EntityRelationships" and a "RelationshipAggregationFunction" was provided, too.

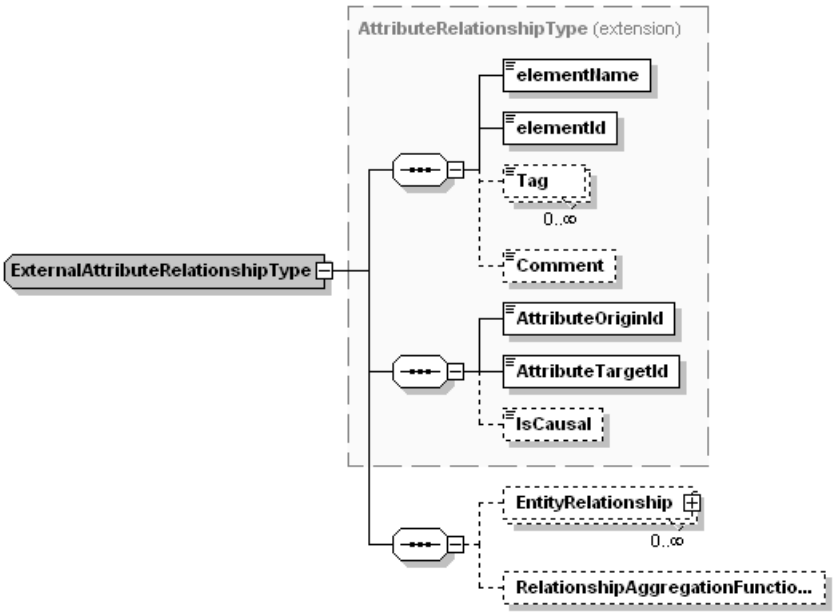


Fig. 70. Abstract Model External Attribute Relationship

The properties of an "InternalAttributeRelationship" (*Fig. 71*) are known from the "ExternalAttributeRelationships". A difference was made concerning the entity relationships, where just one instance is allowed to be saved. In addition, no aggregation functions were stored.

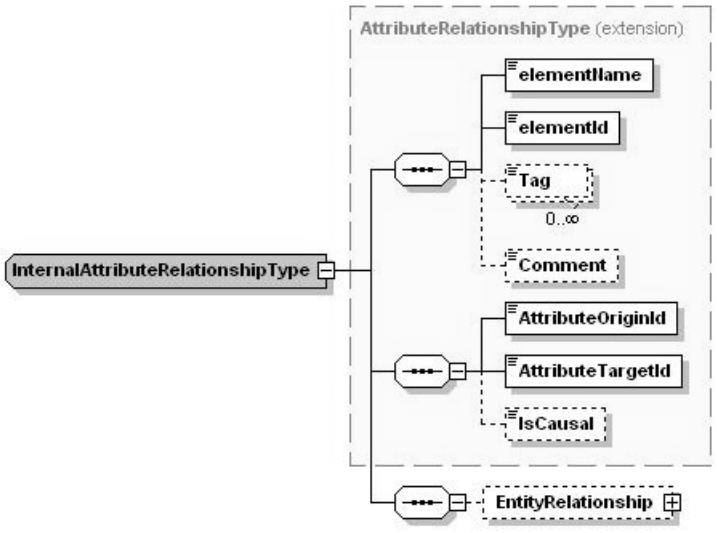


Fig. 71. Abstract Model Internal Attribute Relationship

On the following pages, the content of the abstract model XSD document is presented:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd=http://www.w3.org/2001/XMLSchema
    targetNamespace=http://www.ics.kth.se
    xmlns=http://www.ics.kth.se
    elementFormDefault="qualified">
<xsd:complexType
    name="RelationshipAggregationFunctionType">
</xsd:complexType>
<xsd:complexType name="MaxRelType">
    <xsd:complexContent>
        <xsd:extension
            base="RelationshipAggregationFunctionType">
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
<xsd:complexType name="MinRelType">
    <xsd:complexContent>
        <xsd:extension
            base="RelationshipAggregationFunctionType">
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
<xsd:complexType name="AverageRelType">
    <xsd:complexContent>
        <xsd:extension
            base="RelationshipAggregationFunctionType">
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
<xsd:element name="AbstractModel" type="AbstractModelType" />
<xsd:complexType name="AbstractModelElementType">
    <xsd:sequence>
        <xsd:element minOccurs="1" maxOccurs="1" name="elementName"
            type="xsd:string" />
        <xsd:element minOccurs="1" maxOccurs="1" name="elementId"
            type="xsd:int" />
        <!-- Addition for Tagging -->
        <xsd:element minOccurs="0" maxOccurs="unbounded" name="Tag"
           type="xsd:string" />
        <xsd:element name="Comment" type="xsd:string" maxOccurs="1"
           minOccurs="0">
        </xsd:element>
    </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="AbstractModelType">
    <xsd:complexContent mixed="false">
        <xsd:extension base="AbstractModelElementType">
            <xsd:sequence>
                <xsd:element minOccurs="1" maxOccurs="unbounded"
                    name="Entity" type="EntityType" />
                <xsd:element minOccurs="0" maxOccurs="unbounded"
                    name="EntityRelationship"
                    type="EntityRelationshipType" />
                <xsd:element minOccurs="0" maxOccurs="unbounded"
                    name="ExternalAttributeRelationship"
```

```
                    type="ExternalAttributeRelationshipType" />
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
<xsd:complexType name="EntityType">
    <xsd:complexContent mixed="false">
        <xsd:extension base="AbstractModelElementType">
            <xsd:sequence>
                <xsd:element minOccurs="0" maxOccurs="unbounded"
                    name="Attribute" type="AttributeType" />
                <xsd:element minOccurs="0" maxOccurs="unbounded"
                    name="InternalAttributeRelationship"
                    type="InternalAttributeRelationshipType" />
                <xsd:element name="XPos" type="xsd:int" minOccurs="1"
                    maxOccurs="1"/>
                <xsd:element name="YPos" type="xsd:int" minOccurs="1"
                    maxOccurs="1"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
<xsd:complexType name="EntityRelationshipType">
    <xsd:complexContent mixed="false">
        <xsd:extension base="AbstractModelElementType">
            <xsd:sequence>
                <xsd:element minOccurs="1" maxOccurs="1"
                    name="OriginEntityId" type="xsd:int" />
                <xsd:element minOccurs="1" maxOccurs="1"
                    name="OriginMultiplicity"
                    type="EntityMultiplicityType" />
                <xsd:element minOccurs="1" maxOccurs="1"
                    name="TargetEntityId" type="xsd:int" />
                <xsd:element minOccurs="1" maxOccurs="1"
                    name="TargetMultiplicity"
                    type="EntityMultiplicityType" />
                <xsd:element name="OriginName" type="xsd:string"
                    maxOccurs="1" minOccurs="0">
                </xsd:element>
                <xsd:element name="TargetName" type="xsd:string"
                    maxOccurs="1" minOccurs="0">
                </xsd:element>
                <xsd:element name="RelationshipAggregationFunctionType"
                    maxOccurs="1" minOccurs="0"
                    type="RelationshipAggregationFunction">
                </xsd:element>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
<xsd:complexType name="AggregationFunctionType">
    <xsd:sequence>
        <xsd:element minOccurs="1" maxOccurs="unbounded" name="State"
            type="xsd:string" />
        <xsd:element minOccurs="1" maxOccurs="unbounded" name="Prior"
            type="ProbType" />
        <xsd:element name="CPM" type="CPMType" maxOccurs="1"
            minOccurs="0">
        </xsd:element>
    </xsd:sequence>
```

```
</xsd:complexType>
<xsd:complexType name="MaxType">
    <xsd:complexContent>
        <xsd:extension base="AggregationFunctionType">
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
<xsd:complexType name="MinType">
    <xsd:complexContent>
        <xsd:extension base="AggregationFunctionType">
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
<xsd:complexType name="NegationType">
    <xsd:complexContent>
        <xsd:extension base="AggregationFunctionType">
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
<xsd:complexType name="ExclusiveOrType">
    <xsd:complexContent>
        <xsd:extension base="AggregationFunctionType">
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
<xsd:complexType name="FrequencyType">
    <xsd:complexContent>
        <xsd:extension base="AggregationFunctionType">
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
<xsd:complexType name="AdderType">
    <xsd:complexContent>
        <xsd:extension base="AggregationFunctionType">
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
<xsd:complexType name="ParametricType">
    <xsd:complexContent>
        <xsd:extension base="AggregationFunctionType">
            <xsd:sequence>
                <xsd:element minOccurs="1" maxOccurs="unbounded"
                    name="parameter" type="xsd:double" />
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
<xsd:complexType name="WeightedType">
    <xsd:complexContent>
        <xsd:extension base="AggregationFunctionType">
            <xsd:sequence>
                <xsd:element minOccurs="1" maxOccurs="unbounded"
                    name="weight" type="xsd:double" />
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
<xsd:complexType name="AttributeType">
    <xsd:complexContent mixed="false">
        <xsd:extension base="AbstractModelElementType">
```

```
            <xsd:sequence>
                <xsd:element minOccurs="1" maxOccurs="1"
                    name="AggregationFunctionType"
                    type="AggregationFunction" />
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
<xsd:complexType name="EntityMultiplicityType">
    <xsd:sequence>
        <xsd:element minOccurs="1" maxOccurs="1" name="Min"
             type="xsd:string" />
        <xsd:element minOccurs="1" maxOccurs="1" name="Max"
             type="xsd:string" />
    </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="AttributeRelationshipType">
    <xsd:complexContent mixed="false">
        <xsd:extension base="AbstractModelElementType">
            <xsd:sequence>
                <xsd:element minOccurs="1" maxOccurs="1"
                    name="AttributeOriginId" type="xsd:int" />
                <xsd:element minOccurs="1" maxOccurs="1"
                    name="AttributeTargetId" type="xsd:int" />
                <xsd:element minOccurs="0" maxOccurs="1" name="IsCausal"
                    type="xsd:boolean" />
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
<xsd:complexType name="InternalAttributeRelationshipType">
    <xsd:complexContent mixed="false">
        <xsd:extension base="AttributeRelationshipType">
            <xsd:sequence>
                <xsd:element name="EntityRelationship"
                    type="EntityRelationshipType" minOccurs="0"
                    maxOccurs="1">
                </xsd:element>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
<!-- <xsd:complexType name="ExternalAttributeRelationship">-->
<xsd:complexType name="ExternalAttributeRelationshipType">
    <xsd:complexContent mixed="false">
        <xsd:extension base="AttributeRelationshipType">
            <xsd:sequence>
                <xsd:element minOccurs="0" maxOccurs="unbounded"
                    name="EntityRelationship"
                    type="EntityRelationshipType" />
                <xsd:element name="RelationshipAggregationFunctionType"
                    maxOccurs="1" minOccurs="0"
                    type="RelationshipAggregationFunction">
                </xsd:element>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
<xsd:simpleType name="ProbType">
    <xsd:restriction base="xsd:double">
```

```
        <xsd:minInclusive value="0"/>
        <xsd:maxInclusive value="1"/>
    </xsd:restriction>
</xsd:simpleType>
<xsd:complexType name="StaticType">
    <xsd:complexContent>
        <xsd:extension base="AggregationFunctionType">
            <xsd:sequence>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
<xsd:complexType name="CPMType">
    <xsd:sequence>
        <xsd:element name="Row" type="CPMRowType"
            maxOccurs="unbounded" minOccurs="1">
        </xsd:element>
    </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="CPMRowType">
    <xsd:sequence>
        <xsd:element name="StateName" type="xsd:string" maxOccurs="1"
            minOccurs="1">
        </xsd:element>
        <xsd:element name="Col" type="xsd:double" maxOccurs="unbounded"
            minOccurs="1">
        </xsd:element>
    </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="MedRelType">
    <xsd:complexContent>
        <xsd:extension base="RelationshipAggregationFunctionType">
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
<xsd:complexType name="MedType">
    <xsd:complexContent>
        <xsd:extension base="AggregationFunctionType">
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
</xsd:schema>
```

Concrete Model

On the top level of the XSD document (*Fig. 72*) the same structure was created as it was already explained for the abstract model (*Fig. 66*).

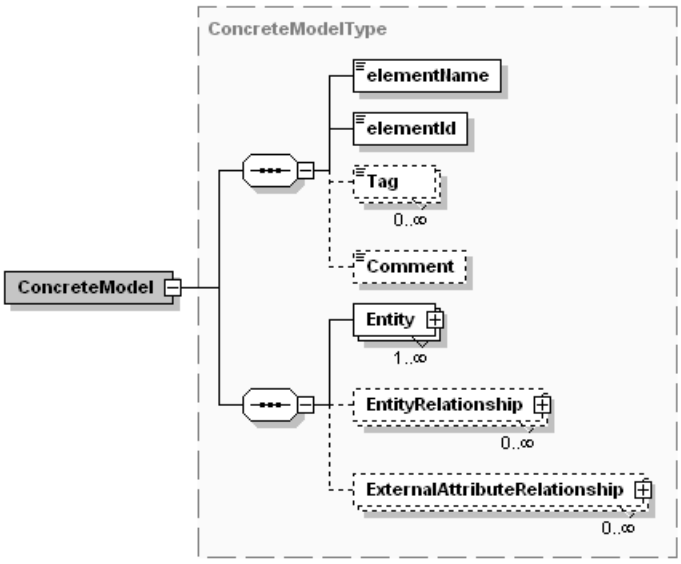


Fig. 72. Concrete Model Overview

The "Entity" element of the concrete model was copied from the abstract equivalent (*Fig. 73*). The ability to save the name of this corresponding object was added.

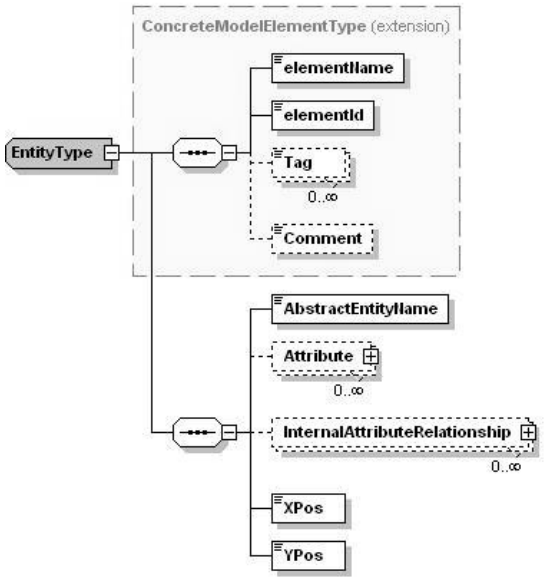


Fig. 73. Concrete Model Entity

Besides element "Name" and "ID", "Tag" and a "Comment", "Calculated Values", a "CPM" and "AttributeEvidence" were defined for the "Attributes" in the concrete model (*Fig. 74*).

A CPM (*Fig. 76*) of an attribute was built up from multiple "Rows". They were described with a "State" and numerous Columns ("Cols"). With this structure, a matrix equivalent was achieved.

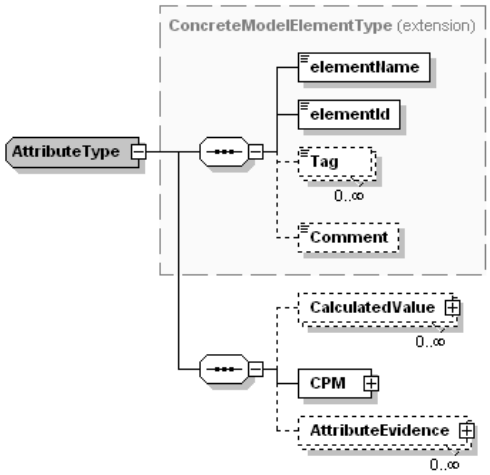


Fig. 74. Concrete Model Attribute

The properties of “Evidence” (*Fig. 75*) were set to be its “Name” and “ID”, “Tag”(s), a “Comment”, the evidence “Source”, a “CPM”, and a “Statement”.

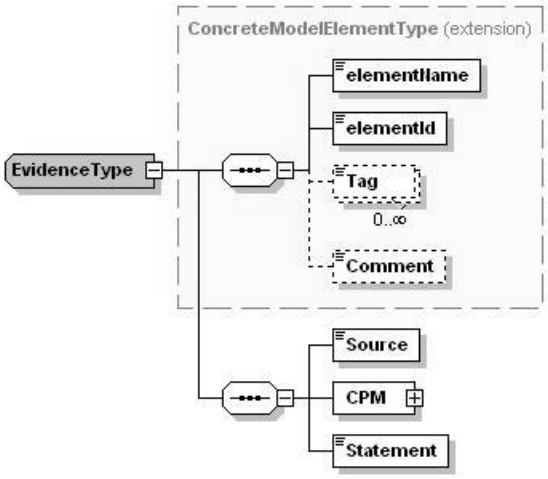


Fig. 75. Concrete Model Evidence

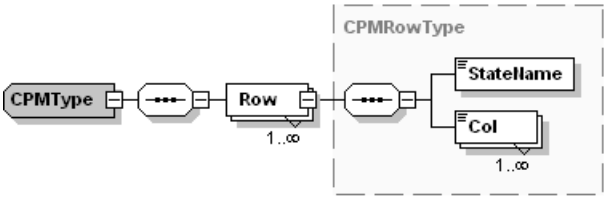


Fig. 76. Concrete Model CPM

The entities in a concrete model were also connected through relationships (*Fig. 77*). The typical properties were added. Additionally Origin- and Entity- ID, numerous external attribute relationships and the ID of the underlying abstract entity relationships can be stored.

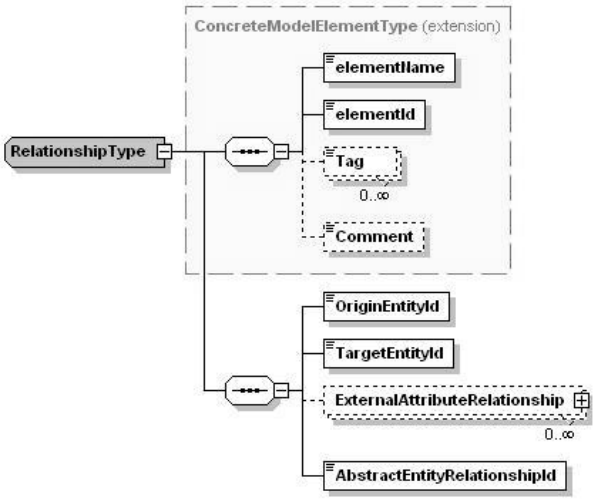


Fig. 77. Concrete Mode Entity Relationship

In comparison to the abstract model XSD, no difference was made between internal and external attribute relationships in the concrete model schema (*Fig. 78*). The "AttributeRelationship" structure was adopted from the "EntityRelationship" setup. Instead of "AttributeRelationships", "EntityRelationships" were stored. In addition, the reference to the abstract model was changed. For an "AttributeRelationship" the ID of the attribute relationship in the abstract model was saved.

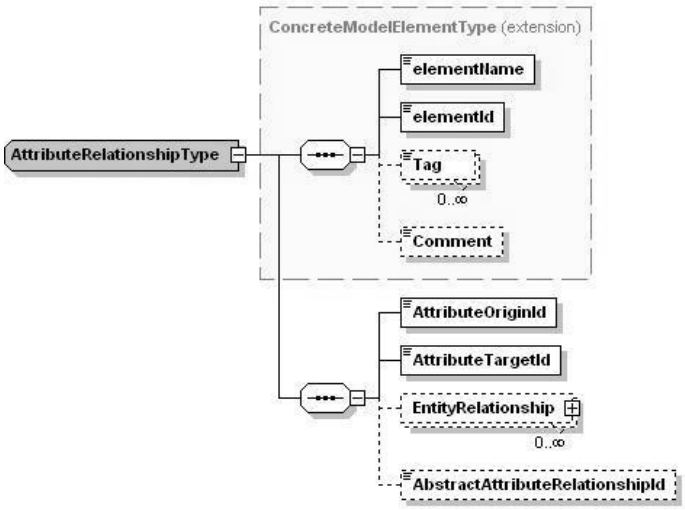


Fig. 78. Concrete Model Attribute Relationship

The XSD in which the described models are contained is presented on the following pages:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    targetNamespace=http://www.ics.kth.se xmlns=http://www.ics.kth.se
    elementFormDefault="qualified">
<xsd:element name="ConcreteModel" type="ConcreteModelType"/>
<xsd:complexType name="ConcreteModelElementType">
    <xsd:sequence>
        <xsd:element minOccurs="1" maxOccurs="1" name="elementName"
            type="xsd:string" />
        <xsd:element minOccurs="1" maxOccurs="1" name="elementId"
            type="xsd:int" />
        <xsd:element name="Tag" type="xsd:string" minOccurs="0"
            maxOccurs="unbounded">
        </xsd:element>
        <xsd:element name="Comment" type="xsd:string" maxOccurs="1"
            minOccurs="0">
        </xsd:element>
    </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="ConcreteModelType">
    <xsd:complexContent mixed="false">
        <xsd:extension base="ConcreteModelElementType">
            <xsd:sequence>
                <xsd:element name="Entity" type="EntityType"
                    minOccurs="1" maxOccurs="unbounded"/>
                <xsd:element name="EntityRelationship"
                    type="RelationshipType" minOccurs="0"
                    maxOccurs="unbounded"/>
                <xsd:element name="ExternalAttributeRelationship"
                    type="AttributeRelationshipType" minOccurs="0"
                    maxOccurs="unbounded"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
<xsd:complexType name="EntityType">
    <xsd:complexContent mixed="false">
        <xsd:extension base="ConcreteModelElementType">
            <xsd:sequence>
                <xsd:element name="AbstractEntityName" type="xsd:string"
                    minOccurs="1" maxOccurs="1"/>
                <xsd:element name="Attribute" type="AttributeType"
                    minOccurs="0" maxOccurs="unbounded"/>
                <xsd:element name="InternalAttributeRelationship"
                    type="AttributeRelationshipType" minOccurs="0"
                    maxOccurs="unbounded"/>
                <xsd:element name="XPos" type="xsd:int" minOccurs="1"
                    maxOccurs="1"/>
                <xsd:element name="YPos" type="xsd:int" minOccurs="1"
                    maxOccurs="1"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
<xsd:complexType name="RelationshipType">
    <xsd:complexContent mixed="false">
```

```
        <xsd:extension base="ConcreteModelElementType">
            <xsd:sequence>
                <xsd:element name="OriginEntityId" type="xsd:int"
                    minOccurs="1" maxOccurs="1" />
                <xsd:element name="TargetEntityId" type="xsd:int"
                    minOccurs="1" maxOccurs="1" />
                <xsd:element name="ExternalAttributeRelationship"
                    type="AttributeRelationshipType" minOccurs="0"
                    maxOccurs="unbounded" />
                <xsd:element name="AbstractEntityRelationshipId"
                    type="xsd:int" minOccurs="1" maxOccurs="1">
                </xsd:element>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
<xsd:complexType name="AttributeRelationshipType">
    <xsd:complexContent mixed="false">
        <xsd:extension base="ConcreteModelElementType">
            <xsd:sequence>
                <xsd:element name="AttributeOriginId" type="xsd:int"
                    minOccurs="1" maxOccurs="1" />
                <xsd:element name="AttributeTargetId" type="xsd:int"
                    minOccurs="1" maxOccurs="1" />
                <xsd:element minOccurs="0" maxOccurs="unbounded"
                    name="EntityRelationship" type="RelationshipType" />
                <xsd:element name="AbstractAttributeRelationshipId"
                    type="xsd:int" maxOccurs="1" minOccurs="0">
                </xsd:element>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
<xsd:complexType name="AttributeType">
    <xsd:complexContent mixed="false">
        <xsd:extension base="ConcreteModelElementType">
            <xsd:sequence>
                <xsd:element name="CalculatedValue"
                    type="CalculatedValueType" minOccurs="0"
                    maxOccurs="unbounded"/>
                <xsd:element name="CPM" type="CPMType" minOccurs="1"
                    maxOccurs="1"/>
                <xsd:element name="AttributeEvidence"
                    type="EvidenceType" minOccurs="0"
                    maxOccurs="unbounded"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
<xsd:complexType name="CalculatedValueType">
    <xsd:sequence>
        <xsd:element name="State" type="xsd:string" minOccurs="1"
            maxOccurs="1"/>
        <xsd:element name="Value" type="ProbType" minOccurs="1"
            maxOccurs="1"/>
    </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="CPMType">
    <xsd:sequence>
        <xsd:element name="Row" type="CPMRowType" minOccurs="1"
```

```
            maxOccurs="unbounded"/>
    </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="CPMRowType">
    <xsd:sequence>
        <xsd:element name="StateName" type="xsd:string" minOccurs="1"
            maxOccurs="1"/>
        <xsd:element name="Col" type="ProbType" minOccurs="1"
            maxOccurs="unbounded"/>
    </xsd:sequence>
</xsd:complexType>
<xsd:simpleType name="ProbType">
    <xsd:restriction base="xsd:double">
        <xsd:minInclusive value="0"/>
        <xsd:maxInclusive value="1"/>
    </xsd:restriction>
</xsd:simpleType>
<xsd:complexType name="EvidenceType">
    <xsd:complexContent mixed="false">
        <xsd:extension base="ConcreteModelElementType">
            <xsd:sequence>
                <xsd:element name="Source" type="xsd:string"
                    minOccurs="1" maxOccurs="1"/>
                <xsd:element name="CPM" type="CPMType" minOccurs="1"
                    maxOccurs="1"/>
                <xsd:element name="Statement" type="xsd:string"
                    minOccurs="1" maxOccurs="1"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
</xsd:schema>
```

Appendix C: Evaluation Table

5 = very good	4 = good	3 = average	2 = bad	1 = very bad	1.5 = n/a = not applicable , likely not available = likely bad

Criteria \ Candidates		Netbeans Visual Library	Qt Jambi	JHotDraw	GEF (SWT)
Drawing and Layout					
	Routingalgorithm	3, existant, performance bugfix pending (problem known)	1.5 = n/a, nothing found, manual effort	2 , rudimental routing (abstract), less effort for implementing own algorithm (at least interface provided)	4, GEF provides "Manhattan", "Default", less effort for Bezier-algo, no performance information
	Built-In Functions	5, exceeding variety of features, see video: http://www.javalobby.org/eps/netbeans_visual_library/	3, little graphic built-in capabilities, more manual work, in comparison to other: quite featurable	4, focus on graphical modelling (framework), weaknesses in beautiful visualisation	4, feasible for modeling (built on SWT functionality -> provides easier access to complex, graphical routines (already implemented)
Graphical appearance					
	Customizability	5, see summary	4, see summary	3, preimplemented Figures, inherit and override for new Figures, see summary	5, see summary
	Feasability of managing Widgets	**The feasability of managing widgets depends on the way of implementation rather then library capabilities**			
Documentation (Learnability)					
	API Doc	5, API Doc: http://bits.netbeans.org/dev/javadoc/org-netbeans-api-visual/org/netbeans/api/visual/widget/doc-files/documentation.html, Java DOC: http://bits.netbeans.org/dev/javadoc/org-netbeans-api-visual/index.html (good)	3, confusing because of spongy borders between Jambi and C++ doc, not as extensive as Netbeans	3, javadoc, well arranged	3, chaotic, API Doc available (in Eclipse API)
	Samples	5, exceeding because of high diffusion rate, http://graph.netbeans.org/examples.html, Tutorial: http://platform.netbeans.org/tutorials/nbm-visual_library.html, Videos, Community	4, available, executable in browser	2, old nearly unused forum at sourcefourge, samples: JHotDraw itself	3, available
General					
	Dependencies (ease of install and diffusion of the final app.)	4, bind libraries in jar File	2, native libraries for each OS	4, pure Java, Code dusting	3, .dll additionally to jars
	Diffusion Rate	5, several partners (amazon web services, eBay, Ricoh, SonyEricsson) -> http://www.netbeans.org/community/partners/ (1.8.07), Use Netbeans in projects: http://www.netbeans.org/about/press/12.html (1.8.07)	2, Qt used in KDE, Opera, Skype, Mathematica, GoogleEarth, VirtualBox (also has several partners http://trolltech.com/company/partner/directory (1.8.08)) but no real Qt Jambi projects	3, UML editors, WFMS, but no well-known apps (http://www.jhotdraw.org/survey/applications.html (1.8.08)), mostly used in academic context for design pattern training	5, delivered with and base of Eclipse
	Maturity (Stability)	4, Shipped with Netbeans 6, v2.0, SUN (experienced), very active development	3, quite new but wrapper for mature c++ libs, trolltech as a GUI specialist	1, 6.0 beta phase, few changing developer (Erich Gamma, Werner Randelshofer), "JHotDraw 7 is currently marked as unstable" (http://sourceforge.net/forum/forum.php?thread_id=2121342&forum_id=39886, 1.8.08), "private" project	4, stable version 3.3, IBM , Eclipse Community
	License	5, CDDL/GPL/LGPL	3, dual license: can be used for open (GPL) as well as closed, but free products. Commercial licence min. 1420 €	5, GPL/LGPL	4, CPL, EPL, Third party components built using GEF are made available under their own licenses

References

1. Architekturwissen, http://www.architektur-wissen.de/etymologie.html accessed in 02/2009

2. Ben-Gal I.: Bayesian Networks. In: Ruggeri F., Faltin F. & Kenett R. (Eds.), Encyclopedia of Statistics in Quality and Reliability, John Wiley & Sons (2007)

3. Boehm, B.W.: A Spiral Model of Software Development and Enhancement. In: IEEE Computer 21(5), pp. 61--72 (1988)

4. Boekhoudt, C.: The Big Bang Theory of IDEs. In: ACM Press, 1(7):74-82 (2003)

5. Buoy UI Toolkit, http://buoy.sourceforge.net/ accessed in 02/2009

6. BusinessDictionary.Com, http://www.businessdictionary.com/definition/scenario.html, accessed in 02/2009

7. Castor Features, http://www.castor.org/features.html accessed in 2009

8. Castor, Quick Description, http://www.castor.org/ accessed in 02/2009

9. Clements, P., Kazman, R., Klein, M.: Evaluating Software Architectures: Methods and Case Studies. Addison-Wesley, 2001

10. Clinger-Cohen Act of 1996 (formerly called Information Technology Management Reform Act (ITMRA)), Division E, National Defense Authorization Act for FY 1996 (P.L. 104-106, February 10, 1996)

11. Daphne Koller's Research Group, Probabilistic Relational Models: http://dags.stanford.edu/PRMs/ accessed in 02/2009

12. Decision Systems Laboratory Wiki: http://genie.sis.pitt.edu/wiki/Introduction:_GeNIe accessed in 02/2009

13. Drobik, A.: Enterprise Architecture: The Business Issues and Drivers. Gartner, Inc. (2002)

14. Druzdzel, M., van der Gaag, L.: Building probabilistic networks: Where do the numbers come from? In: IEEE Transactions on knowledge and data engineering, 12, IEEE, Los Angeles, USA, pp. 289--299 (2000)

15. Eclipse - The Standard Widget Toolkit, http://www.eclipse.org/swt/ accessed in 02/2009

16. Ekstedt, M.: Enterprise Architecture as a Means for IT Management. unpublished, Department of Industrial Information and Control System, KTH, Stockholm

17. Fenton, N., Neil, M.: Combining evidence in risk analysis using Bayesian Networks. In: Safety Critical Systems Club Newsletter 13 (4), pp 8-13 Sept (2004)

18. Fenton, N., Neil, M.: Managing Risk in the Modern World - Applications of Bayesian Networks. A Knowledge Transfer Report from the London Mathematical Society and the Knowledge Transfer Network for Industrial Mathematics, London Mathematical Society,De Morgan House, 57/58 Russell Square London WC1B 4HS (2007)

19. Frechet, M.: Les tableaux de correlation dont les marges et des bornes sont donnees. Annales de l'Universite de Lyon, Sciences Mathematiques et Astronomie, 20:13–31 (1957).

20. GEF Developer FAQ, http://wiki.eclipse.org/index.php/GEF_Developer_FAQ accessed in 02/2009

21. GEF, http://wiki.eclipse.org/GEF accessed in 02/2009

22. GeNIe & SMILE, http://genie.sis.pitt.edu/ accessed in 02/2009

23. Genie Java Tutorial 1: Creating a Bayesian Network: http://genie.sis.pitt.edu/wiki/Java_Tutorials:_Tutorial_1:_Creating_a_Bayesian_Network accessed in 02/2009

24. Genie SMILE Tutorial 1: Creating a Bayesian Network: http://genie.sis.pitt.edu/wiki/SMILE_Tutorial_1:_Creating_a_Bayesian_Network accessed in 02/2009

25. Getoor, L., Friedman, N., Koller, D., and Pfeffer, A.: Learning Probabilistic Relational Models. In: Relational Data Mining, S. Dzeroski and N. Lavrac, Eds, Springer-Verlag (2001)

26. Getoor, L., Friedman, N., Koller, D., and Taskar, B.: Learning probabilistic models of link structure. In: Journal of Machine Learning Research (2002)

27. Good, I., Card, W.: The diagnostic process with special reference to errors. In: Method of Information Medicine, 10(176–188) (1971)

28. Groupable Header Example, http://www.java2s.com/Code/Java/Swing-Components/GroupableGroupHeaderExample.htm accessed in 02/2009

29. Heschl, J.: COBIT in Relation to Other International Standards. In: Information System Control Journal Volume 4. (2004)

30. Hiirsalmi, M.: Method feasibility Study: BayesianNetworks. In: RESEARCH REPORT TTE1-2000-29. Appendix 5, Page 34 (2000)

31. Holschke, O.: Probabilistic Decision Support for Enterprise Architecture Management. unpublished, Insititue for Business Informatics, University of Technology, Berlin (2008)

32. Howard, R.A.: Decision analysis: Practice and promise. In: Management Science Vol. 34, No. 6, pp. 679-695. (1988)

33. International Organization for Standardization/International Electrotechnical Commission, Software engineering -- Product quality -- Part 1: Quality model (2001)

34. Jagt, R. M.: Support for Multiple Cause Diagnosis with Bayesian Networks, unpublished M.Sc. Thesis, Department of Mediamatics, Information Technology and Systems, Delft University of Technology, the Netherlands and Information Sciences Department, University of Pittsburgh, Pitsburgh, PA. (2002)

35. Jensen, Finn V.: An Introduction to Bayesian Networks. Springer, New York, NY (1996).

36. JHotDraw as Open-Source Project, http://www.jhotdraw.org/ accessed in 02/2009

37. JHotDraw unstable, http://sourceforge.net/forum/forum.php?thread_id=2121342&forum_id=39886 accessed in 02/2009

38. Johansson, E.,Johnson P.: Assessment of Enterprise Information Security - Estimating the Credibility of the Results. In: Proceeding of the Symposium on Requirements Engineering for Information Security (SREIS) in the 13th International IEEE Requirements Engineering Conference, 13 Paris, France (2005)

39. Johnson P., Lagerström, R., Närman, P, and Simonsson, M., Extended Influence Diagrams for System Quality Analysis. In: Journal of Software, p.2, III. Influence Diagramms (2007)

40. Johnson, P., and Ekstedt, M.: Enterprise Architecture - Models and Analyses for Information Systems Decision Making. Studentlitteratur, Lund (2007) Chapter 1.3

41. Ibid., Chapter 2

42. Ibid., Chapter 3

43. Ibid., Chapter 5

44. Ibid., p. 28ff

45. Johnson, P., Ekstedt, M., Silva E., and Plazaola L.: Using Enterprise Architecture for CIO Decision-Making: On the imporance of theory. In: Proceedings of the Second Annual Conference on Systems Engineering Research. (2004)

46. Johnson, P., Johansson, E., Sommestad, T., and Ullberg, J.: A Tool for Enterprise Architecture Analysis. In: Enterprise Distributed Object Computing Conference, 2007. EDOC 2007. 11th IEEE International (2007)

47. Johnson, P., Lagerström, R., Närman, P., and Simonsson, M.: Enterprise Architecture Analysis with Extended Influence Diagrams. In: Information Systems Frontiers, Vol.9, Number 2. (2007)

48. Keeney, R., von Winterfeldt, D.: Eliciting Probabilities from Experts in Complex Technical Problems. In: IEEE Transactions on engineering management 38, IEEE, pp. 191--201 (1991)

49. Kjaerulff, U.: Reduction of Computational Complexity in Bayesian Networks through Removal of Weak Dependences. In: Proceedings of the 10th Conference on Uncertainty in Artificial Intelligence, pp. 374-382 (1994)

50. Koller, D.: Probabilistic Relational Models. In: Lecture Notes in Computer Science. Volume 1634/1999, Inductive Logic Programming, Springer Berlin/Heidelberg (1999)

51. Krallmann, H., Schoenherr, M., Trier, M.: Systemanalyse im Unternehmen - Prozessorientierte Methoden der Wirtschaftsinformatik. Oldenbourg, München/Wien, 5.Auflage (2007)

52. Lahres, B., Rayman, G.: Praxisbuch Objektorientierung - Von den Grundlagen zur Umsetzung. Galileo Computing, Bonn (2006)

53. Lange, C., DuBois , B., Chaudron, M., Demeyer, S.: An Experimental Investigation of UML Modeling Conventions. In: O. Nierstrasz et al. (Eds.): MODELS 2006, LNCS 4199, pp. 27–41, 2006. Springer-Verlag Berlin Heidelberg (2006)

54. Lee, D., and Chu, W.W.: Comparative Analysis of Six XML Schema Languages. In: ACM SIGMOD Record (2000)

55. Lehtola, T.: Enterprise Architecture Analysis-Development of a Java based assessment tool, unpublished M.Sc. Thesis, Department of Industrial Information and Control System, KTH, Stockholm (2008)

56. Lillehagen, F., Karlsen, D.: Enterprise Architectures – Survey of Practices and Initiatives. In: Proceedings of the First International Conference on Interoperability of Enterprise Software and Applications. Geneva (2006)

57. Losavio, F., Chirinos, L., Lévy, N., and Ramdane-Cherif A.: Quality Characteristics for Software. In: Journal of Object Technology, vol. 2, no. 2, pp. 133-150. (2003)

58. Ludewig, J.: Modelle im Software Engineering – eine Einführung und Kritik. In: Modellierung 2002, Tutzing (2002) Chapter 5.2

59. MacLean, A., Young, R., Bellotti, V., Moran, T.: Questions, Options, and Criteria: Elements of Design Space Analysis. In: Human-Computer Interaction 6 (3 & 4) (1991)

60. Malveau, R.: Bridging the Gap: Business and Software Architecture, Part 2. (2004), Cutter Consortium, www.cutter.com/research/2004/edge040203.html accessed in 2004

61. Marakas, G. M.: Decision support systems in the twenty-first century. Upper Saddle River, N.J., Prentice Hall (1999)

62. Medvidovic, N., Rosenblum, D., Taylor, R.: A Language and Environment for Architecture-Based Software Development and Evolution. Proceedings of the 21st International Conference on Software Engineering (1999)

63. Mengenshoel, O.J., and Wilkins, D.: Abstraction and aggregation in belief networks. In AAAI97 Workshop on Abstractions, Decisions and Uncertainty. (1997)

64. Meyer, B. Object-Oriented Software Construction, Second Edition. Part B-3 "Modularity", Prentice Hall Professional Technical Reference. ISE Inc., Santa Barbara, (1997)

65. Mintzberg, H.: The structure of organizations: A Synthesis of the Research. Prentice-Hall, Englewood Cliffs, N.J. (1979)

66. Namespaces in XML 1.0 (Second Edition), http://www.w3.org/TR/REC-xml-names/#ns-using accessed in 02/2009

67. Närman P. et al.: Data Collection Prioritization for System Quality Analysis. In: Proceedings of the 2nd International Workshop System Quality and Maintainability (SQM), Elsevier, Athens (2008)

68. Närman, P., Johnson, P., and Nordstrom L.: Enterprise Architecture: A Framework Supporting System Quality Analysis. In: Enterprise Distributed Object Computing Conference, 2007. 11th IEEE International Volume , Issue , 15-19 Oct. Page(s):130 - 130 (2007)

69. NetBeans architecture summary, http://bits.netbeans.org/dev/javadoc/org-netbeans-api-visual/architecture-summary.html accessed in 02/2009

70. NetBeans IDE 6.5 Download page, http://www.netbeans.org/downloads/ accessed in 02/2009

71. NetBeans Partner Program, http://www.netbeans.org/community/partners/ accessed in 02/2009

72. NetBeans Press Release, http://www.netbeans.org/about/press/12.html accessed in 02/2009

73. NetBeans Visual Library Documentation, http://bits.netbeans.org/dev/javadoc/org-netbeans-api-visual/org/netbeans/api/visual/widget/doc-files/documentation.html#Installation accessed in 02/2009

74. NetBeans Visual Library Routing Issue, http://wiki.netbeans.org/GraphLibraryOrthogonalRoutingEnhancementsAPIReview accessed in 02/2009

75. NetBeans Visual Library, http://graph.netbeans.org/ accessed in 02/2009

76. NetBeans, Guided Tour of Subversion, http://www.netbeans.org/kb/60/ide/subversion.html accessed in 02/2009

77. OMG Unified Modeling Language (OMG UML), Infrastructure, V2.1.2, http://www.omg.org/spec/UML/2.1.2/ accessed in 02/2009

78. Online Etymology Dictionary, http://www.etymonline.com/index.php?term=enterprise accessed in 02/2009

79. Pearl, J., Russel S.: Bayesian networks. Report (R-277), November 2000, In: Handbook of Brain Theory and Neural Networks, Arbib M (ed). MIT Press: Cambridge, MA (2001)

80. Pearl, J.: Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference. Morgan Kaufmann, San Mateo, CA. (1988)

81. Pepper, P.: Programmieren lernen. Springer-Verlag (2nd Edition), Berlin, Page 382 (2006)

82. Pereira, C.M., Sousa, P.: A method to define an Enterprise Architecture using the Zachman Framework. In: Proceedings of the 2004 ACM symposium on Applied computing. ACM, Nicosia (2004)

83. Pin-Shan Chen, P.: The Entity-Relationship Model — Toward a Unified View of Data. In: ACM Transactions on Database Systems, 1(1):9-36, March (1976)

84. Qidan, C.: PRM-based multi-relational association rule mining. In: Theses, Dissertations, and other Required Graduate Degree Essays (2007)

85. Qt Jambi Reference Documentation, http://doc.trolltech.com/qtjambi-4.4/html/com/trolltech/qt/qtjambi-index.html accessed in 02/2009

86. Qualiware, http://www.qualiwareinc.com/ accessed in 02/2009

87. Rubinstein, A.: Modeling Bounded Rationality. The MIT Press, Cambridge, Massachusetts (1998)

88. Schoenherr, M.: Towards a common terminology in the discipline of Enterprise Architecture. In: proceedings of the International Conference on Service Oriented Computing (ICSOC) 2008, Sydney, Australia, to be published in Springer 2009

89. Schuster, N., Zimmermann, O., Pautasso, C.: ADkwik: Web 2.0 Collaboration System for Architectural Decision Engineering. In: Proceedings of the Nineteenth International Conference on Software Engineering & Knowledge Engineering (SEKE 2007), KSI (2007)

90. Shannon, C. E.: A mathematical theory of communication. In: Bell System Tech. J., 27:379–423, 623–656 (1948)

91. Simonsson, M., and Johnson, P.: The IT organization modeling and assessment tool: Correlating IT governance maturity with the effect of IT. In: Proceedings of the 41st Hawaii International Conference on System Sciences (2008)

92. Sommerville, I.: Software Engineering. Edition: 8, Addison-Wesley, Longman (2007)

93. Spiegelhalter, David J., Knill-Jones, Robin P.: Statistical and knowledge-based approaches to clinical decision-support systems, with an application in gastroenterology. In: Journal of the Royal Statistical Society, 147, Part 1: pp. 35–‐77 (1984)

94. Stachowiak, H.: Allgemeine Modelltheorie. Springer-Verlag, Wien etc. (1973)

95. Stensmo, M., Sejnowski J. Terrence.: A mixture model diagnosis system. In: Technical Report Series, INC-9401, San Diego (1994)

96. Sun Java AWT API, http://java.sun.com/j2se/1.4.2/docs/api/java/awt/package-summary.html accessed in 02/2009

97. Sun Java Swing API, http://java.sun.com/j2se/1.4.2/docs/api/javax/swing/package-summary.html accessed in 02/2009

98. Svahnberg, M., Wohlin, C., Lundberg, L., Mattsson, M.: A Quality-Driven Decision Support Method for Identifying Software Architecture Candidates. In: International Journal of Software Engineering and Knowledge Management 13(5) (2003)

99. Swing Application Framework, Class "View", https://appframework.dev.java.net/nonav/javadoc/AppFramework-1.03/org/jdesktop/application/View.html accessed in 02/2009

100. Swing Application Framework, Issue 58, https://appframework.dev.java.net/issues/show_bug.cgi?id=58 accessed in 02/2009

101. SwingWT, http://swingwt.sourceforge.net/ accessed in 02/2009

102. Telelogic System Architect, http://www.telelogic.com/products/systemarchitect/ accessed in 02/2009

103. The XML FAQ, http://xml.silmaril.ie/authors/dtds/ accessed in 02/2009

104. Thomas, D., interviewed by Bill Venners (2003-10-10): Orthogonality and the DRY Principle. http://www.artima.com/intv/dryP.html accessed in 02/2009

105. Tyree, J., and Akerman, A.: Architecture Decisions: Demystifying Architecture. In: Capital One Financial, IEEE (2005)

106. Vaughan-Nichols, S. J.: The Battle over the Universal Java IDE. In: IEEE Computer, Volume 36 Issue 4. pp. 21 - 23. ACM Press (2003)

107. Wellman, M.P., Liu, C.: State-space abstraction for anytime evaluation of probabilistic networks. In: Proceedings of the Tenth Conference on Uncertainty in Artificial Intelligence (1994)

108. Würthinger, T.: Visualization of Program Dependence Graphs. unpublished M.Sc. Thesis, Institute for System Software, Johannes Kepler University, Linz (2007)

109. XML Editors Review, http://www.cmsreview.com/XML/Editors/index.2.en.html accessed in 02/2009

110. XML Schema Requirements, http://www.w3.org/TR/1999/NOTE-xml-schema-req-19990215#Purpose accessed in 02/2009

111. Younes, H.Y.: Current tools for assisting intelligent agents in realtime decision making. unpublished M.Sc. Thesis, Department of Industrial Information and Control System, KTH, Stockholm, Chapter 3.4.1, Page 16 ff (1998)

112. Zachman, J.A.: A Framework for Information Systems Architecture. IBM Systems Journal, 26: pp. 276--292 (1987)

113. Zaval Light-Weight Visual Component Library (GUI Designer Package), http://lwvcl.com/gdp.php accessed in 02/2009

114. Zimmermann, O., et al.: Reusable architectural decision models for enterprise application development. In: Quality of Software Architecture (QoSA), Boston, USA (2007)