NTNU
Innovation and Creativity

Model Driven Enterprise Analysis

A model-driven tool-assisted process for criticality and availability analysis of enterprise systems

Thomas Hermansen

Master of Science in Computer Science
Submission date: June 2006
Supervisor: Tor Stålhane, IDI

Norwegian University of Science and Technology
Department of Computer and Information Science

Problem Description

When analyzing a system, one uses an analysis method and take basis in a representation of the system. The aim of this Master thesis is to evaluate which configurations of method and representation that best lets us assess the reliability of business critical enterprise systems. The methods are taken from the field of System Safety, and are altered if needed. The representations are taken from well used notations used in IT, such as UML. Further more, an extension to the RUP framework that facilitates a complete process for identifying and treating these reliability problems is to be created. This process should implement a selection of the configurations mentioned above. Lastly, a set of tools which automates and makes this extended process more efficient is proposed and implemented. Examples are tools that administrate and automate activities in the extended process, and tools that assist in the analysis, making it more thorough.

Assignment given: 20. January 2006
Supervisor: Tor Stålhane, IDI

Abstract

Today more and more companies acquire enterprise-scale solutions for their organization. Enterprise-scale solutions connect departments and business functions in the organization in order to facilitate the coordination, communication, and work flow between them. However, when systems get more interconnected and complex, they are also more prone to faults. If business critical parts of the system are affected, this can be devastating for a company.

When designing large enterprise scale systems, one uses a wide range of specialized models with different view points and applications. This fragmentation and specialization of the representation of the system decreases the clarity of the total enterprise model and implies that it is difficult to analyze the enterprise as a whole. To overcome this problem, specialized software tools that can integrate the sub models in a total model can be developed.

This thesis will develop a tool assisted extended process to the development process Rational Unified Process that helps analyzing the design of enterprise solutions by integrating the behavioral and structural models of the system into a unified representation.

The tools take basis in digitized models represented in UML, the industry standard language for modeling software systems. We will focus on the two quality attributes availability and criticality.

Preface

This report is part of a master thesis carried out at the Department of Computer and Information Science at the Norwegian University of Science and Technology (NTNU) during the spring semester of 2006.

I would like to thank my supervisor Dr. Tor Stålhane for feedback and for allowing me to define my own project within the frames of his associated research group.

Trondheim, June 16th 2006.

Thomas Hermansen

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Terminology

1.1.1 System Safety

The field of *System Safety* is a set of methods and practices to assess the risks connected with the development and execution of a system, so that we can take appropriate measures to minimize these risks. System Safety is about *what* can go wrong and *how* it can go wrong. For an introduction to the concept of risk, see Appendix B.

1.1.2 Rational Unified Process

The *Rational Unified Process (RUP)*[1] is a process framework developed by Rational Software, now fully owned by IBM. A process framework is a systematic description of the process of software development, defining goals, tasks, and plans and distributing work among the participants in the project. *RUP* lets the user configure a custom process of development which is tailored to the projects needs based on proven best practices. RUP provides a web-based electronic process guide that leads the whole development team through the development process. Extensions to this electronic process guide can be created by developing software addons called RUP plug-ins.

RUP is based upon the idea of iterative development, as apposed to the traditional Waterfall process (See Appendix C). The main focus of RUP is the requirements and the architecture of the system(s).

[1] http://www.rational.com/products/rup/

1.1.3 Enterprise systems

Enterprise systems are systems that integrate the business functions in an organization to enhance the efficiency and the flexibility of the organization. Enterprise systems realize business processes, which are the main activities of a business.

1.1.4 Business critical systems

A business critical system is a system on which the organization is fully reliant to conduct its business. A breakdown in a business critical system will halt the business itself.

1.1.5 Unified Modeling Language (UML)

The Unified Modeling Language (UML) is a general purpose language for modeling object oriented systems, [13].

1.1.6 UML Profiles

Since the UML language is a general purpose modeling language and has been designed to model a wide range of object oriented systems, we sometimes need a more specialized language to model special features of a domain. One solution to this problem is to extend the UML language by creating a UML Profile. A UML Profile does not change the structure of the UML language, but it introduces three extension mechanisms: stereotypes, tagged values, and constraints, [24].

Figure 1.1 shows a UML description of the operation "fetching data from a database" in the base UML language, as well as a UML Profile which has been created to capture this interaction more precisely.

Stereotypes are extensions of the constructs in the UML language. You can take the modeling construct of *Subsystem* in standard UML and extend it to *Database* or *Client*, or you can take the standard UML connector *Uses* and extend it by the more precise relation *GetsDataFrom*.

Tagged values are pairs of names and values that stereotypes can take. In Figure 1.1, the "Client" stereotype has a tagged value named *OS* with value "Windows" indicating that the client runs on the Windows OS.

Constraints are defined using the Object Constraint Language (OCL). They define a set of rules which describes what a well-formed or legal model is.

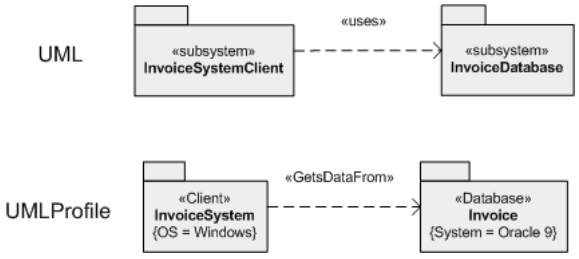

Figure 1.1: UML Profile Example.

1.2 Motivation

Modern enterprise systems bring great benefits to the companies that acquire them. Yet, the interconnectedness of the systems makes the business more vulnerable. If business critical parts of the system are crippled, this can be devastating for a company. Business critical parts are those parts of the system that are imperative for the day-to-day run of the business and help the business conduct its primary functions. The consequence of a fault in a business critical function can be loss of profit, loss of clients, and even loss of the business itself.

To minimize the vulnerability of the system during operation, it is imperative that we locate these vulnerabilities at the design stage of the project, so that we can protect ourselves against them. This can be achieved by building and analyzing models.

In the early days of system design it was easy to analyze a system based on a small set of models. Because of the complexity of the modern enterprise systems, the fact that they are technically complex, and that they incorporate socio-technical aspects, developers have to use a wide range of specialized models. This makes it very hard to analyze the enterprise because the description (models) are fragmented and hard to interrelate. This has led to the need of software tools specialized to integrate the different sub models, thereby providing a unified view of the enterprise as a whole.

1.3 Problem description

The aim of this work is to create tools and work process that can assist in the analysis of an enterprise solution. The idea is found in Figure 1.2. Using the process and a set of tools that take models as input, automatic or semi-automatic analysis of the enterprise is performed. This will be carried out using the XMI standard. XMI is a flat file format for the storage of

UML models based on XML, [5], [12].

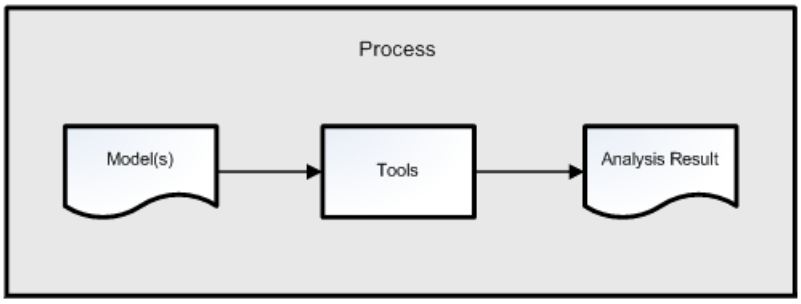

Figure 1.2: The aim of the process.

I will develop two software tools. The first is called the *Enterprise Analyzer*. It takes all the enterprise models (both behavioral and structural) and merges them into an internal representation of the enterprise. The result will be a report of the analysis of the enterprise. The tool will focus on the two quality attributes *availability* and *criticality*.

The second tool is called the *Model Analyzer*. It takes a single model as input and transforms it into a list of risks that threaten the enterprise. The users should be able to create their own transformation logic. While the Enterprise Analyzer analyzes the whole enterprise, the Model Analyzer analyzes specific parts of the system. The Model Analyzer will also have the ability to store the risks it uncovers in a database. The tools will be developed using C# 2.0 on .NET platform (v2.0) with Visual Studio 2005.

Then we define how the enterprise should be modeled. A set of UML Profiles is created to precisely model the different parts of the enterprise. We call these the *Enterprise UML Profiles*.

Also, a set of transformation profiles need to be defined. A transformation profile is a piece of logic that describes how the Model Analyzer turns a UML model into a set of risks. We must create transformation logic that turns models described by the Enterprise UML Profiles into risks concerning the enterprise.

I will also create a process extension to the Rational Unified Process (RUP). For a more detailed description of RUP, see Appendix C.

1.4 Related work

Skene and Emmerich, [19], argues that Model Driven Development (MDD) is a suitable framework for integrating analysis and design. MDD is a concept where the design and development of applications are almost completely done by creating models that is transformed into code. They saw the possibility of integrating the analysis directly into the modeling tools. Their focus was on performance, which is only loosely and indirectly related to reliability. Performance is a measure of the ability to handle work load.

Barth, [20], introduced another approach to performance analysis of computer systems represented in UML models and developed a prototype tool that simulated the performance of the system.

Earlier there has been a myriad of papers discussing the use of models as basis for automatic performance analysis. Balsamo and Simeoni, [17], gives a review of many of these approaches.

Majzik, Pataricza, and Bondavalli, [25], describe a transformation that does a dependability analysis of a structural UML diagram.

Rodrigues and Rosenblum, [18], proposed a UML Profile for model driven reliability analysis of general computer systems. They have not developed any tools, but rather used an XLS[2] transformation to turn the XMI models into the input of the LTSA tool (Labelled Transition Systems Analyzer) which is a tool developed by [16] for verification of concurrent systems.

Earlier work seems to be mostly related to performance analysis, and to some degree to reliability analysis. I emphasize two other quality attributes, namely availability and criticality. Most of the work seems to range from conceptual to early prototypes. I have developed tools that can be used within the RUP framework that covers both the behavioral and structural aspect of the enterprise.

1.5 Report outline

In chapter 3 we define an Enterprise Representation and the models needed to create this representation.

The two tools, the Enterprise Analyzer and the Model Analyzer, will be described in Chapter 4 and 5 respectively.

Chapter 6 describes the Enterprise UML Profiles.

[2] An XML based language for transforming an XML file into another textual representation

Chapter 7 defines the Transformation Profiles that the Model Analyzer uses to transform the models into risk lists.

In Chapter 8 an extension to the RUP Process is created to facilitate the use of the tools in the development of enterprise systems.

Chapter 9 describes a test of the tools using an example enterprise modeled with the Enterprise UML Profile.

Chapter 10 contains the tools' user manuals.

A discussion of the work is found in Chapter 11.

Chapter 12 gives the conclusions and proposes future work.

Appendix A contains the class diagrams of the tools, as well as an overview of what each source file in the tool does.

Appendix B gives an introduction to the concept of risk and risk analysis.

Appendix C gives an introduction to the Rational Unified Process.

Appendix D contains a description of the two types of risk analysis that the Model Analyzer performs.

Appendix E shows the contents of the ZIP file attached to this report.

Chapter 2

Introduction to the Tools

2.1 Quality attributes

When developing an enterprise system, it is helpful to define a set of criteria to measure the performance. Two such criteria, availability and criticality, are defined in the following and are the focus of my tools.

2.1.1 Availability

The purpose of the enterprise systems is to assist and perform business functions, which in turn run the enterprise. The first identifiable goals in the engineering of an enterprise system is to increase the availability of these business functions. The enterprise should have some kind of availability level goal, such as "The online store should never be unavailable more than 1 minute a day on average".

If the developers are to design an enterprise system that meets a set of these goals, they must have some way of calculating this availability. Having a high availability implies that the systems, components, infrastructure etc in the enterprise are reliable. To identify the reliability issues, we need to perform risk analysis.

According to Musa [9] reliability of a software component is

"The probability of execution without failure for some specified interval of time or other natural units."

E.g, a component can have reliability of 0.9 (90 percent chance) of not failing within one year of operation.

A common way to define reliability is simply as the time between failures, Mean Time Before Failure (MTBF), which is the average time between failures.

In order to have a high availability, we also need high maintainability. Once an enterprise function is down, we must restore it as quickly as possible. Mean Time To Repair (MTTR), which is the average time it takes to repair the component, is a measure of the maintainability of the component.

With these two factors we can define availability as

$$Availibility = \frac{MTBF}{MTBF + MTTR}. \tag{2.1}$$

As seen, the availability is not the same as the reliability. The availability is a function of reliability and maintainability. If we speed up the time it takes to repair a component, it does not make the component more reliable (the time between failures does not increase), but the availability of the component is increased.

Assume that a system is composed of several components that are used to realize the system functions. The system function can rely on the components in two different ways.

The components can be arranged in an AND-configuration, which means that the function uses a series of components that all need to be successful for the function itself to be successful. The availability of such a function is defined in (2.2), [8].

$$availibility = \prod_{i=0}^{n} A_i. \tag{2.2}$$

The overall reliability of an AND-configuration with n components is the product of the availability of each component.

An example is a process that needs to gather data from two different databases and calculate some value. For the process to succeed, the two databases must be operational, as well as the component that performs the calculation. In other words, we have three components that need to succeed. Assume that all the components have an availability of 0.99. Then the overall availability of the process is $0.99 \cdot 0.99 \cdot 0.99 = 0.970299$.

The other arrangement of components is the OR-configuration, where a function uses a series of components in which at least one of them must succeed for the whole process to succeed. The formula for calculating the availability of such a configuration is shown in (2.3), [8].

$$availibility = 1 - \prod_{i=0}^{n}(1 - A_i). \tag{2.3}$$

One example is a component with the availability of 0.99 that does a calculation based on the data in a database. The data exists in two parallel databases, both of which have an availability of 0.99. The overall availability of the process is then $0.99 \cdot (1 - ((1 - 0.99) \cdot (1 - 0.99))) = 0.989901$.

When developing an enterprise system, the reliability and maintainability are not hard to find. Enterprise systems uses to a large extent OTS[1] products where the vendors supply the reliability information. The same applies for computer hardware.

When determining the reliability of in-house developed software, there exists thumb rules and mathematical models for calculating this based on inputs like number of statements, numbers of conditions and the experience of the programmer, code coverage etc.

The time to repair any failures may be determined by looking at the enterprises unique situation, such as accessibility to support, technicians etc.

The companies also usually have legacy systems that they want to integrate, and know themselves what the reliability and maintainability of these systems are.

2.1.2 Criticality

Knowing the availability of a certain part of the enterprise is not enough. The more critical a business function is, the higher level of availability is needed. Some business processes will be more critical than other, and the components realizing a process will inherit this criticality. When doing risk analysis we should have a clear picture of which components that are more critical than others, so that we can focus more time on them.

A good measure of criticality would be the amount of profit a company misses if the business process is down, plus the amount of money it costs, e.g in the form of the loss of clients etc.

The two main performance indicators of the enterprise system analysis will be the *availability* and the *criticality*. The criticality can be viewed as the processes' influence on the structural parts of the enterprise, while the availability can be viewed as the structural parts influence on the processes.

[1] Off The Shelf - A piece of software acquired externally. Either commercially (COTS) or with an Open Source license.

Further, one can see that it is availability and criticality that together gives the most precise answer to the question about where it is best to concentrate resources in development. Knowing the criticality of the enterprise is then important when designing a reliable enterprise system.

2.2 The tools relation to quality attributes

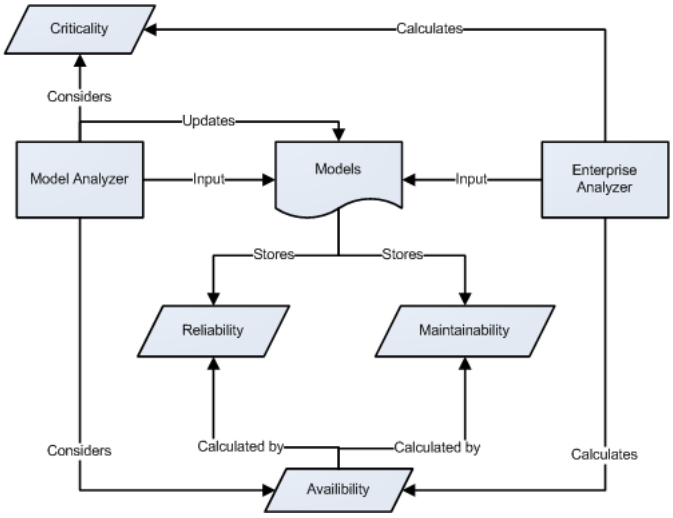

Figure 2.1: The proposed tools.

Figure 2.1 shows the relation between the two quality attributes (availability and criticality) of the extended process and the tools.

When a part of the enterprise is modeled, it is analyzed using the Model Analyzer. Based on the risk analysis assisted by this tool, the analyst updates the models' reliability and maintainability information if needed (*MTBF* and *MTTR* tag values). The general design is altered to reduce any risks that the analysis has uncovered.

The model is then fed into the Enterprise Analyzer which appends it to the current representation of the enterprise. One can then use the tool to calculate the criticality and availability of the updated enterprise. This information can also be considered when using the Model Analyzer.

The Model Analyzer helps the modelers create more reliable solutions, whilst the Enterprise Analyzer helps these same modelers to verify that the enterprise meets the availability goals and to see which parts of the enterprise that are most critical.

2.3 Tool requirements

Given that the tools are to be used in connection with the RUP process, it enforces certain requirements or constraint on how the extended process and tools should function. With basis in the best practices of RUP, I will now lay down these requirements.

- Develop iteratively.

The process should allow iterative development. This means that the practices forced by the extended process should not conflict with the overall RUP practice of iterative development.

- Use component architecture.

The enterprise software should be modeled with a component-based view.

- Model visually.

The design of the enterprise should be done as visually as possible. Since the system is modeled using UML models to describe every facet of the enterprise, this is already achieved.

- Continuously verify quality.

Quality, in view of our tools and extended process, is criticality and availability. As the enterprise design and development is progressing, we should be able to verify the criticality and availability of the enterprise.

We sum up the requirements in Table 2.1.

REQ	Description
R1	The tools should allow componential and iterative development of the enterprise.
R2	The tools should efficiently analyze the criticality of the enterprise
R3	The tools should efficiently analyze the availability of the enterprise
R4	The analyzers should be able to verify the availability and criticality of the enterprise whilst it is developed.

Table 2.1: Tool Requirements.

Chapter 3

The Enterprise Representation

3.1 Introduction

Models can be said to have a *level* and a *view*. The level is level of detail of the model. For example system level or component level. The view is what the model tries to represent, such as behavior or structure.

A business analyst might design the business processes, one architect might model the behaviorial aspect of the systems in the enterprise, whilst another might model the structural aspect of a sub system. Each models a piece of the enterprise. The problem arrives when it is time to analyze the enterprise. The enterprise representation is fragmented and it is not easy to see the relation between the different parts of the enterprise. It is very hard or near impossible to do a full analysis of the system. Also, few, if any persons have knowledge about every facet of the enterprise.

One way to reduce the fragmentation is to define a model that encompasses the whole enterprise at every view and level. Such a model would not only be very large and difficult to read, but also hard to analyze from a human perspective. Fragmentation leads to more precise models, but it decreases our ability to analyze the enterprise.

Defragmentation leads to more easily analyzable models, but makes it harder to fully model a part of the enterprise. If we have to omit details of the enterprise, these details will not be analyzed during the model-based analysis.

We need to find a way where we can model at different levels and views, that is, have a fragmented representation, without the loss of clarity.

The solution to this problem is to merge these digitized models (XMI stored models) into an all-encompassing model that is analyzed by a software tool. See Figure 3.1.

This unified model is from now referred to as the *Enterprise Representation*. It is a data structure used by Enterprise Analyzer tool.

As the system is gradually designed, the models are parsed into this data structure, thus growing a representation of the enterprise. The business analyst can at any time review the availability of the business processes based on the models supplied by the architects and, likewise, the architects can review the criticality of the different parts in the system based on the models that the business analyst created. The analyst can see if the current solution meets the availability requirements, and the architects can identify weak spots.

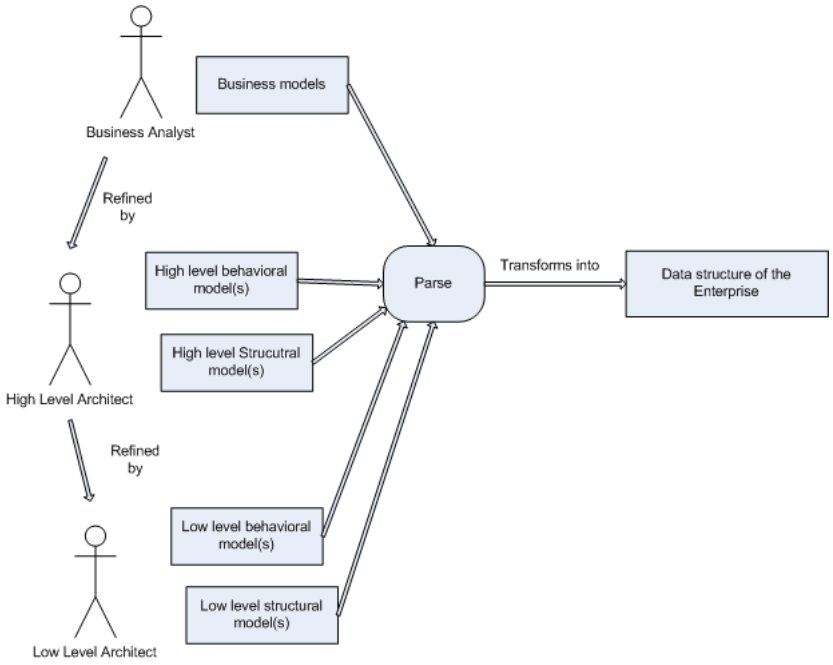

Figure 3.1: Creating the Enterprise Representation.

This chapter will define a representation of the enterprise and select a set of models that lets us model this representation.

3.1.1 The nature of Enterprise Systems

In [3] we see that the typical Enterprise Systems are net centric, that is, they use a client-server architecture. This is logical as many users need to access the same information in the enterprise system. By storing the data centrally, it is easily accessed by users across the organization. Not only over LAN, but also over the internet, as organization can span over large geographic distances. Enterprise solutions can also connect to other organizations for business to business commerce and collaboration. Enterprise systems are therefore highly distributed systems with a high degree of information flow and many transactions.

A study of large companies using enterprise systems showed that 70-80 percent of all transactions was made in legacy[1] systems, often developed in old languages like COBOL[2], which means that a lot of the effort in the development of an enterprise system is connecting old systems with new, [4]. This implies that the component based architecture of RUP is justified in this domain.

3.2 Exploring the enterprise

In order to automatically analyze the enterprise, we need to define its structure. In other words we must create a meta model of the enterprise domain. A blue print in which we can define a wide range of different enterprises that all conform to the same meta model.

In Figure 3.2 such a meta model is defined. The model was created by doing a top down decomposition of the enterprise.

As said, the role of the enterprise system is to realize one or more business processes. Such a process might be “Sell an item”. The business process is in turn realized by a set of business functions. A business function is a task performed by the enterprise. In the example of the business process “Sell an item”, a business function might be “Accept order” and “Handle Order”.

The business functions are realized by a set of systems which support these functions. Systems such as “Invoice-database” and “Order-database” Systems are in turn realized by a set of components. A Software component may be a class or a set of classes that perform a special function such as

[1] Computer systems or application programs which are outdated and incompatible with other systems, but are too costly to replace or redesign. They are often large, intimidating, and difficult to modify. -http://www.rigi.csc.uvic.ca/Pages/description/glossary.html

[2] Common Business Oriented Language. Programming language developed in the 1960s for the creation of financial and administrational software.

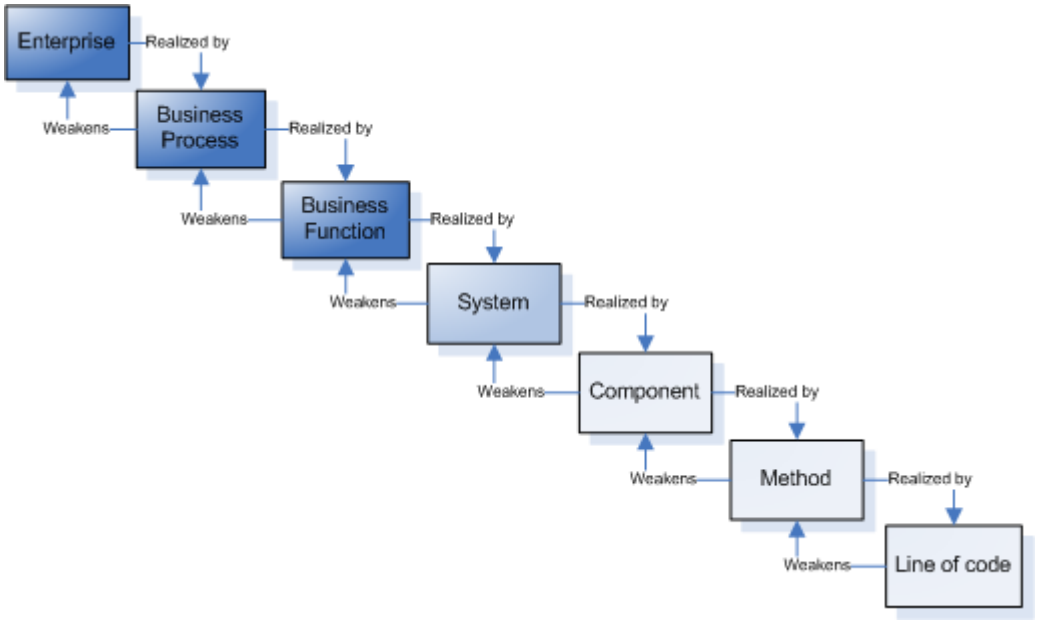

Figure 3.2: Enterprise meta model.

a “Database Handler” component which handles communication with the database. Each component is realized by a set of methods, which in turn are realized by a piece of code.

This decomposition shows the connection between the socio-technical construct of a business function to the technical construct of a line of code.

Each construct in this model weakens its super construct, and weaknesses are passed upwards in the model. A weak line of code could result in the weakening of the entire enterprise. By having this traceable link we may easily analyze the enterprise. For example, if we know what business functions that are critical to the enterprise, we can easily trace this criticality down to component level, and therefore know which components we need to spend more time on (both in analysis and development).

In Figure 3.2 we can see that there exists three distinct levels of modeling and analysis. These three levels are shown in Figure 3.3. The upper level deals with the business processes of the enterprise, the middle level deals with how the systems realize these processes, and the lower level deals with the inner workings of a single system.

At the start of the project, we don’t know anything about the architecture, we only have set of requirements which defines a set of business processes. The next step is to realize these business processes with a set of systems.

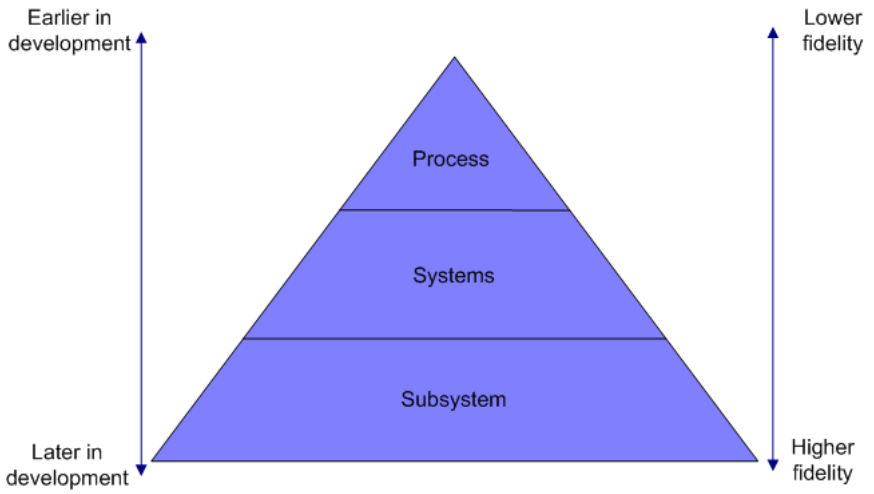

Figure 3.3: The enterprise pyramid structure.

As we pass along the time line of the project, we go from conceptual to real, from socio-technical to purely technical. The design becomes more and more detailed, and the amount time and resources needed to analyze them increases. Since every level is important for the overall well-functioning of the enterprise, we cannot ignore the lower levels. This produces the unwanted effect of ever increasing analysis work.

As previously mentioned, a construct weakens it's super-construct. Another way to view this is that sub-construct inherits the criticality of its super-construct. For example a critical system must have a critical component, and a non critical system cannot have critical components.

By moving downwards in the pyramid, we can filter out non critical aspects from our analysis, so that even if the fidelity of the design increases, the work of analysis does not.

The nature of criticalness is simple. A critical business process is realized by critical systems, which uses critical components. The criticalness is inherited downwards in the hierarchy (Figure 3.2).

Capturing this connection on the other hand is not simple. There is no diagram that captures all aspects of the system. Even if we had this, such a diagram would be confusing to both make and read. Diagrams focus on different views and levels of detail. A component would be depicted at a static low level logical diagram, while a business process would be depicted in a behavioral high level process diagram. In other words, we must use several kinds of models to model the diagram and their connection needs

to be created and maintained outside the models. The Enterprise Analyzer could then take these models as input, assemble an internal representation of the system, and assist us in analyzing and visualizing the criticality of the system.

Such a tool will become useful when a system grows so large that one person cannot easily see how the criticality expands into the system. A business analyst knows which business functions that are critical for the enterprise, but does not have knowledge about which system components that are critical in realizing these processes. On the other side, you may have a developer that knows what components are critical to the task that they are realizing, but they do not know how to relate this criticality to the business processes. The first step in creating such a tool is to define an enterprise representation.

Based on the structure in Figure 3.3 and Figure 3.2, we can define a representation of the enterprise (Figure 3.16).

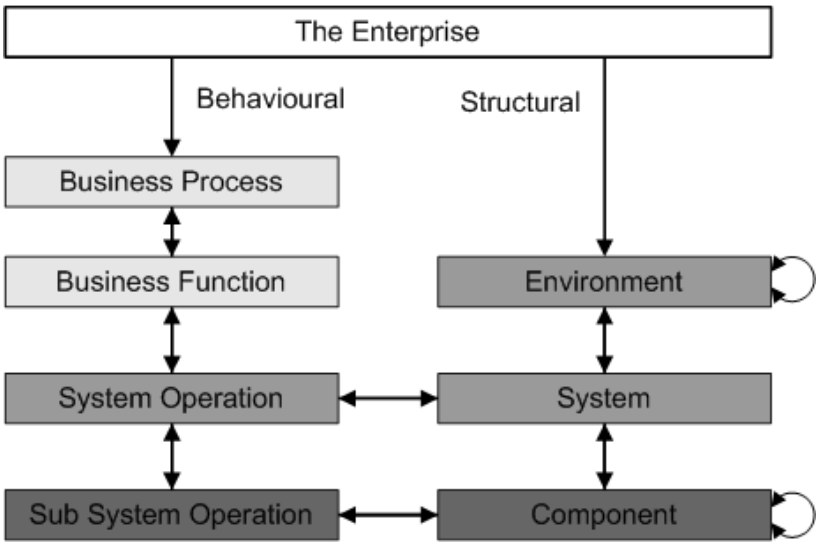

Figure 3.4: The structure of the enterprise.

The three layers in the pyramid of Figure 3.3 are shown with three different shades of grey in the figure. The enterprise is also divided into two concerns, a behavioral and a structural concern. The entities at the left side constitute the behavioral parts of the system, whilst the right side in the figure constitutes the structural parts.

The behavior of the enterprise is divided into *business processes*, which themselves are realized by a set of *business functions*. Each business function is divided into *system operations*, which are interactions between systems or a highly abstracted operation within a single system.

System operations in turn are divided into *subsystem operations*, which are the interactions between components in the system(s).

The structure of the enterprise starts with the *environment*, which is the environment in which the system runs. It can be physical, like a server, or logical, like an operating system. Environments themselves consist of other environments or *systems*. Systems are distinct parts of the enterprise that offer functionality to the business functions. The Systems are divided into *components*, which themselves can be divided into one or more components.

To complete the representation we need to link the behavioral and structural parts. This is done by linking each system operation with its associated systems and, likewise, linking subsystems to its associated components.

We now have the structure of the enterprise. The next step is to select a set of models that lets us model something that can be transformed into this representation.

3.3 Model selection

In the previous section it was established that the enterprise representation should have a three-layered nature. One is in the business process domain, largely dealing with the business processes. The second concentrates on the systems and their interplay. The third focuses on the components of the systems and their interplay.

3.3.1 Business processes

At this level we need to capture the business process and the business functions. The entities of this level of modeling are business functions, and a single model should represent a single business process.

The business functions transcend the technical aspect of the systems and is more than a sequence of system operations. For example, the activity of selling an item from an online store may span over several weeks, from ordering the item, storing it, selling it, and dispatching it.

At this stage, we don't aim to find out how one system in the process may fail, but rather how the process itself might fail. To assess the business process, we must effectively model it conceptually. First it must support parallelism, since business processes are often divided into parallel sub-activities.

The entities of the model must be able to represent business functions, since

this is the main focus of business process modeling. The connectors in the model must represent the transition from one business function to another. Furthermore, the model must be able to handle conditional forks, as each process can be divided into several scenarios.

There are five behavioral UML models: Communication-, Interaction-, State Machine-, Sequence-, and the Activity Diagram. Based on the requirements above, we can compare all of the behavioral diagrams (see Figure 3.5).

	Parallelism	Entities as activies	Conditional	Activity transitions
Use Case Diagram	N	Y	N	N
State Machine Diagram	N	Y	Y	Y
Activity Diagram	Y	Y	Y	Y
Sequence Diagram	Y	N	Y	N
Communication Diagram	N	N	N	N
Timing Diagram	Y	N	N	N

Figure 3.5: Comparing the diagram types on business process modeling.

Only the Activity Diagram satisfies all our needs to model business processes. Figure 3.6 shows such a diagram applied to the modeling of a business process. The figure also points out the different parts the activity diagram is decomposed into. The *action states* represent business functions, the *transition* show the sequence of the business functions. The *decision* lets us branch out into several scenarios based on a decision. The *fork* and *merge* let us model parallel activities.

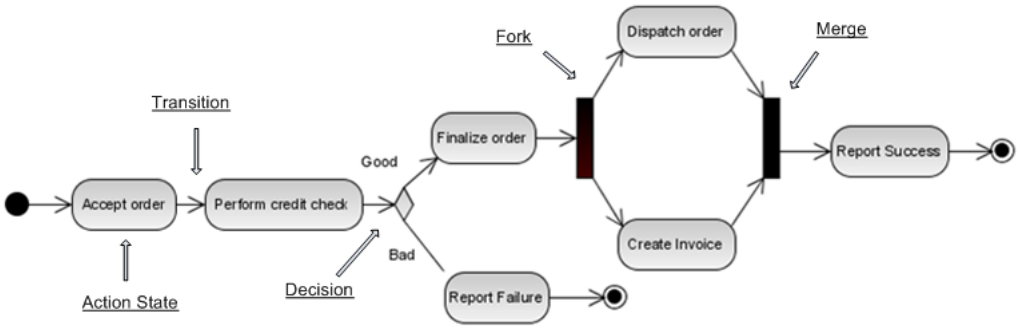

Figure 3.6: An example Activity Diagram.

3.3.2 Systems

When realizing a business function, the systems in the enterprise cooperate. Information will flow from department to department. The goal is to capture this interplay of entities in the enterprise. at this level we need to model the systems, the environments in which the systems run, and the interaction between these parts.

Beginning with behavioral part, the entities must be able to depict a system, and the connectors must depict the communication (data retrieval, messaging etc.) between these systems. The model should also be able to depict the sequence of actions, so that we may easily see the consequences of a failed action. Based on these requirements, we can compare all of the behavioral diagrams (see Figure 3.5).

Only the sequence diagram satisfies all our needs to model the behavioral interactions of the Systems level. Figure 3.8 depicts such a diagram. The sequence diagrams are composed of *life lines*, which are structural parts of the system (in our case whole Systems), and *messages*, which are the interactions between these (in our case System Actions).

On the structural side, we need to represent systems and environments. We can define the following requirements: It should show the mapping of the system to hardware and other software (the environments), and its entities should be systems.

In Figure 3.9, all the structural diagrams are compared w.r.t these two requirements. Only the Deployment Diagram satisfies these two requirements. Figure 3.10 shows such a diagram applied to our example. The *nodes* represent environments, while *components* represent systems.

	Messeging	Sequence	Entities as systems	Entities as systems
Use Case Diagram	N	N	N	N
State Machine Diagram	N	N	N	N
Activity Diagram	N	Y	N	N
Sequence Diagram	Y	Y	Y	Y
Communication Diagram	Y	N	Y	Y
Timing Diagram	N	N	N	N

Figure 3.7: Comparing the diagram types on systems behavioral modeling.

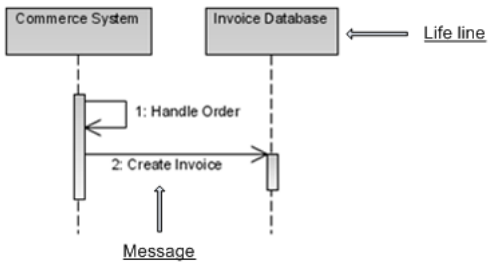

Figure 3.8: An example Sequence Diagram.

	Map software to hardware	Entities as subsystems
Class Diagram	N	N
Component Diagram	N	Y
Composite Structure Diagram	N	N
Deployment Diagram	Y	Y
Object Diagram	N	N
Package Diagram	N	N

Figure 3.9: Comparing the diagram types on systems structural modeling.

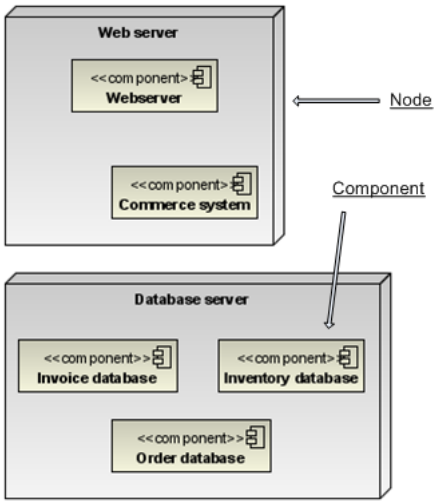

Figure 3.10: Example Deployment Diagram.

3.3.3 Subsystems

When analyzing the subsystem part of the system, we can uncover risks concerning components and the interaction between them. The entities in the subsystem domain are components.

In the structural part of the layer, the model should depict the logical structure of components and the relation between them.

	Components as entities	Show interfaces
Class Diagram	N	N
Component Diagram	Y	Y
Composite Structure Diagram	N	Y
Deployment Diagram	Y	N
Object Diagram	N	N
Package Diagram	N	N

Figure 3.11: Comparing the diagram types on sub systems structural modeling.

By comparing all the structural diagrams in Figure 3.11, we see that only the Component Diagram meets all of our requirements. Figure 3.12 depicts such a model. The component in the diagram represents a component in the enterprise.

The interactions between components follow the same needs as the interactions between the systems, therefore the sequence diagram is the best option for modeling this view. The only difference is that messages represents subsystem operations instead of system actions and that the lifelines represent components rather than systems.

3.3.4 Summary

In the previous section it was argued that the modeling of the enterprise should be done at three different levels of detail or abstraction, namely, enterprise level, systems level and subsystems level. At each of these levels we selected a set of requirements which we considered important at these levels,

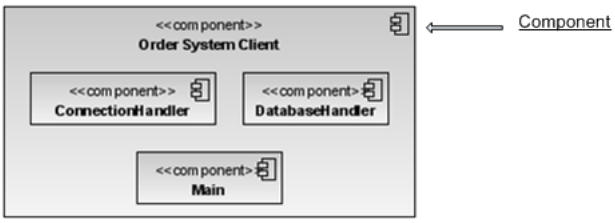

Figure 3.12: An Example Component Diagram.

and then, based on this, we selected a model to capture these requirements. Figure 3.13 shows this choice.

Level	View	Model
Process	Behaviorial	Activity Diagram
Systems	Behaviorial	Sequence Diagram
	Structural	Deployment Diagram
Subsystem	Behaviorial	Sequence Diagram
	Structural	Component Diagram

Figure 3.13: The main modeling views.

We have also established which parts of the diagrams that represent the different parts of the enterprise (See Figure 3.16). Our aim is now to take all these models and merge them into a data structure of the enterprise that permits easy analysis. The next step is therefore to assign an object representation to each of these parts.

Figure 3.14 shows the relation between the elements in the UML diagrams and the data structure of the enterprise representation. The UML diagrams and diagram elements are colored yellow, while the objects in the enterprise representation are colored gray.

The enterprise environment is represented by a single Deployment Diagram. The diagram is decomposed into components and nodes, where the first represents a System, while the latter represents an Environment.

A System object is represented by Component Diagram, which consists of components that represents Component objects.

A business process is represented by an Activity Diagram. The Activity Diagram is decomposed into several parts. The Action represents a business function. The Transition represents the transition between business functions. The decision, fork and merge elements all have their own object in the enterprise representation.

Each business function is represented by a Sequence Diagram. Sequence Diagrams are composed of messages and lifelines. The messages represent System Actions, whilst the lifelines represent systems. A System Action is represented by a Sequence Diagram, where the messages represent Sub System Actions and the lifelines represent components.

3.4 Designing the enterprise representation data structure

We must now use the enterprise objects (marked as grey in Figure 3.14) to define a data structure that follows the structure of Figure 3.15. From now on we call this data structure the *EnterpriseRepresentation*.

3.5 The EnterpriseRepresentation object model

Figure 3.16 shows the object structure of the EnterpriseRepresentation data structure. It represents the structural and behavioral parts of the enterprise and it builds the structure from UML models.

The structure of the enterprise is extracted from the structure of the UML diagrams, but to be able to describe the enterprise richly enough to analyze it, we also need to supply the models with extra information. This information is stored as stereotypes and tag values in the models.

Every entity in the Enterprise extends the base class EnterpriseEntity, which facilitates the storage of such extended information. Each entity has one or several stereotypes, and a set of tags. The TagValues- and Stereotype-objects are stored in an ArrayList (a dynamic array in the C# language).

An EnterpriseEntity also has a name and an id that resolves any possible name conflicts.

I will present the different parts of the data structure in the next sections. As this is a complex data structure, I will use object instance models to illustrate how the structure manifests itself.

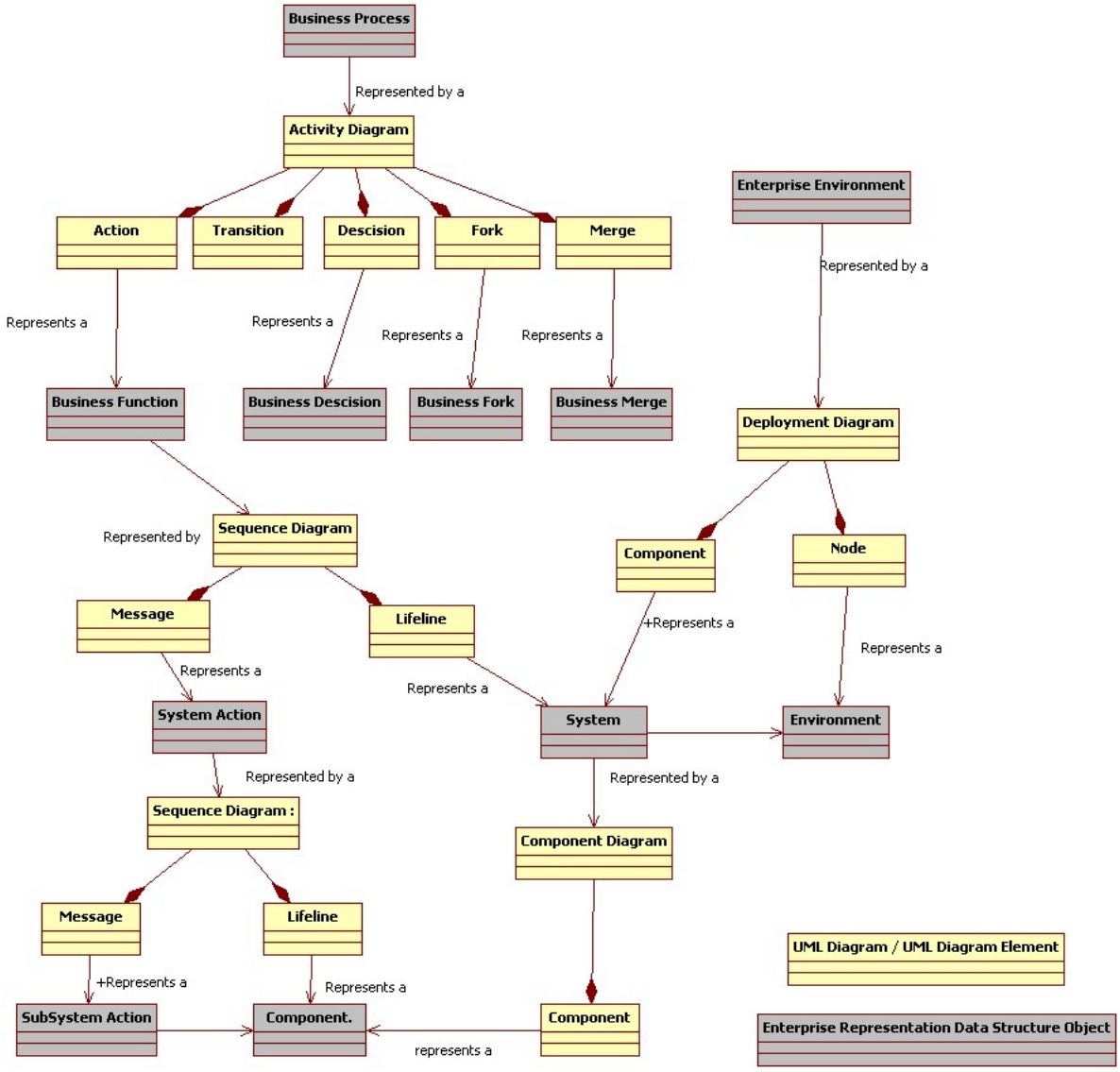

Figure 3.14: The Relation between UML and the objects in the Enterprise Representation.

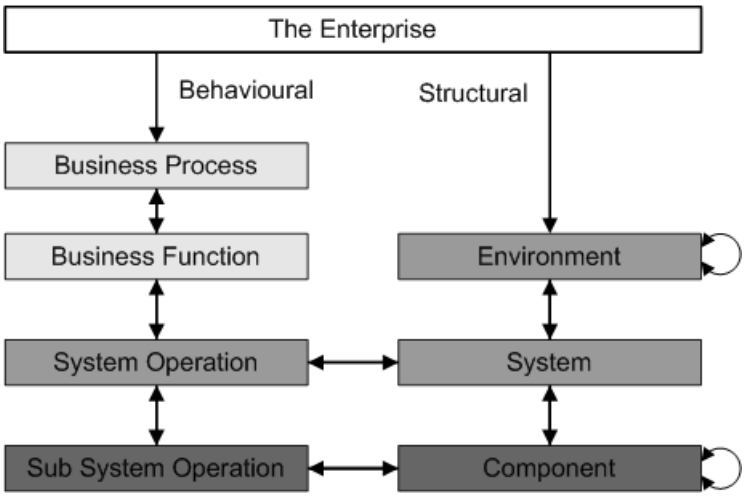

Figure 3.15: The structure of the enterprise.

3.5.1 Behavioral part

The EnterpriseRepresentation has an ArrayList populated with BusinessProcess objects. This list contains all the business processes in the enterprise. A business process is realized by a set of business functions, but as described earlier, when modeling the business process, we use more than the business functions. We also have a set of objects that are used to describe things such as decisions etc.

The business process is therefore realized by a graph of BusinessFunctionObjects. There are four different objects (nodes in the graph) that extend the BusinessFunctionObjects. They all share the property that they follow another BusinessFunctionObject through the attribute *prev*. The first BusinessFunctionObject in the graph has prev value *null*.

The BusinessFunctionFork is a node where the business flow branches out in two or more paths. The BusinessFunctionFork therefore has an ArrayList named *nexts*, which contains all the BusinessFunctionObjects that it branches to.

The BusinessFucntionMerge is a node where all the branches from a BusinessFunctionFork meet up. The BusinessFunctionDecision is a node where the business flow branches into two or more paths, but unlike the fork node only one of the paths are executed. Each path has a probability assigned to it. Like with the BusinessFunctionFork, the nexts are stored in an ArrayList.

The BusinessFunction is a node in this graph, which represents a business

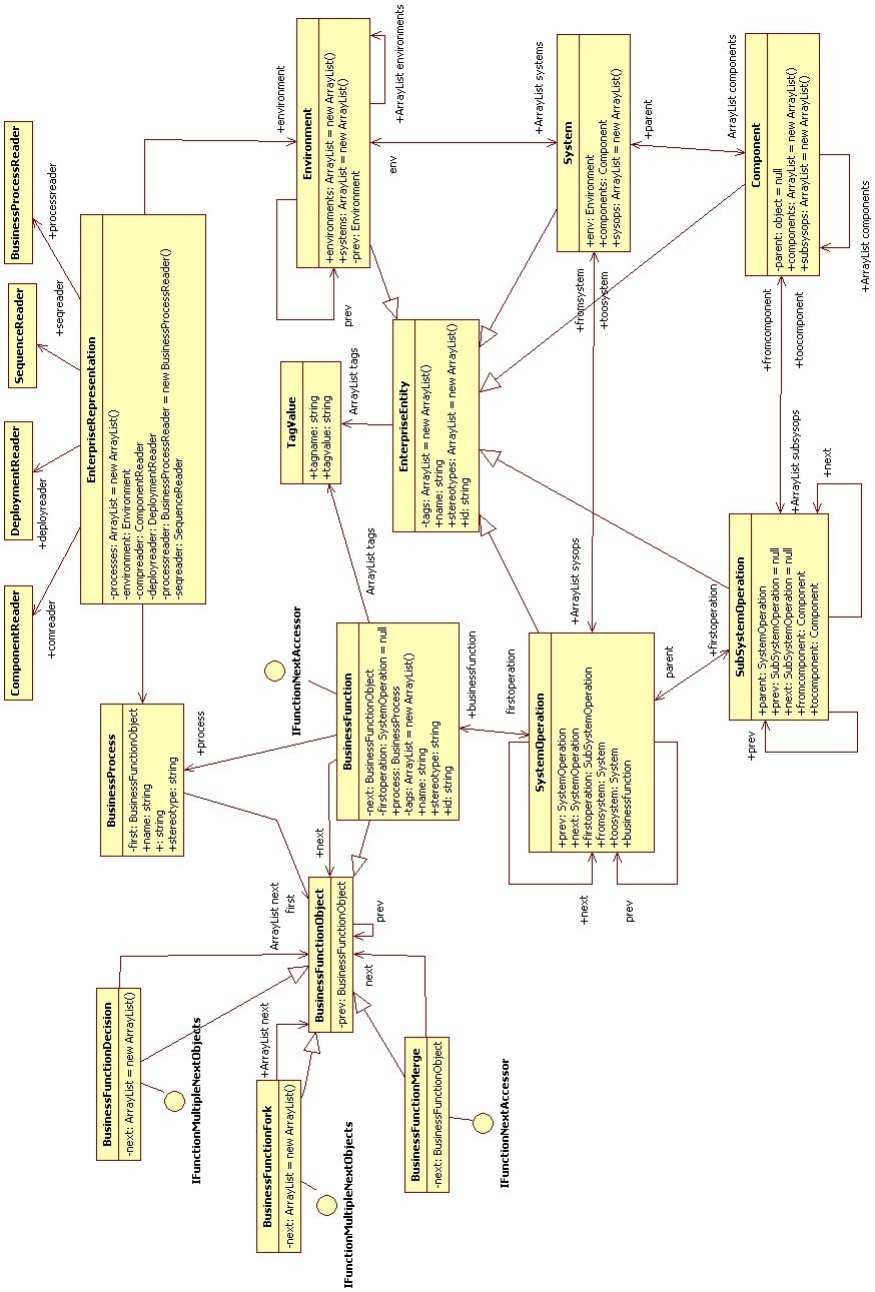

Figure 3.16: The EnterpriseRepresentation data structure.

function. The BusinessFunction is followed by an another BusinessFunctionObject in the case where the given BusinessFunction is the last one in the graph, it has the value *null*. It also holds a pointer to the BusinessProcess of which it is a part.

Figure 3.17 shows an object instance model of an example graph as it manifests itself at runtime.

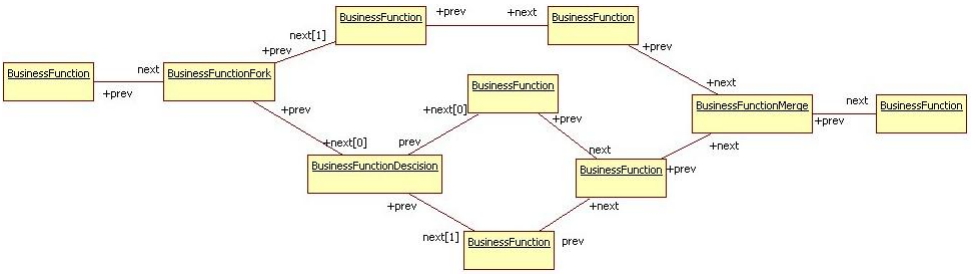

Figure 3.17: An example graph of BusinessFunctionObjects.

Note that BusinessFunction does not extend EnterpriseEntity, but implements its members separately. The reason for this is that BusinessFunction also must extend the base class BusinessFunctionObject, and the C# language does not support multiple inheritance.

Each BusinessFunction is composed of a set of SystemOperation objects. Unlike the BusinessProcess which is modeled in the parallel activity diagram, it takes basis in the Sequence Diagram. In other words, it is only a linked list of SystemOperations objects (see Figure 3.18). Each of the SystemOperation objects also keeps a reference to its business function. The BusinessFunction has a reference to the first System Operation through the *firstoperation* member.

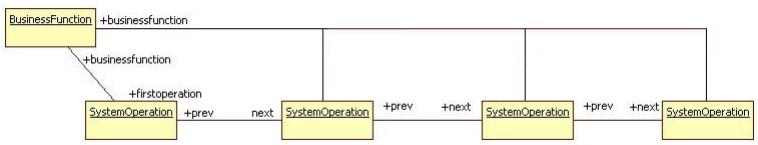

Figure 3.18: An example System Operation linked list.

The SystemOperation is composed by a sequence of SubSystemOperations, which is represented by a linked list with the *firstoperation* attribute in SystemOperation pointing to the first item in the list. Each of the SubSystemOperation objects in this linked list keeps a reference parent to its SystemOperation.

The SystemOperation keeps a reference to the two systems that it involves (Figure 3.19). If the system operation is only performed on one system, these two references both reference this single system.

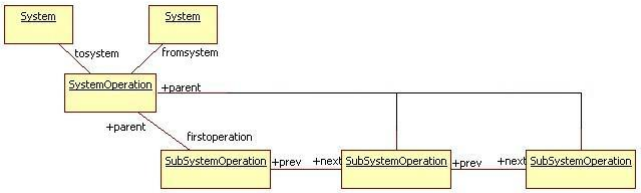

Figure 3.19: SystemOperation with neighboring objects.

The SubSystemOperation keeps a reference to the components involved in the operation (Figure 3.20).

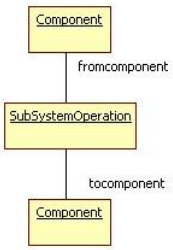

Figure 3.20: The structure of a SubSystemOperation.

3.5.2 Structural part

The structure of the Enterprise is represented in the EnterpriseRepresentation by an Environment object environment with the name "Enterprise". As previously described, an Environment can be further divided into sub environments. Therefore each Environment has an ArrayList of environments. The Environment also has a set of systems which run within the environment stored in the ArrayList *systems*.

The System is divided into a group of components in the ArrayList *components*, which in turn may be divided into components. Each of these entities also keep a reference to their super entity.

The components, systems, and environments make up a tree structure (Figure 3.21).

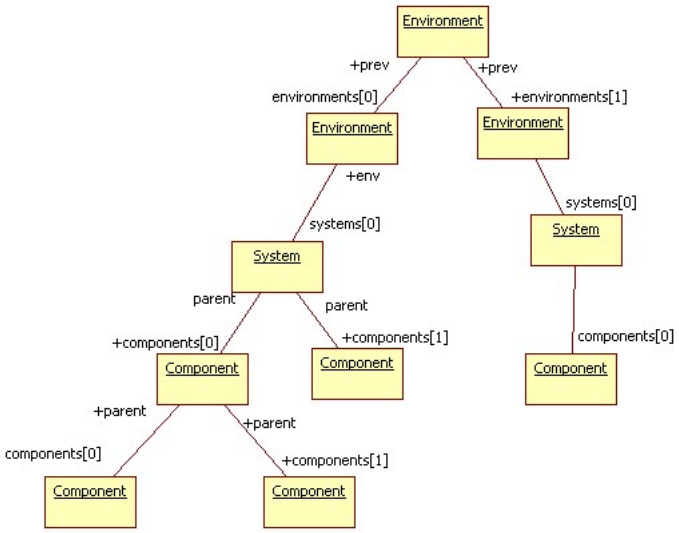

Figure 3.21: Enterprise structure tree - object instance model.

System and Component also keep an ArrayList (sysop and subsysops) that reference the (sub)operations that is associated with the respective entity. These redundant references are used so that we can easily traverse the structure. Figure 3.22 shows the access to the sub systems from the System object.

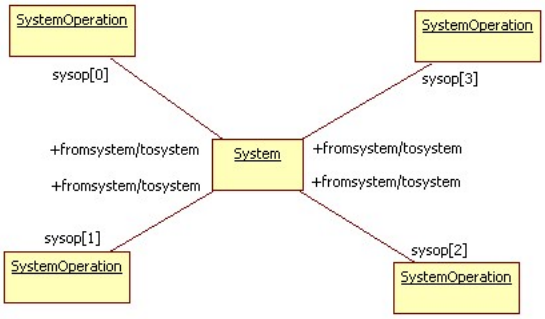

Figure 3.22: System/SystemOperation relation - object instance model.

3.6 The XMI-Readers

The EnterpriseRepresentation class has four classes for parsing UML diagrams.

1. The DeploymentReader which reads Deployment diagrams and populates the environments and systems.
2. The ComponentsReader which reads Component diagrams and populates a System object with Components.
3. The BusinessProcessReader which reads an Activity Diagram and populates a BusinessProcess.
4. The SequenceReader which reads Sequence Diagrams and populates the structure with either subsystem operations or system operations.

In the next sections the different readers will be described in more detail.

3.6.1 ComponentReader

Introduction

The *ComponentReader* transforms the externally XMI-stored Component Models into a tree structure of *Component* objects within the *EnterpriseRepresentation*.

The input

The structure of the Component Diagram XMI-representation is found in Figure 3.23. UML has many features within its Component Diagram, but the meta model below uses only the parts that we need in our case, which is representing a hierarchy of components within a system.

Figure 3.24 shows a Component Diagram that conforms to the meta model in Figure 3.23, and has the XMI-representation shown in Figure 3.25.

We do not represent any communication paths between the components, as they are given by the behavioral diagrams.

Every model is encapsulated in a *<UML:Model>* element, and is the top element of a Component Diagram. A model further contains an element named *<UML:Namespace.ownedElement>* which encapsulates the elements

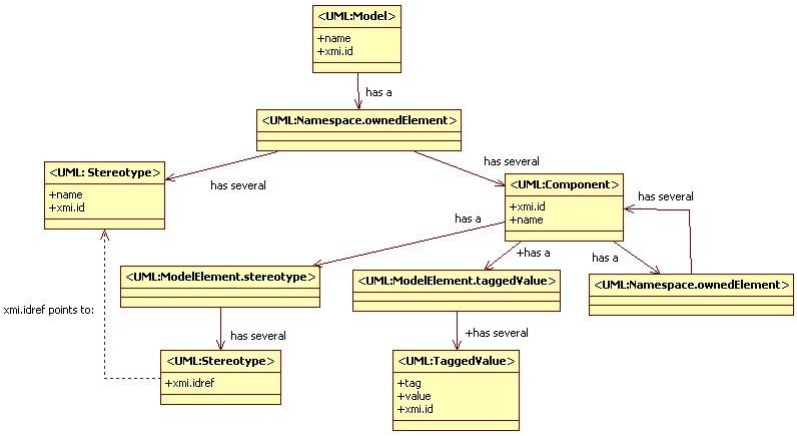

Figure 3.23: The structure of the XMI representation of a Component model.

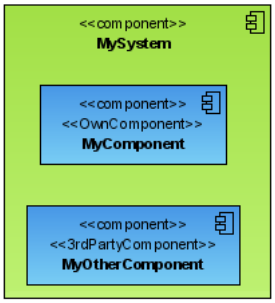

Figure 3.24: An example Component Diagram.

```
<?xml version="1.0" encoding="UTF-8" ?>
<XMI xmi.version="1.2" xmlns:UML="org.omg/UML/1.4">
  <XMI.content>
    <UML:Model name="untitled" xmi.id="XqKa9lCD.AAAAQlD">
      <UML:Namespace.ownedElement>
        <UML:Component name="MySystem" xmi.id="y.Oy9lCD.AAAAQik">
          <UML:Namespace.ownedElement>
            <UML:Component name="MyComponent" xmi.id="4Ley9lCD.AAAAQi9">
              <UML:ModelElement.stereotype>
                <UML:Stereotype xmi.idref="XD0K9lCD.AAAAQj9" />
              </UML:ModelElement.stereotype>
              <UML:ModelElement.taggedValue>
                <UML:TaggedValue tag="needsnet" value="yes" xmi.id="vecq9lCD.AAAAQkz" />
              </UML:ModelElement.taggedValue>
            </UML:Component>
            <UML:Component name="MyOtherComponent" xmi.id="YV.y9lCD.AAAAQjW">
              <UML:ModelElement.stereotype>
                <UML:Stereotype xmi.idref="hn4K9lCD.AAAAQju" />
              </UML:ModelElement.stereotype>
            </UML:Component>
          </UML:Namespace.ownedElement>
        </UML:Component>
        <UML:Stereotype name="3rdPartyComponent" xmi.id="hn4K9lCD.AAAAQju" />
        <UML:Stereotype name="OwnComponent" xmi.id="XD0K9lCD.AAAAQj9" />
      </UML:Namespace.ownedElement>
    </UML:Model>
  </XMI.content>
</XMI>
```

Figure 3.25: An example Component Diagram XMI file.

in the model. In our case we have two possible elements. A component represented with the element *<UML:Component>* and stereotypes represented by *<UML:Stereotype>*.

The components themselves may have a *<UML:Namespace.ownedElement>* element which contains more components, as well as their own components. Each component may have several *<UML:Stereotype>* elements under the owned element *<UML:ModelElement.stereotype>* and a set of tag values, represented by the element *<UML:TaggedValue>* under the owned element *<UML:ModelElement.taggedValue>*.

Parsing the input

In the following section I will describe the most central algorithms that parse the input and build the structure. There are three central methods. The first serves as an accessor method for outside classes. This method calls a second method that locates the starting point in the XMI node tree. Finally, a third recursive method traverses through this node tree until its fully parsed and gradually builds the object representation along the way.

Pseudo code 1 *GetComponent*

Input: *systemid* - The unique id to the system which the components are to be appended.
Input: *filename* - The filepath to the XMI file containing the component diagram.
Input: *env* - An Environment object that contains the system that is the Component Diagram is going to populate.
Returns: Component
Description: Parses an XMI-document representing a UML Component model into a tree of Component objects (From the EnterpriseRepresentation data structure) and appends it to a system.
Filename: ComponentReader.cs

```
1: Parse XMI-document into XMLDocument document
2: name ← GetReferenceToSystemByID(id).Name
3: return GetSuperComponent(name)
```

Pseudo code 1 parses the XMI-file into an XMLDocument object (A .NET object for storing XML files), which gives us a data structure representation of the XMI-document. The XMLDocument consists of XMLNode objects that each stores an element.

Then the name of the system which is to be populated with components is extracted. The reason for this is that the components in the Component Diagram must be nested in a component with the same name as the system it decomposes. This serves as a runtime validity check while parsing, gives the models more readability, and lastly it works as a dummy node in the top of the tree, making it possible for the first level of component detail to

have multiple components. Then, lastly, *GetSuperComponent* (see Pseudo Code 2) is called using the name of the system.

Pseudo code 2 *GetSuperComponent*

Description: This method locates the Component in the diagram which has the same name as the system it is going to represent and creates a tree of Component objects of its sub components.
Input: *name* - The name of the system that is going to be populated.
Returns: Component
Filename: ComponentReader.cs

```
1:  for each XMLNode node with type "UML:Component" in document do
2:    if node.Name equals name then
3:      thecomponent ← create new Component based on data in node
4:      if node has child nodes then
5:        for each XMLNode subnode in node.children do
6:          if subnode.Type equals "UML:Namespace.ownedElement" then
7:            AppendSubComponents(subnode.children,thecomponent)
8:          end if
9:        end for
10:     end if
11:   end if
12: end for
13: return thecomponent
```

The *GetSuperComponents* locates the component within the Component Diagram that shares the same name as the system that is to be populated, and then creates a tree structure of its sub-components. The Component *thecomponent* is created. This is the top node of the tree we are going to grow. The tree structure is built using the recursive method *AppendSubComponents* in pseudo code 3.

AppendSubComponents takes two objects as input. The first is the tree structure of components that has already been built, and the second is the XML-representation of the subcomponents of the component (the child nodes of the *XMLNode*) that was parsed in the previous recursive call.

The method extracts all the components owned by the current component, creates *Component* objects, appends these to the tree structure, and passes it recursively to another call of the AppendSubComponent.

The branching of the recursive calls are only stopped when a component doesn't have any children. When all branches are finalized, we have a built a tree structure of *Component* objects out of our XMI representation.

Pseudo code 3 *AppendSubComponents*

Description: A recursive algorithm that traverses the part of an XMI model stored in an XMLNodeList and builds a tree of components.
Input: XmlNodeList *nodes* - A list of XMI child nodes of the previouly parsed component.
Input: Component *thecomponent* - The component to which we are going to append the subcomponents of *nodes* too.
Returns: None (void)
Filename: ComponentReader.cs

```
 1: for each XmlNode node in nodes do
 2:    if node.Name equals "UML:Component" then
 3:       c ← new Component(based on data in node)
 4:       thecomponent.component.Add(c)
 5:       for each XMLNode subnode in node.children do
 6:          if subnode.Name equals "UML:Namespace.ownedElement" then
 7:             AppendSubComponents(subnode.children,c)
 8:          end if
 9:          if subnode.Name equals "UML:ModelElement.stereotype" then
10:             c.Stereotype ← GetStereotypeNameByID(based on id in subnode)
11:          end if
12:          if subnode.Name equals "UML:ModelElement.taggedValue" then
13:             for each XMLNode subsubnode in subnode.children do
14:                c.AddTag(new TagValue(based on data in subsubnode))
15:             end for
16:          end if
17:       end for
18:    end if
19: end for
20: return thecomponent
```

3.6.2 DeploymentReader

The *DeploymentReader* transforms the externally XMI-stored Deployment Diagram into a tree structure of *System* and *Environment* objects within the *EnterpriseRepresentation*.

The input

The structure of the Deployment Diagram is found in Figure 3.26. The structure of a Deployment Diagram is closely related to that of the Component Diagram.

The Deployment Diagram contains Nodes (Environments), which themselves can be further partitioned into sub nodes or have a set of components (Systems). Both nodes and component extend a set of stereotypes and a set of tag values.

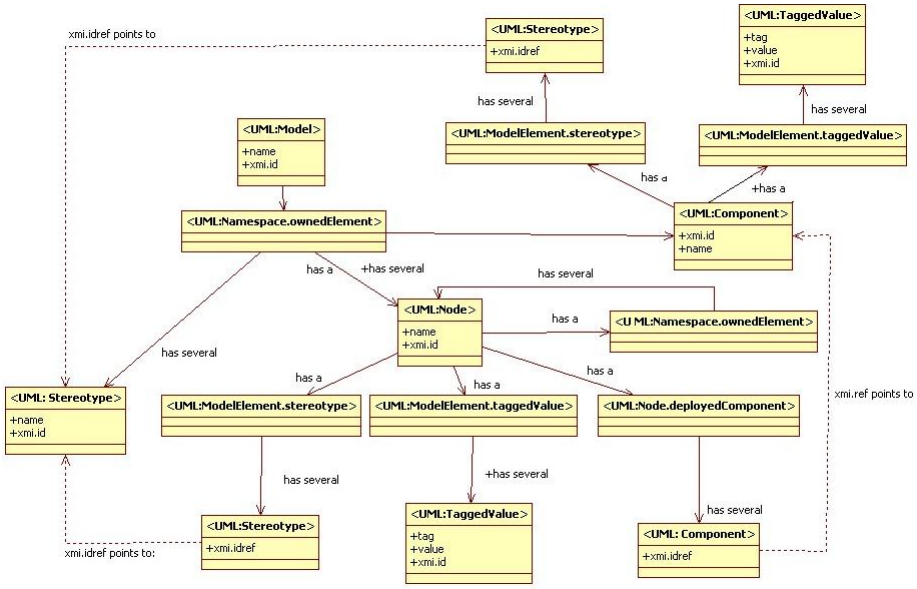

Figure 3.26: The structure of the XMI representation of a Deployment model.

Parsing the input

This parsing algorithm follows the same principle as the one with the Component Reader. One method to access the parser, one method that localizes the "starting point" of the model, and a recursive method which does the actual parsing and object structure creation.

Pseudo code 4 *GetDeployment*

Description: A method that takes an XMI stored Deployment Diagram and transforms it into a tree structure of Environment and System objects.
Input: String filname - The file path of the XMI document that is to be parsed.
Returns: Environment
Filename: DeploymentReader.cs

```
1: Parse XMI-document into XMLDocument document
2: return GetSuperNode()
```

Pseudo code 4 parses the XMI-file into an XMLDocument object and calls the GetSuperNode (see pseudo code 5 mothod which locates the first (as in the node that has no parents) node in XMI-representation.

The GetSuperNode method locates the Node "Enterprise" which every representation of the systems and environments in the enterprise is encapsulated within.

Pseudo code 5 *GetSuperNode*

Description: A method that starts the parsing of a Deployment Diagram currently loaded in the global *document* object.
Input: None.
Returns: Environment object which is the top node of the tree structure of Environment and System objects.
Filename: DeploymentReader.cs

```
 1: for each node with type "UML:Node" in document do
 2:    if node.Name equals "Enterprise" then
 3:       Environment env ← new Environment(based on data in node)
 4:       if node has children then
 5:          for each subnode in node do
 6:             if subnode.Name equals "UML:Namespace.ownedElement" then
 7:                env ← GetSubNodes(subnode.children, env)
 8:             end if
 9:          end for
10:       end if
11:    end if
12: end for
13: return env
```

When it is found, the recursive *GetSubNodes*-method (6) is called and traverses the structure very much like the ComponentReader did. The only difference being that this parser builds a tree structure with two different kinds of object, both Systems and Environments.

3.6.3 BusinessProcessReader

Introduction

The *BusinessProcessReader* transforms the externally XMI-stored Activity Diagrams into a *BusinessProcess* object.

The input

The Activity Diagram can have two kinds of states, AcitityStates and PseudoStates. The ActivityState is an action and represents a business function. There are four kinds of pseudo states. The Initial and Final state, which defines where the Activity Diagram starts and ends. The Junction is a state that handles several input or output transitions. The Junction is a Merge Node if it has multiple incoming transitions, and a Decision Node if it has only a single incoming transition. The last pseudo state is the Fork, which represents a Fork node.

Pseudo code 6 *GetSubNodes*

Description: A recursive method that parses a Deployment Diagram stored in the XMI-notation.

Input: XmlNodeList *nodes* - A list of XMI child nodes of the previously parsed environment.

Input: Environment *env* - The environments that the sub nodes is to be appended to.

Returns: Environment object which is the top node of the tree structure of Environment and System objects.

Filename: DeploymentReader.cs

```
1:  for each node in nodes do
2:    if node.Name equals "UML:Node" then
3:      Environment e ← New Environment(based on data in node)
4:      env.environments.Add(e)
5:      for each subnode in node.children do
6:        if subnode.Name equals "UML:Namespace.ownedElement" then
7:          GetSubNodes(subnode.children, e)
8:        end if
9:        if subnode.Name equals "UML:Node.deployedComponent" then
10:         for each subsubnode in subnode.children do
11:           if subsubnode.Name equals "UML:Component" then
12:             System s ← new System(based on data in subsubnode)
13:             Add s to e.systems
14:             Add Tags to s
15:           end if
16:           if subnode.Name equals "UML:ModelElement.taggedValue" then
17:             for each subsubnode in subnode.children do
18:               if subsubnode.Name equals "UML:Component" then
19:                 e.AddTag(new TagValue(based on data in subsubnode))
20:               end if
21:             end for
22:           end if
23:         end for
24:       end if
25:     end for
26:   end if
27: end for
28: return env
```

All of these states are encapsulated within the *UML:CompositeState.subvertex element*. The transition between the different states is defined by the *UML:Transistion* elements which reside under the *UML:Model/UML:Namespace.ownedElement*.

The ActivityStates also have a set of stereotypes and tag values affiliated with it, which are stored under the elements of *<UML:ModelElement.stereotype>* and *<UML:ModelElement.taggedValue>*.

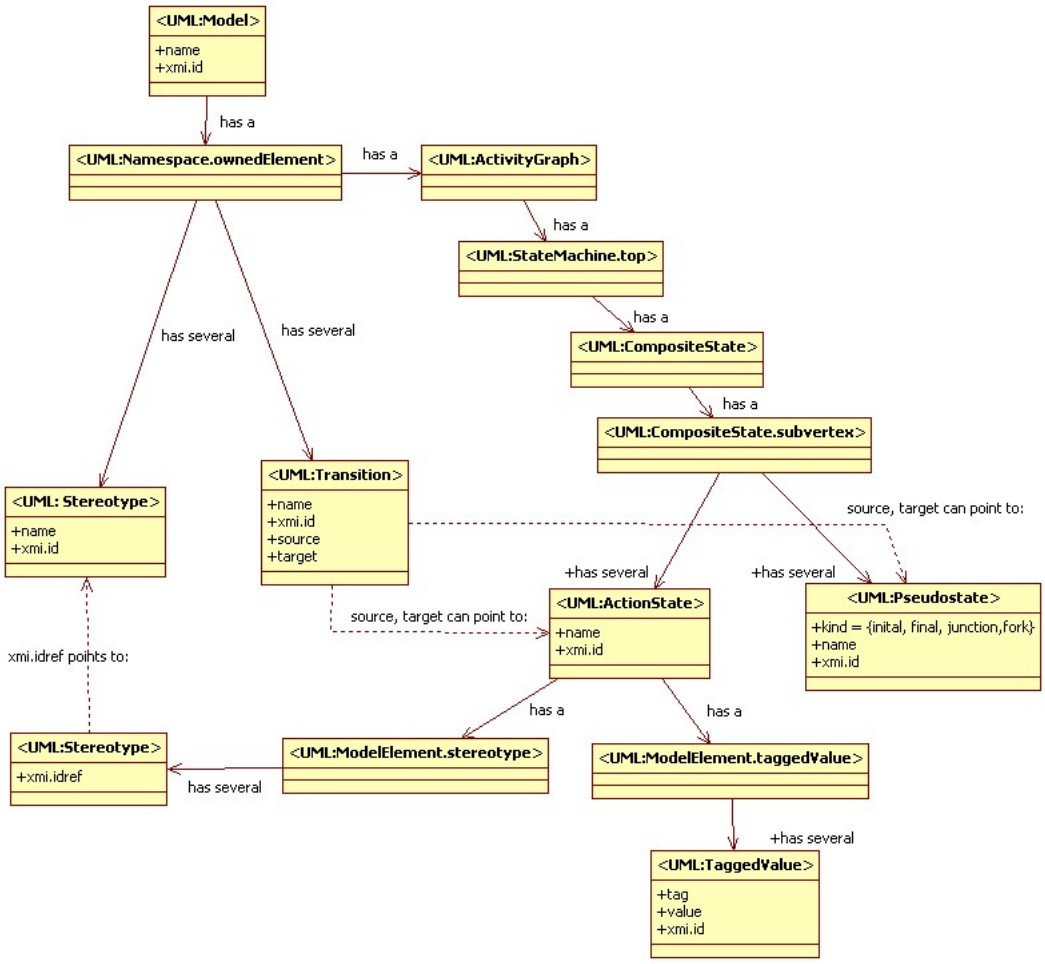

Figure 3.27: The structure of the XMI representation of an Activity Diagram.

Parsing the input

This parsing algorithm has three main methods. One method that starts the parsing, one method that finds the initial node, and finally a recursive method that builds the object representation of the business process.

Pseudo code 7 *GetProcess*

Description: This method is the access method of the BusinessProcessReader, it lets the user supply an Activity Diagram which is parsed into a BusinessProcess object.
Input: string *filename* - The filpath of the Activity Diagram that holds the Business Process.
Input: string *name* - The name of the Business Process.
Returns: BusinessProcess
Filename: BusinessProcessReader.cs

```
1: Parse filename into XMLDocument document
2: process ← new BusinessProcess
3: process.Name ← name
4: ParseProcess(process)
5: return process
```

The GetProcess method (Pseudo Code 7) loads the XMI file into an XML-Document object and creates a *BusinessProcess* object.

The next step is the *ParseProcess* (Pseudo Code 8) method which locates the initial state and then locates the transition that goes from the initial node. With this information the first state is located.

This state can either be an AcitivtyState, a BusinessForkNode, or a BusinessFunctionDescion. It cannot be a BusinessFunctionMerge node as these join several incoming transitions, and there can only be one incoming transition from the initial node. Either way, the first object is created and appended to the business process as the first function.

The first object either has one or multiple next objects. The method now calls the recursive *ParseFunctionObject* method for each of the nexts. The *ParseFunctionObject* takes three parameters, namely the current object, the previous object, and the id of the current object. The first thing done in the method is to append the current object to previous objects next attribute. This two-way reference between objects makes it easier to traverse the structure.

The next step is to see see if the current object has one or multiple next objects. If there only is one next object, the id of that object is found by going through all the transitions and looking for the state that has an incoming transition from the current object. When this id is found, a new object is created based on this value, and then the method recursively calls itself.

If there are multiple next objects, the next id values are extracted and the procedure in the prevents paragraph is done for each one.

3.6.4 SequenceReader

Introduction

The SequenceReader parses a Sequence Diagrams and produces a linked list of SystemOperation or SubSystemOperation objects.

The input

The main parts of the Sequence Diagram is the message and the classifier role, also known as the life line, which are the vertical lines that messages pass between. Both can have several stereotypes and have a set of tagged values. The messages are linked to the life lines with the two pointers, *receiver* and *sender*.

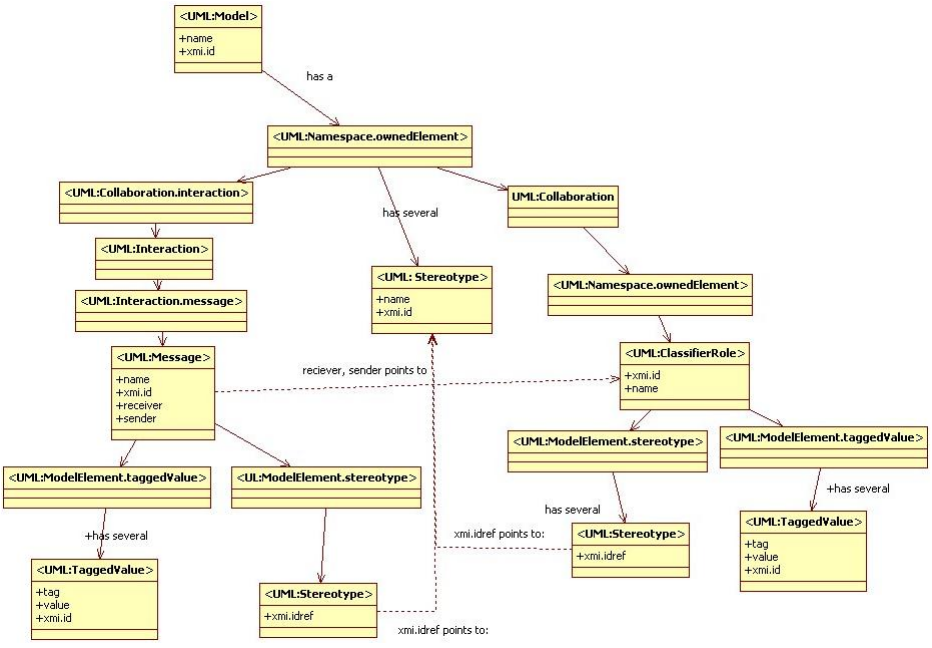

Figure 3.28: The structure of the XMI representation of a Sequence Diagram.

Parsing the inputs

The structure representation of the operations is simple. The SystemOperations is simply a double linked list that point to their parent BusinessFunction. Likewise the SubSystemOperation is a simple linked list where each member has a pointer to its parent SystemOperation. Given the structure of the XMI document, this is the most straightforward parsing with respect to structural traversal.

I am now going to describe the process of parsing the SystemOperations. Keep in mind that the way we parse SubSystemOperations is nearly identical, the difference being in what kind of objects that is created, and which they reference. SubSystemOperations reference *Component* objects rather than *System* objects. Given this, I will only document the first.

There are two main methods. The first is GetSystemOperation, which is the accessor method that outside classes use to parse a diagram. It simply reads the XMI document into an XMLDocument, extracts the sequence of the messages, and stores it in an ArrayList called *msg*.

Then we go through each of the id values in the *msg* list and create a SystemOperation for each of them. A doubly linked list structure is maintained.

The *SystemOperation* object itself is created by the method *CreateSystemOperationMessage*. It assures that the *SubSystem* object maintains the right references to the other parts of the *EnterpriseRepresentation* structure, namely the systems it involves and the *BusinessFunction* object it realizes.

When creating a list of *SubSystemOperations*, we use a similar method named *CreateSubSystemOperation*.

Pseudo code 8 *ParseProcess*

Description: The method takes an empty business process as input and locates the first business function object in the globally stored XMI file. The first business function is appended to the business process. It then uses the *ParseFunctionObject* method to build the whole business function graph.

Input: BusinessProciess *process* - The business process that the business functions are to be assocatied with.

Returns: BusinessProcess

Filename: BusinessProcessReader.cs

```
 1: firstobject ← Get First object in the activity diagram.
 2: if firstobject is an ActivityState then
 3:     process.First ← new BusinessFunction
 4:     Append tags and stereotypes to process.First
 5:     next ← get the id of the next state
 6:     if next is a Fork Node then
 7:         nextobj ← new BusinessFunctionFork
 8:     end if
 9:     if next is a Decision Node then
10:         nextobj ← new BusinessFunctionDesicion
11:     end if
12:     if next is an Action then
13:         nextobj ← new BusinessFunction
14:     end if
15:     Process.First.Next ← nextobj
16:     ParseFunctionObject(nextobj,process.First, next)
17: else if firstobject has multiple nexts then
18:     if firstobject is a Fork Node then
19:         process.First ← new BusinessFunctionFork
20:     end if
21:     if firstobject is a Decision Node then
22:         process.First ← new BusinessFunctionDecision
23:     end if
24:     for each next in firstobject do
25:         if next is an Action then
26:             nextobj ← new BusinessFunction
27:         else if next is a Merge Node then
28:             nextobj ← new BusinessFunctionMerge
29:         else if next is a Decision Node then
30:             nextobj ← new BusinessFunctionDescision
31:         else if next is a Fork Node then
32:             nextobj ← new BusinessFunctionFork
33:         end if
34:         if nextobj not null then
35:             if firstobject is a BusinessFunctionDesicsion then
36:                 Add probabilities to firstobject.
37:             end if
38:             set nextobj as a next object in process.First
39:             ParseFunctionObject(nextobj, process.First, next)
40:         end if
41:     end for
42: end if
43: return process
```

Pseudo code 9 *ParseFunctionObject*

Description: A recursive method that builds the business function graph. It takes the current node as input and recursively passes the node's children to itself.
Input: Object *function* - The current function.
Input: Object *prev* - The previously parsed function.
Input: String *current* - The ID of the current object.
Returns: None (*void*)
Filename: BusinessProcessReader.cs

```
 1: function.previous ← prev
 2: currenttype ← get the current object type
 3: if currentobject has multiple next objects then
 4:     for each nextobject in currentobject do
 5:         BusinessFunctionObject next ← null
 6:         if nextobject is an ActivityState then
 7:             next ← new BusinessFunction
 8:             Extract stereotypes and tags to next.
 9:         else if nextobject is Merge Node then
10:             next ← new BusinessFunctionMerge
11:         else if nextobject is a Decision Node then
12:             next ← new BusinessFunctionDecision
13:         end if
14:         if nextobject is a Fork Node then
15:             next ← new BusinessFunctionFork
16:         end if
17:         if next not null then
18:             current.next.Add(next)
19:             ParseFunctionObject(function.next,function,nextid)
20:         end if
21:     end for
22: else
23:     nextid ← GetNextID(current)
24:     ObjectType nexttype ← GetObjectType(nextid)
25:     BusinessFunctionObject next ← null
26:     if MultipleNexts(nexttype) then
27:         if nexttype is a ObjectType.DescsionNode then
28:             next ← new BusinessFunctionDecision
29:         else
30:             next ← new BusinessFunctionFork
31:         end if
32:     else
33:         if nexttype is a ObjectType.ActivityState then
34:             next ← new BusinessFunction(GetActivityStateName(nextid), process,
                nextid)
35:             Extract stereotypes and tags to next.
36:         else if nexttype is a ObjectType.MergeNode then
37:             next ← new BusinessFunctionMerge
38:         end if
39:     end if
40:     if next is not null then
41:         function.Next ← next
42:         ParseFunctionObject(function.Next, function, nextid)
43:     end if
44: end if
```

Pseudo code 10 *GetSystemOperations*

Description: A method that parses all the messages in a Sequence Diagram and builds an doubly linked list of System Operations.

Input: Environment *enterprise* - The Enterprise environment that contains the all the systems that the System Operations are associated with.

Input: string *filename* - The filepath of the Sequence Diagram.

Input: BusinessFunction *parent* - The business function that the all of the System Operations realize.

Returns: SystemOperation

Filename: SequenceReader.cs

```
 1: Parse file into XMLDocument document
 2: msg ← GetMessageSquence
 3: SystemOperation first ← new SystemOperation of msg[msg.Count - 1]
 4: msg.RemoveAt(msg.Count - 1)
 5: SystemOperation current
 6: SystemOperation newop
 7: current ← first
 8: while msg.Count greater than 0 do
 9:     newop ← CreateSystemOperationMessage of msg[msg.Count - 1]
10:     current.next ← newop
11:     newop.prev ← current
12:     current ← newop
13:     msg.RemoveAt(msg.Count - 1);
14: end while
15: return first
```

Pseudo code 11 *CreateSystemOperationMessage*

Description: A method that creates a single SystemOperation object based on an id.
Input: string *id* - The XMI id of the message in the Sequence Diagram that we want to create an SystemOperation of.
Returns: SystemOperation
Filename: SequenceReader.cs

```
 1: for each node with type "UML:Message" in document do
 2:     if node.id equals id then
 3:         SystemOperation sysop ← new SystemOperation(based on data in node)
 4:         sysop.fromsystem ← Get reference to system based on data in node
 5:         sysop.toosystem ← Get reference to system based on data in node
 6:         sysop.fromsystem.AddSystemOperation(sysop)
 7:         sysop.toosystem.AddSystemOperation(sysop)
 8:         if node has children then
 9:             for each subnode in node do
10:                 if subnode.Name equals "UML:ModelElement.stereotype" then
11:                     sysop.AddStereotype(based on id in subnode)
12:                 end if
13:             end for
14:             if subnode.Name equals "UML:ModelElement.taggedValue" then
15:                 for each subnode in node do
16:                     sysop.AddTag(based on id in subnode)
17:                 end for
18:             end if
19:         end if
20:     end if
21: end for
22: return sysop
```

Chapter 4

Enterprise Analyzer

4.1 Introduction

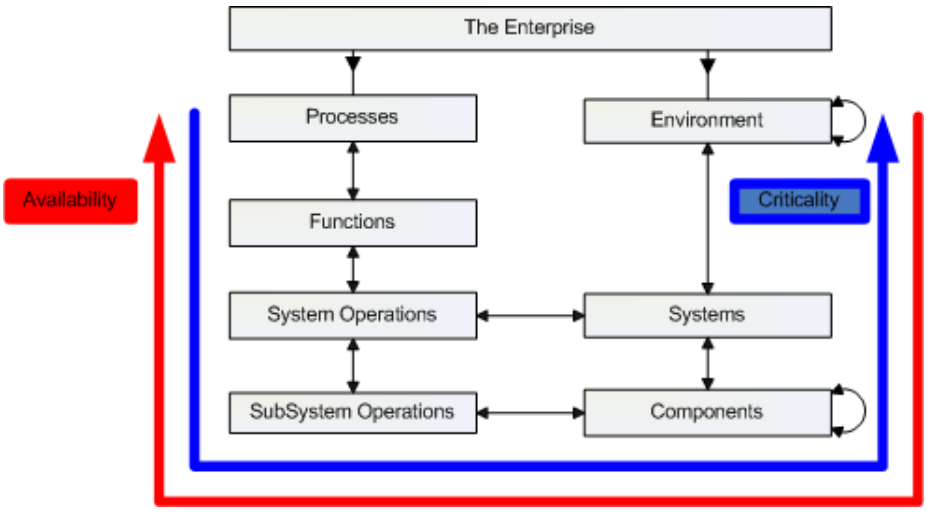

Figure 4.1: The principle of analysis.

The Enterprise Analyzer takes the enterprise representation as input and performs an automatic analysis of it. Figure 4.1 shows the principle of the analysis.

The structural parts of the enterprise are the basis for the availability of the business functions, and this availability is inherited from the structural parts to the behavioral.

Each business process has a criticality value assigned to it, which is inherited

from the business process to the structural parts of the enterprise.

4.2 Structure of the Enterprise Analyzer

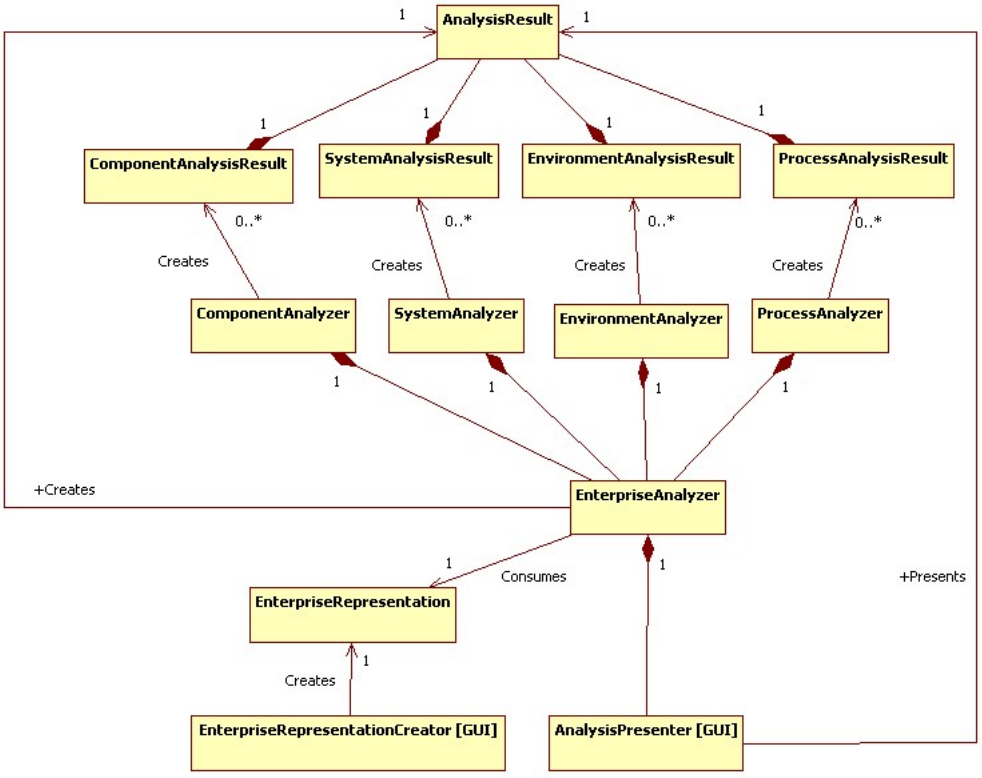

Figure 4.2: Enterprise Analyzer tool structure.

Figure 4.2 shows the structure of the Enterprise Analyzer tool. The tool is divided into two parts. The first part lets the user build, save, and load enterprise representations through the *EnterpriseRepresentationCreator* GUI, whilst the other part analyzes an enterprise representation and presents it with the *AnalysisPresenter* GUI.

The *EnterpriseRepresentationCreator* creates an *EnterpriseRepresentation* object. This object has a set of methods used to build the representation. *EnterpriseRepresentationCreator* simply bridges the user and this functionality.

The *AnalysisPresenter* has an *EnterpriseAnalyzer* object which has all the

functionality to analyze the enterprise, and it consumes an *EnterpriseRepresentation* object.

The *EnterpriseAnalyzer* has a *ComponentAnalyzer*. The *ComponentAnalyzer* analyzes the components and produces a *ComponentAnalysisResult* object that contains the results.

Likewise, the EnterpriseAnalyzer also has analyzers that perform analysis on systems, environments, and processes. All the analysis results are gathered in the object *AnalysisResult*, which is presented by the *AnalysisPresenter*.

How to use the Enterprise Analyzer described in the user manual in the appendix.

4.3 Analysis

The result of the analysis is shown in Figure 4.3.

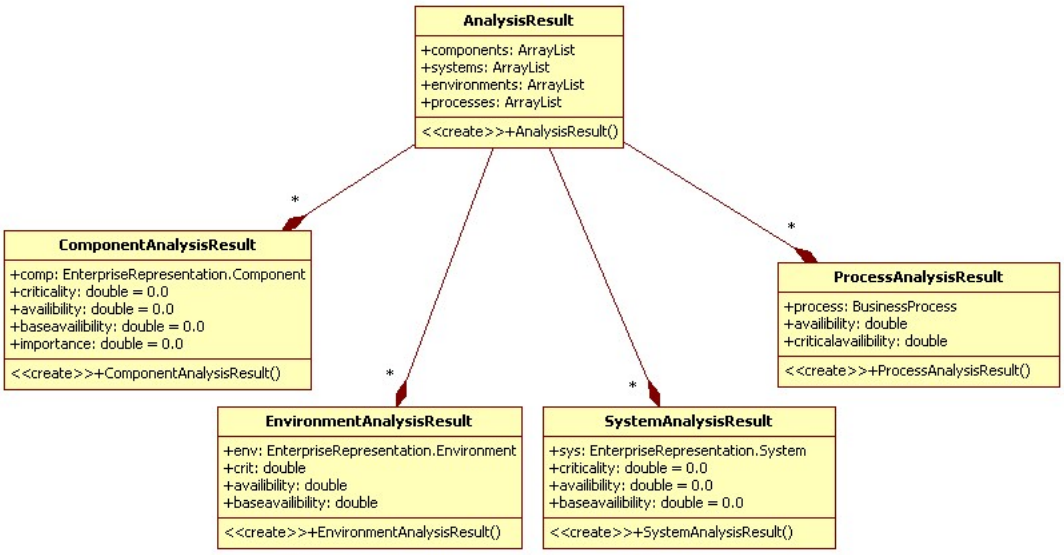

Figure 4.3: Structure of an AnalysisResult object.

The AnalysisResult class encapsulates four ArrayLists which stores classes that hold the analysis result of the different parts of the enterprise.

Each result consists of a reference to the object it analyzed in the enterprise representation and a set of analysis values.

Every structural part has a criticality. The criticality is a number that describes how critical the part is to the overall enterprise.

The structural parts have two kinds of availability. The base availability A_B is the availability of a structural part isolated. The total availability A_T is the availability of the part also considering external contributions to its availability.

For example, $A_B(C)$ is the availability of a component C, whilst $A_T(C)$ is the availability of the component when considering the availability of the system and environments it runs under.

The components also have an *importance* value, which is a function of availability and criticality. A higher importance, the more we gain by spending time on increasing its availability.

The process has an *availability* and a *critical availability*. The *availability* is the percentage of time that the whole process is functioning completely. The *critical availability* is how often the all the critical parts that a of the process needs are available. A critical part is something that is necessary for the business processes to succeed.

The next sections will describe these parameters further.

4.4 Calculating criticalness

One of the two main focus points of the Enterprise Analyzer is calculating the criticalness of different parts of the system. Each of the business processes are given a relative criticality value based on their criticality.

The enterprise representation model encompasses the whole enterprise, and therefore we are able to trace this criticality to all the parts of the system.

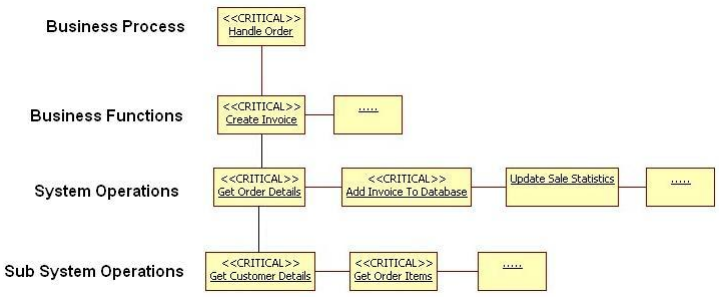

Figure 4.4: Criticalness in the behavioral part.

Figure 4.4 shows the business process *"Handle Order"* decomposed into the three levels of modeling. It starts with a business analyst who identifies that handling orders are critical for the enterprise and therefore marks it as critical.

Finally, the business analyst creates the business scenario by defining a set of business functions. One of the business functions identified is that we have to create an invoice. The business analyst then asks himself the question. "Is creating an invoice critical to realizing the business process "Handle Order"?", which of course it is, and he marks it as "critical".

At the next level a software architect is given the task of modeling a solution that realizes "Create Invoice" using the systems. E.g., the architect identifies several system operations such as, "Get Order details" "Add invoice to database" and "Update sales statistic". The architect asks himself, "Which of these three are critical for realizing "Create Invoice"?" To create an invoice he has to know what the customer bought, and he must store the invoice. Therefore the two first are marked critical. Updating the sales statistics is not critical. If it fails, the invoice will still be created.

Lastly, another architect is given the task to create a solution that realizes the "Get Order Details". He identifies several critical sub system operations such as "Get Customer Details" and "Get Order Items".

At each level the modelers have only decided what is critical at their own level in realizing their goal. They have no knowledge about how their models fit in with the whole enterprise. The architect who identified the task "Get Customer Details" had no idea about whether his solution introduced business critical functionality.

From the enterprise model, we may track the criticalness downwards and identify it as critical to realizing "Handle Order". If this functionality fails, we will not be able to handle incoming orders. If creating invoices was not critical to handling an order, "Get Customer Details" would not be critical either, since we couldn't track a continuous chain of criticalness upwards from "Get Customer Details" to "Handle Order".

Hence, by making the modelers use the stereotype "Critical" to mark what is important at their level of behavioral modeling, we may calculate how the criticalness transcends into the enterprise.

The next logical step is to relate this criticalness to the structural parts of the enterprise. Because the Enterprise Representation interlinks the behavioral and structural parts of the enterprise, we may trace this criticality further.

Suppose that we give a business process a relative criticalness of 100. We can see how this spreads out in the enterprise in Figure 4.5. Through system

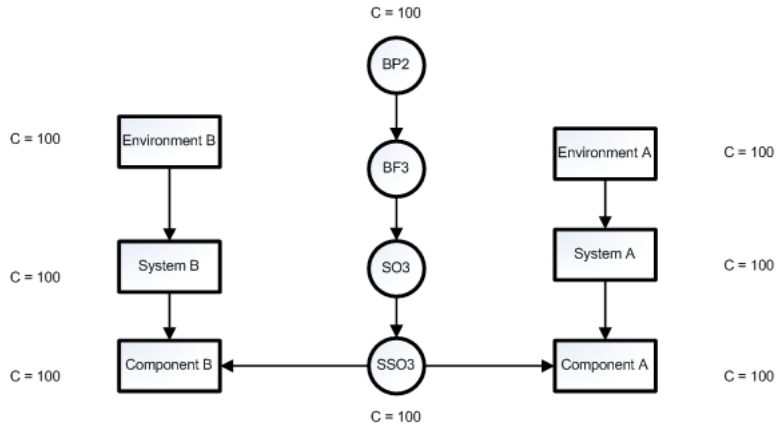

Figure 4.5: Component criticality inheritance - One business process.

operation and sub system operations the behavioral part of the enterprise is related to the structural part. Therefore the operations inherit the criticality from the business processes they realize, and the structural parts of the enterprise inherits this criticality by realizing the operations.

4.4.1 Calculating component criticality

Figure 4.6 shows how the criticality of a component is calculated. The figure shows a simplified enterprise with two business processes *BP1* and *BP2*, both with a criticality of 100. We assume that all the business functions, system operations, and sub system operations are critical in the following examples.

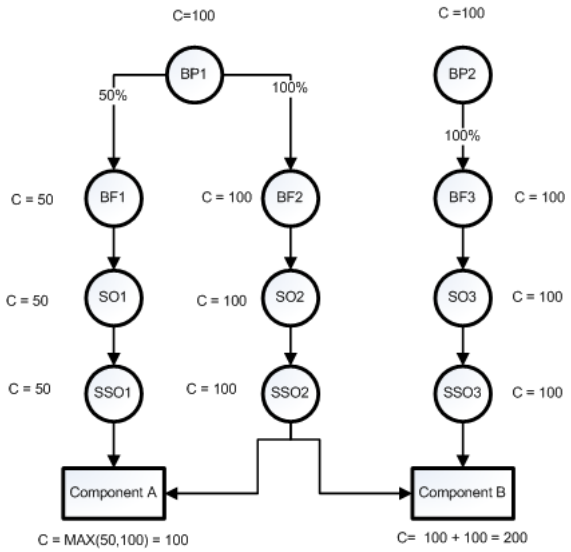

Figure 4.6: Component criticality inheritance - Multiple business process.

Components interact with sub system operations, therefore the criticality of a component is inherited from the sub system operations with which it is linked.

Component A is used by two operations SSO1 and SS02, which realizes BP1 and BP2. BP1 has a criticality of 50, and BP2 has a criticality of 100. SSO1 only have a criticality of 50 since it is only used 50 percent of the times BP1 is executed.

What is then the criticality of Component A? By adding the criticality of the sub system operations we get 150, but a component cannot be more critical than the business processes it realizes. We could have a hundred sub system operations linked with this component, yet it would not be 150 times as critical as the business process. The criticality of a component C that realizes a business process BP is

$$Criticality(C, BP) = MAX(\{SSO_0 \dots SSO_i\}), \tag{4.1}$$

where SSO_i criticality realizes BP, and SSO_i uses component C. In the

case above with BP1, this would mean MAX(50,100) which is 100.

When a component is realizing more than one business process, the calculation is a bit different. The criticality of a component that realizes a set of business processes BP_0 to BP_n is

$$Criticality = \sum Criticality(C, BP_0) + \ldots + Criticality(C, BP_n), \quad (4.2)$$

where Criticality(C,BP) is given by (4.1). In other words, it is the sum of the maximum sub system operation from each business process. Component B will always be used to realize both BP1 and BP2, therefore it gets a criticality score of 200. The algorithm for calculating this value is found in Pseudo Code 12.

Pseudo code 12 *ComponentCriticalityCalculation*

Description: Calculates the criticality of the components in an enterprise based on the business processes.

Input: An Enterprise Representation with a set of business processes with associated values of criticality.

Filename: ComponentAnalysis.cs

```
 1: for each component in EnterpriseRepresentation do
 2:     maxcrit ← 0
 3:     critsum ← 0
 4:     for each process in EnterpriseRepresentation do
 5:         prob ← Get the probability of component being used during an execution of the
            process.
 6:         for each subsysop realized by component do
 7:             if subsysop is critical then
 8:                 if subsysop realizes process then
 9:                     crit ← GetCriticality(subysop) · prob
10:                     break out of for each
11:                 end if
12:             end if
13:         end for
14:         critsum ← critsum + crit
15:     end for
16:     Component.criticality ← critsum
17: end for
```

The algorithm calculates the criticality of each component by going through each business process. Each sub system operation of the component is then checked to see wether it realizes the given business processes and if it is critical.

The probability of the component being executed during the process is calculated, and the criticality contribution from that business process is calculated as the product of the process criticality and the probability. The criticality

of the component is the sum of the criticality of the business processes that is realized by the component.

4.4.2 Calculating system and environment criticality

Calculating the criticality of a system is done the same way as with components, except that the criticalness is inherited through the system operations usage of systems (See Figure 4.7), and the criticalness is multiplied with the probability of the system being used in an execution of the business process. See Pseudo Code 13.

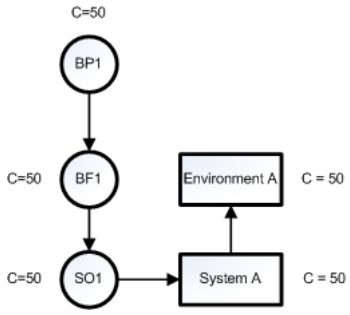

Figure 4.7: System/Environment criticality inheritance.

Pseudo code 13 *SystemCriticalityCalculation*

Description: Calculates the criticality of the systems in an enterprise based on the business processes.

Input: An Enterprise Representation with a set of business processes with a value of criticality associated with them.

Filename: SystemAnalysis.cs

```
 1: for each system in EnterpriseRepresentation do
 2:     critsum ← 0
 3:     crit ← 0
 4:     for each process in EnterpriseRepresentation do
 5:         prob ← Get the probability of system being used during one execution of the
            process.
 6:         for each sysop realized by system do
 7:             if sysop is critical then
 8:                 if sysop realizes process then
 9:                     crit ← GetCriticality(sysop) · prob
10:                     break out of for each statement
11:                 end if
12:             end if
13:         end for
14:         critsum ← critsum + crit
15:     end forSystem.criticality ← critsum
16: end for
```

The criticality of an environment is inherited by the systems that run under it. Pseudo Code 14 shows the pseudo code for calculating the environments criticality. For each environment the criticality is summed by the criticality

of each process that is realized by the environment multiplied with probability of the environment being used during one execution of the given business process.

Pseudo code 14 *EnvironmentCriticalityCalculation*

Description: Calculates the criticality of the systems in an enterprise based on the business processes.

Input: An Enterprise Representation with a set of business processes with associated values of criticality.

Filename: SystemAnalysis.cs

```
1: for each environment in EnterpriseRepresentation do
2:     crit ← 0.0
3:     for each businessprocess in EnterpriseRepresentation do
4:         prob ← Get the probability of environment being used during an execution of
           the process.
5:         crit ← crit + bp.Criticality · prob
6:     end for
7:     environment.criticality ← crit
8: end for
```

4.5 Calculating Availability

The other value we are concerned with in the Enterprise Tool are the availabilities.

4.5.1 Calculating structural availability

There are two kinds of structural availability, the base availability and the total availability, as was discussed earlier in this chapter.

The three structural parts in the enterprise (components, systems, and environments) can have two tag values named *MTBF* and *MTTR*.

Based on these parameters we can calculate the availability of the part as shown in (4.3).

$$A_B = \frac{MTBF}{MTBF + MTTR}. \tag{4.3}$$

If the part also have the tag value *parallel* that says that the part has n-parallel components whose availability is not inter-dependant. In this case the calculation of the availability is shown in (4.4).

$$A_B = 1 - (1 - (\frac{MTBF}{MTBF + MTTR}))^n. \tag{4.4}$$

The availability of a component is the product of its own base availability multiplied with the base availability of n components super components ($n \geq 0$), multiplied with the base availability of the system the components run under and the product of all the k environments that the systems run under ($k > 0$). (Equation (4.5)).

$$A_T(C) = A_B(C) \cdot \prod_{i=0}^{n} A_B(C_i) \cdot A_B(S) \cdot \prod_{i=0}^{k} A_B(E_i). \tag{4.5}$$

The pseudo node is found in Pseudo Code 15. It uses the method *GetAvailibilityOf*, which takes a *EnterpriseEntity* object and calculates its availability (shown in (4.4) and 4.3) based on the tag values in the XMI document. The algorithm starts with a component and traverses the structural part of the enterprise representation upwards, while calculating the overall availability.

Likewise, the availability of a system is the product of the availability of the system itself and all the environments that the system runs under. (Equation (4.6)).

Pseudo code 15 *ComponentAvailibilityCalculation*

Description: The code for calculating the availability of a component.
Input: Component *comp* - The component that is to be analyzed.
Filename: ComponentAnalyzer.cs

```
1: availibility ← GetAvailibilityOf(comp)
2: for each super component c of component do
3:     availibility ← availibility· GetAvailibilityOf(c)
4: end for
5: System sys ← component.System
6: availibility ← availibility · GetAvailibilityOf(sys)
7: for each super environment e of sys do
8:     availibility ← availibility · GetAvailibilityOf(e)
9: end for
10: comp.availibility ← availibility
```

$$A_T(S) = A_B(S) \cdot \prod_{i=0}^{n} A_B(E_i). \tag{4.6}$$

The pseudo code for calculating the availability of a system is found in Pseudo Code 16.

Pseudo code 16 *SystemAvailibilityCalculation*

Description: The code for calculating the availability of a system.
Input: System *sys* - The system that is going to be analyzed.
Filename: SystemAnalyzer.cs

```
1: availibility ← GetAvailibilityOf(sys)
2: for each super environment e of sys do
3:     availibility ← availibility · GetAvailibilityOf(e)
4: end for
5: sys.availibility ← availibility
```

The availability of an environment is the product of the availability of the environment itself and the n-environments it runs under. (Equation (4.7)).

$$A_T(E) = A_B(E) \cdot \prod_{i=0}^{n} A_B(E_i). \tag{4.7}$$

The pseudo code for calculating the availability of an environment is found in Pseudo Code 17.

Figure 4.8 shows the calculation of the availability of a hypothetical enterprise structure. All the structural parts have an availability of 0.99.

Pseudo code 17 *EnvironmentAvailibilityCalculation*

Description: The code for calculating the availability of an environment.
Input: Environment *env* - The environment that is going to be analyzed.
Filename: EnvironmentAnalyzer.cs

```
1: availibility ← GetAvailibilityOf(env)
2: for each super environment e of env do
3:     availibility ← availibility · GetAvailibilityOf(e)
4: end for
5: sys.availibility ← availibility
```

4.5.2 Calculating process availability

Based on the availability of the structural parts of the enterprise, we want to calculate the availability of the business processes in the enterprise. The processes interact with the structural parts of the enterprise through system operations and sub system operations.

Figure 4.9 shows how the sub operations inherit the availability of the components. However, as the components run under the same environments, the environments availability contributes to the total availability of the business process more than once. The complicates the calculation. There is however another approach. By taking the product of all the base availabilities of all environments, systems and components, each part only contributes to the total availability once.

Equation (4.8) shows the calculation of the availability of a business process realized by a set of environments, systems, and components. The availability is the product of the contribution to the availability of the components, systems, and environments.

$$A_T(BP) = A_C(C) \cdot A_C(S) \cdot A_C(E). \tag{4.8}$$

Equation (4.9), (4.10), and (4.11) show the calculation of each contribution from n-parts. The contribution to the overall availability from a single part is the probability (P) that the part is used during an execution of the business process multiplied with its base availability, plus the reminder probability ($1 - P$). The idea is that when a part is not used, its contribution to the overall availability is 1.0. The contribution of n-parts is the product of the contribution of each part.

$$A_C(E) = (\prod_{i=0}^{n} A_B(E_i) \cdot P(E_i) + (1.0 - P(E_i)). \tag{4.9}$$

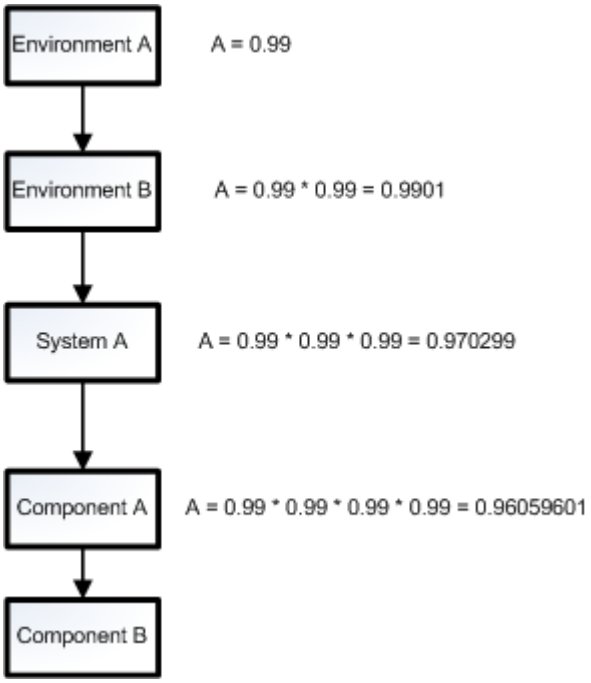

Figure 4.8: Structural availability inheritance.

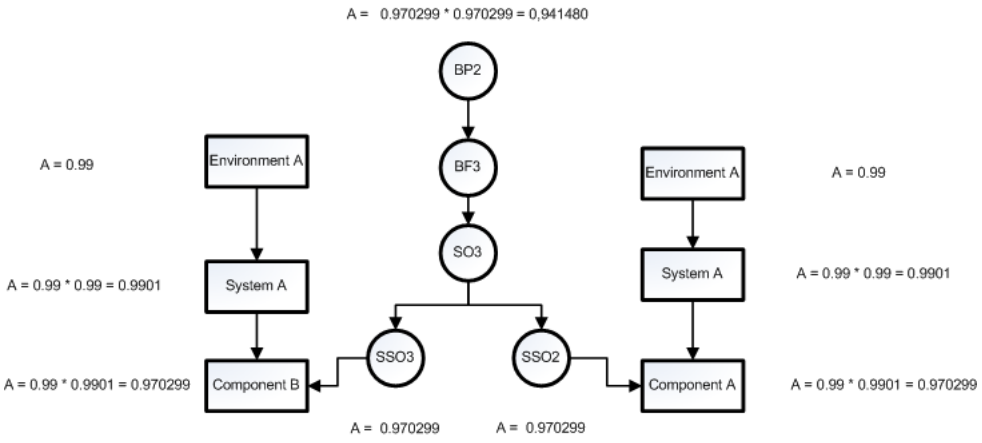

Figure 4.9: Process availability inheritance.

$$A_C(S) = \prod_{i=0}^{n} A_B(S_i) \cdot P(S_i) + (1.0 - P(S_i)). \tag{4.10}$$

$$A_C(C) = \prod_{i=0}^{n} A_B(C_i) \cdot P(C_i) + (1.0 - P(C_i)). \tag{4.11}$$

The pseudo code for calculating the availability of a process is found in Pseudo Code 18.

GetAvgSystemProbability, *GetAvgEnvironmentProbability* and *GetAvgComponentProbability* are methods that calculate how often the part is involved with the realization of a business process.

Pseudo code 18 *ProcessAvailibilityCalculation*

Description: The code for calculating the availability of a business process.
Input: BusinessProcess *bp* - The business process that is going to be analyzed.
Filename: ProcessAnalyzer.cs

```
 1: availibility ← 1.0
 2: components ← GetComponentAnalysis()
 3: environments ← GetEnvironmentAnalysis()
 4: systems ← GetSystemAnalysis()
 5: for each SystemAnalysisResult s in systems do
 6:     if s realizes bp then
 7:         sysprob ← GetAvgSystemProbability(bp,s.system)
 8:         availibility ← availibility · ((sysprob · sys.baseavailibility) + (1.0 - sysprob))
 9:     end if
10: end for
11: for each EnvironmentAnalysisResult e in environments do
12:     if e realizes bp then
13:         envprob ← GetAvgEnvironmentProbability(bp,e.environment)
14:         availibility ← availibility · ((envprob · e.baseavailibility) + (1.0 - envprob))
15:     end if
16: end for
17: for each ComponentAnalysisResult c in components do
18:     if e realizes bp then
19:         compprob ← GetAvgComponentProbability(bp,c.component)
20:         availibility ← availibility · ((compprob · c.baseavailibility) + (1.0 - compprob)
21:     end if
22: end for
23: return availibility
```

The critical availability is the availability of the smallest set of structural parts that can realize the given business processes. In other words, we allow that non-critical parts fail. The pseudo code for this calculation is identical to that of Pseudo Code 18, the only difference being that instead of checking for realization in line 6, 12 and 18, we check for critical realization. Does the structural part realize a critical action that realizes the given business process?

Pseudo code 19 shows the pseudo code for the *GetAvgSystemProbability*. It starts by creating a list called *bfs*. This list will store all the business functions in the given business processes that use the system. Then, for each system operation *subsys* in the system, we check if it realizes the business process. If this is the case, we add it to the bfs-list, but only if it is not already in the list.

After this, we have filled the bfs-list with the business processes that realizes the business process using the given system. The next step is to calculate the probability that, during an execution of the business processes, the system is used. Or in other words, the probability that at least one of the business functions in the bfs-list is used. This is done by passing the first the list to the *GetProbabilityOfBusinessFunctions* method.

Pseudo code 19 *GetAvgSystemProbability*

Description: A method that calculates the probability of system being used during an execution of a business process.
Input: System *sys* - The system that the calculation is performed on.
Input: BusinessProciess *bp* - The business process that the system must realize.
Filename: SystemAnalyzer.cs

```
 1: ArrayList bfs ← new ArrayList
 2: for each SystemOperation sysop in sys.sysops do
 3:    if GetProcessOf(sysop) equals bp then
 4:       contains ← false
 5:       for each BusinessFunction bf in bfs do
 6:          if bf.ID equals sysop.businessfunction.ID then
 7:             contains ← true
 8:          end if
 9:       end for
10:       if contains equals false then
11:          bfs.Add(GetProcessOf(sysop))
12:       end if
13:    end if
14: end for
15: sum ← GetProbabilityOfBusinessFunctions(bfs, bp.First, 1.0)
16: return sum
```

Pseudo code 20 shows the pseudo code for the *GetProbabilityOfBusiness-Functions*.

Before explaining the algorithm code-wise, I will show the principle visually using figures. Figure 4.10 shows an example UML-modeled business process with six business functions and two decisions.

Figure 4.11 (step 1) shows how the object representation of this business process, as it manifests itself in the enterprise representation. The graph is turned into a redundant tree to ease the calculations.

Assume that the system is used to realize business function BF6 and BF3 (marked red in step 2).

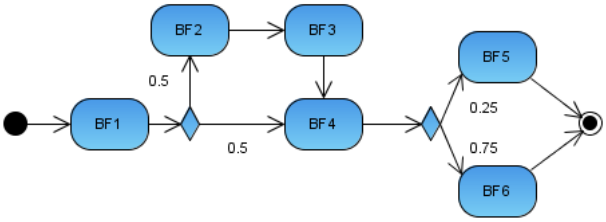

Figure 4.10: Example Activity Diagram.

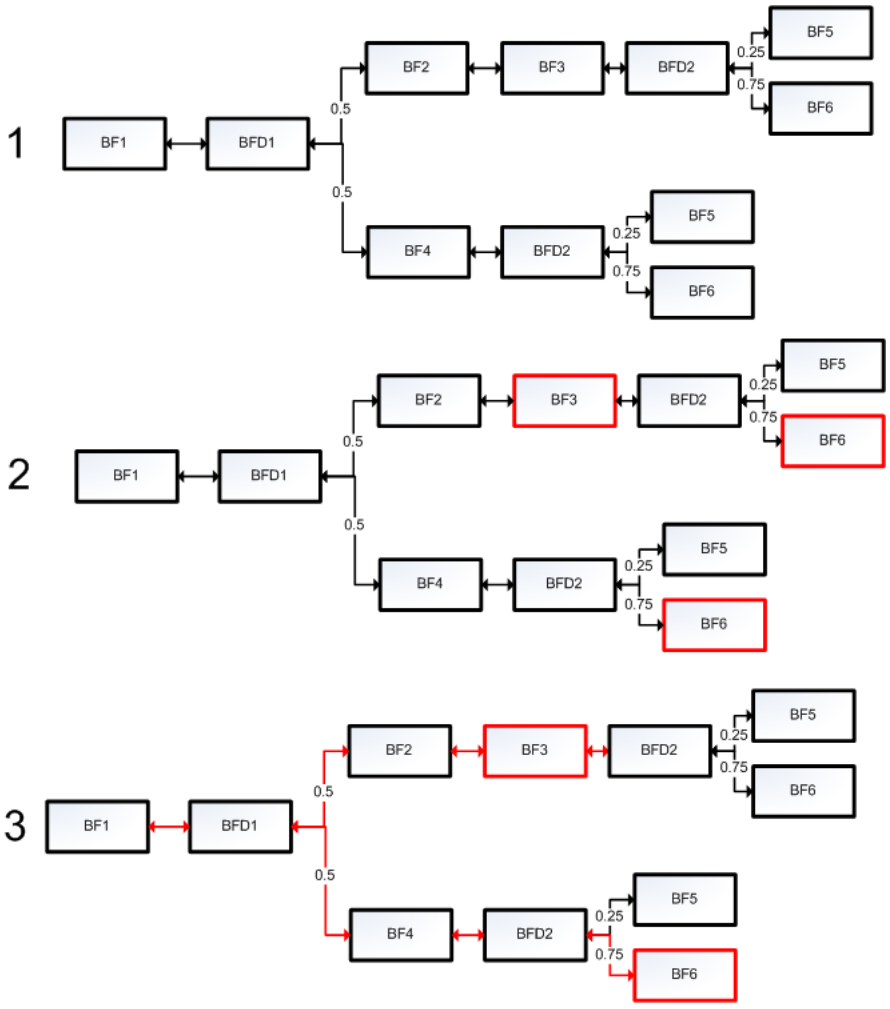

Figure 4.11: GetProbabilityOfBusinessFunctions - Algorithmic steps.

The probability of system usage is then the probability of reaching BF6 or BF3 during an execution of the business process.

This probability can then be calculated by starting in the tree node and tracing each path until we reach either BF6 or BF3. The probability-contribution of one path is the product of each made decision's probability, and the probability of system usage is the sum of all these paths. This is shown in step 3.

The probability of reaching BF6 + Probability of reaching BF3 is

$(0.75 \cdot 0.5 + 0.5) = 0.875$.

The second BF6 that is on the same branch as F3 is not calculated. We cannot come to the second BF6 without using BF3, therefore, the probability of getting to the second BF6 is accounted for in the probability of getting to BF3.

The algorithmic approach is then clear. We need to calculate the probabilities at the *ends*, which are business functions that is realized with the system (marked red in Figure 4.11) and sum them. By traversing the graph recursively we can reach all the ends, but we need to do two more things. Firstly, calculate the probability of getting to the end and then returning it.

The GetProbabilityOfBusinessFunctions method takes a node as input and returns the probability of reaching a red business function by executing its subtree. By passing the top node of the complete tree into the method, we get the probability of encountering a red business function when executing the whole business process.

The structure of the algorithm is then very simple. It recursively calls itself and passes the child(ren) node(s) of the current node. This means that the probability of encountering a red business function during execution of a nodes subtree is the sum of the corresponding probability of all its children, which themselves, in turn, are the sum of the corresponding probability of their own children. This is repeated until we reach a node without traversable children. In the representation there are two kinds of nodes without traversable children. First we have the red business functions, which are business functions that are realized by the system. The second are the nodes that don't have children at all and are not red business functions.

If we reach one of the latter, we know that the current path does not reach a red business function (because then it would have been stopped before), and we return the value 0.0. If we reach a red business function, we return the probability of getting to that business function. This probability value is passed downwards in the call hierarchy. Every time a decision is encountered, the probability is updated by being multiplied by the probability at

the decision. Figure 4.12 visualizes the algorithm applied to the structure shown in Figure 4.11.

The algorithm goes into three branches. One branch ends up in BF5. Since this is a non red node with zero children, its contribution to the sum is zero, and zero is returned. Another branch ends up in BF6. This is a red function and the probability of getting there is 0.375. The last branch ends up in the red function of BF3. The probability of getting there is 0.5. The probability of reaching a red business function in the tree is then the sum of these three branches which is 0.875.

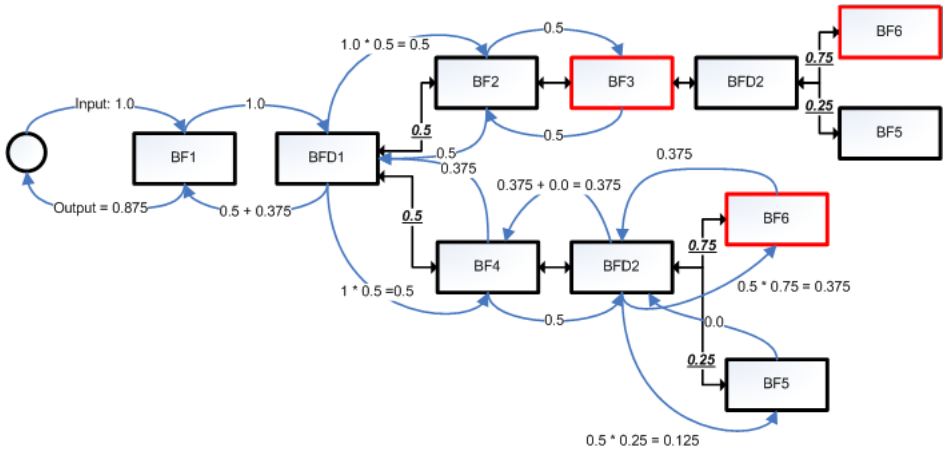

Figure 4.12: GetProbabilityOfBusinessFunctions - Recursive traversal.

Pseudo Code 20 shows the *GetProbabilityOfBusinessFunctions* method which constitutes the algorithm. The method takes three arguments: A *BusinessFunctionObject*, that is, a node in the graph, a list of all the business functions that are realized by the system (red business functions), and a sum value which represents the probability of getting to the node prior to the one that is the input argument.

When the method is first called, we pass along the list of business functions that are realized by the system (red functions), the top node in the tree, and the value 1.0 as sum. The sum value indicates that we have a probability of 1.0 getting to the first node.

There are two kinds of nodes, the ones with one child (*BusinessFunction-Merge-* and *BusinessFunction* objects), and the ones with one or more children (*BusinessFunctionDescision-* and *BusinessFunctionFork* objects).

If the node only have one child, we return the result of the method recursively called by passing along the list, the child node and the sum. The sum is not altered because there is not a conditional forking.

If the node is a *BusinessFunctionDescision*, we return the sum of all the sub branches of its children passed recursively. Since this node is conditional, the sum is multiplied with the probability of getting from the decision node to the given next.

If the node is a *BusinessFunctionFork*, we simply pass along the current sum value to each of the sub branches of its children. We do not update the sum value, since the fork is not conditional.

If we encounter a *BusinessFunction* is in the list (a red function), we do not return a recursive self call, but rather the sum value (the probability of getting to that red function). If we encounter any *BusinessFunction* with no children which is not in the list, we simply return 0, for the reasons mentioned earlier in this section.

In code line 4 there is comparison based on the ID of the business functions. This is done because different business function objects in the tree are different instances, and therefore are we not able to use the equality operator. The reason for this is that when growing the redundant tree, we had to create the business function objects more than once. But all other objects in the enterprise representation are not redundant. This is the reason that this is the first time the equality operator is not used to compare objects.

GetAvgEnvironmentProbability and *GetAvgComponentProbability* follow the same principle, except that we select the business functions that are realized by the given environment and component as input to the method.

4.6 Calculating importance

4.6.1 Calculating component importance

All references to availability in this section will be to base availabilities.

Consider three components in an AND-configuration as shown in Figure 4.13 with an availability of 0.85, 0.90, and 0.95 respectively.

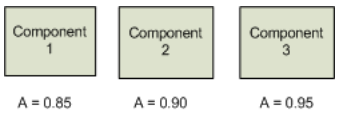

Figure 4.13: An example series of components.

The overall availability of such a configuration is $0.85 \cdot 0.90 \cdot 0.95 = 0.72675$.

Pseudo code 20 *GetProbabilityOfBusinessFunctions*

Description: A recursive method that calculates the probability that a set of business functions are used by a business process.
Input: double *sum* - A value used in the calculation of the probability.
Input: BusinessFunctionObject *obj* - The current object that the recurisve method visits.
Input: Arraylist bfs - The set of business functions that the probability calculation is based on.
Filename: EnterpriseRepresentation.cs

```
 1: partialsum ← 0.0
 2: if obj is a BusinessFunction then
 3:     for each BusinessFunction bf in bfs do
 4:         if bf.ID is obj.ID then
 5:             return sum
 6:         end if
 7:     end for
 8: end if
 9: if obj has multiple nexts then
10:     ArrayList nexts ← Get nexts of obj
11:     if obj is a BusinessFunctionDescision then
12:         ArrayList probs ← Get probabilities of obj
13:         for each next in nexts do
14:             partialsum ← partialsum + GetProbabilityOfBusinessFunctions(bfs, next,
                sum · probability of next)
15:         end for
16:         return partialsum
17:     else
18:         for each next in nexts do
19:             partialsum ← partialsum + GetProbabilityOfBusinessFunctions(bfs, next,
                sum)
20:         end for
21:         return partialsum
22:     end if
23: else
24:     if obj.next not equals null then
25:         return GetProbabilityOfBusinessFunctions(bfs, obj.next, sum)
26:     end if
27: end if
28: return 0.0
```

Assume that the cost of doing an improvement is constant, that is that the cost of going from an availability of 0.2 to 0.4, is the same as going from 0.4 to 0.6. If we had to choose one of the components for improvement, which one would benefit the overall availability the most if it was improved? We show how to quantify the improvement.

The expression for the overall availability is $A(C_1) \cdot A(C_2) \cdot A(C_3)$.

By partial derivation of the overall function by each of the components availability, we can get a quantified value of the improvement of the component.

The improvement of C_1 is then

$$I(C1) = \frac{\partial(A(C_1) \cdot A(C_2) \cdot A(C_3))}{\partial A(C_1)} = A(C_2) \cdot A(C_3). \tag{4.12}$$

Accordingly, the improvement of the tree components is then

$I(C_1) = 0.90 \cdot 0.95 = 0.855.$

$I(C_2) = 0.85 \cdot 0.95 = 0.8075.$

$I(C_3) = 0.85 \cdot 0.90 = 0.765.$

If the component only realizes one business process, the overall importance is the improvement multiplied with the criticality of the business process (C_BP) and the probability (P_C) that the component is used during an execution of the business process.

$$I_{C_i} = \frac{\partial(A(C_0) \ldots A(C_{i-1}) \cdot \ldots A(C_{i+1}) \cdot \ldots \cdot A(C_n))}{\partial A(C_i)} \cdot C_BP \cdot P_C. \tag{4.13}$$

Performing the differentiation, we get

$$I_{C_i} = A(C_0) \cdot \ldots A(C_{i-1}) \cdot \ldots A(C_{i+1}) \cdot \ldots \cdot A(C_n) \cdot C_BP \cdot P_C. \tag{4.14}$$

If a component realizes n business processes, the overall importance is the sum of the importance from each of the n business process.

$$I_{C_i} = \prod_{i=0}^{n} (I_{C_i} = A(C_0) \cdot \ldots A(C_{i-1}) \cdot \ldots A(C_{i+1}) \cdot \ldots \cdot A(C_n) \cdot C_{BP_i} \cdot P_C). \tag{4.15}$$

Pseudo Code 21 contains the pseudo code for the calculation of the component's importance. For each component in the enterprise, we go though each of the business processes. If the component critically realizes the business process, we calculate the probability of the component being used.

We then go through all the components and multiply the base availability of all the components that critically realizes the business process. This product will be the importance of the component in realizing the business process. This importance is further multiplied with the probability that the component is used in the business process and the criticality of the business process. This product can be seen as the gain from the business process of improving the component. This is the calculation in Equation (4.14).

Pseudo code 21 *CalculateComponentImportance*

Description: The pseudo code for the calculation of component importance.

```
 1: for each Component comp in EnterpriseRepresentation do
 2:     sumimportance ← 0.0
 3:     for each BusinessProcess bp in EnterpriseRepresentation do
 4:         if comp critically realizes bp then
 5:             importance ← 1.0
 6:             prob ← GetAvgComponentProbability(comp, bp)
 7:             for each Component c in EnterpriseRepresentation do
 8:                 if c critically realizes bp then
 9:                     if c.ID not equal to comp.ID then
10:                         importance ← importance · GetAvailibilityOf(c)
11:                     end if
12:                 end if
13:             end for
14:             sumimportance ← sumimportance + importance · comp.crit · prob
15:         end if
16:     end for
17:     comp.importance ← sumimportance
18: end for
```

We want the overall gain, therefore we must sum this product for each of the business processes, which corresponds to the calculation in Equation (4.15).

Chapter 5

The Model Analyzer

5.1 Introduction

The Enterprise Tool described in the previous chapter analyzes the representation of the enterprise assembled by a set of models. The Model Analyze does a risk analysis of a single model.

When diagrams have a predefined structure (as UML diagrams have) and contain high fidelity information about the domain (as UML profiles lets us), we can create a tool that transforms the diagram into a set list of risks. This process is from now on called the transformation.

Figure 5.1 shows the process of transformation. A single UML diagram represented in an XMI-document is combined with a *transformation profile*, which is the 'recipe' of the transformation, to produce a list of risks which the risk analyst can read.

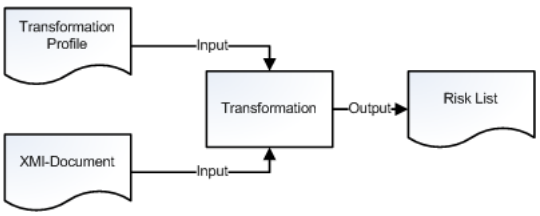

Figure 5.1: The principle of transformation.

The Model Analyzer can do two different types of risk analysis. Preliminary Hazard Analysis (PHA) and Hazard and Operability Study (HAZOP). They are both described in Appendix D. In the depth study done prior to this thesis, [26], I argue that these two methods are the best when doing an risk

analysis of the enterprise.

5.2 The Risk Manager

The Risk Manager GUI is a part of the Model Analyzer that lets the user store risks that are found during analysis. The GUI uses the RiskManager class that writes and reads risks to/from the file "risk.dat". This file stores serialized Risk-objects that represent one risk.

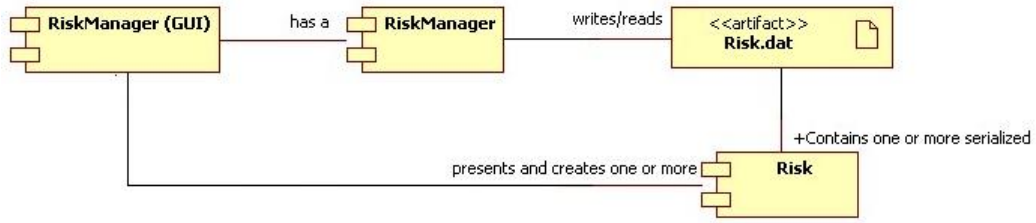

Figure 5.2: The Risk Manager.

The Risk object is shown in Figure 5.3. A Risk contains a name, a description, a likelihood, and a consequence.

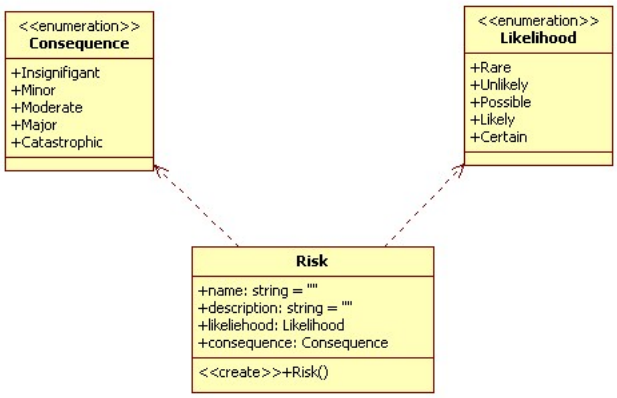

Figure 5.3: The Risk object.

5.3 The Transformers

The transformers turn the diagrams into risk lists. All the transformers share a common architectural pattern which is depicted in Figure 5.4 (a complete class diagram is found in Appendix A).

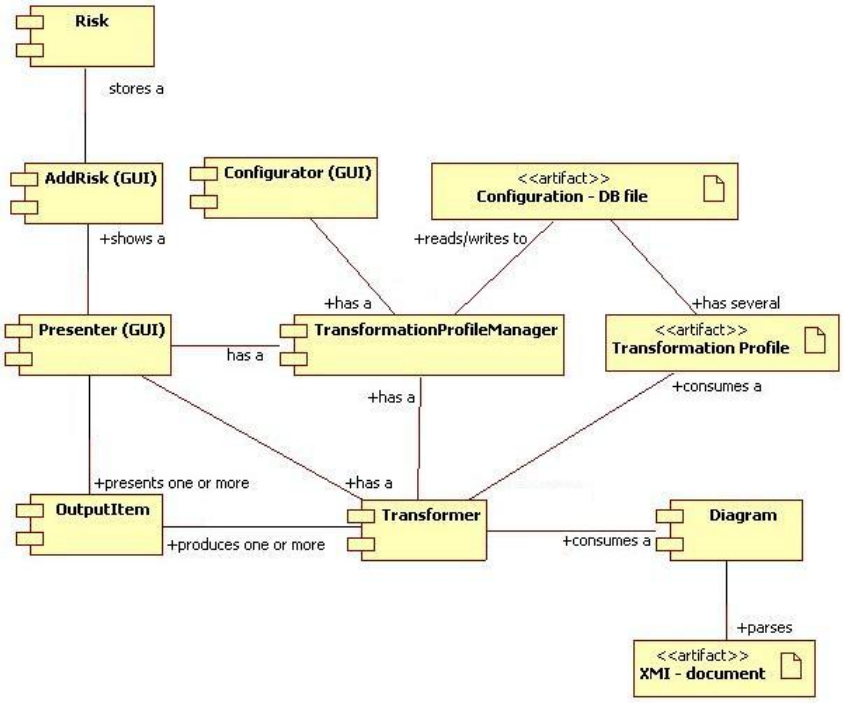

Figure 5.4: Transformer - Architectural pattern.

The transformation logic is stored in a database file, where it is portioned into one or more transformation profiles. The transformer takes one profile and one XMI-file as input.

The TransformationProfileManager is a class that contains methods to store, access, delete, and alter profiles in the configuration database. The Configurator is the user interface that allows users to create and edit profiles.

The Transformer is the class that is responsible for doing the transformation. It consumes two types of data to complete the transformation. It uses a TransformationProfileManager object to retrieve a profile and a *Diagram* object.

The separation of the TransformationProfileManager and the Transformer is a logical one, as both the Presenter and the Transformer need to access the profiles.

The diagrams are stored in the XMI format, but each type of diagram has an object representation. The XMI file is parsed from an XML representation to an object representation. By separating parsing and analysis, both the parsing and transformation functionality can be changed without affecting the other.

The UML specification, which defines the structure of UML diagrams, pretty much stays the same from version to version. The XMI specification, which defines how the UML diagrams is represented in XMI, have seen more revolutionary design changes.

Also, accessing an object structure is much easier than parsing an XML document, and makes future changes in the transformer easier.

The output of the transformer is the OutputItem. This object stores the output of the transformation, in other words one risk.

The last object is the Presenter, which is the object with which the user interacts. The Presenter has both a TransformationProfileManager that lets the user pick a profile and a transformer that transforms a user given XMI-file with the given transformation profile.

The Presenter takes a set of OutputItems and presents them to the screen. By using the AddRisk GUI Dialogue, the user can select an OutPutItem and add it to the risk-list. The separation of presentation and other logic is a common pattern for achieving a clean code base and easy extendability.

Also, by passing the analysis result in an OutputItem object. Both the Presenter and the Transformer may be altered without spawning changes in each other, as long as they both conform to the use of the OutputItem object.

5.3.1 Transformer usage

The Sequence Diagram in Figure 5.5 shows how the transformation works. When the Transformers Presenter is initialized, it creates a TransformationProfileManager and a Transformer. The Presenter uses this TransformationProfileManager to retrieve a list of all the diagram types the user analyze and presents this in a list.

The user selects this diagram type, and the Presenter retrieves and presents a list of all the profiles that is associated with the given diagram type. The user selects the profile he wants to use, selects the file where the model is stored, and starts the transformation.

The Presenter starts the transformation by supplying the filename and pro-

file id to the Transformer. The Transformer gets the transformation logic and parses the diagram into an object representation and starts the transformation. The output is returned to the Presenter and presented for the user.

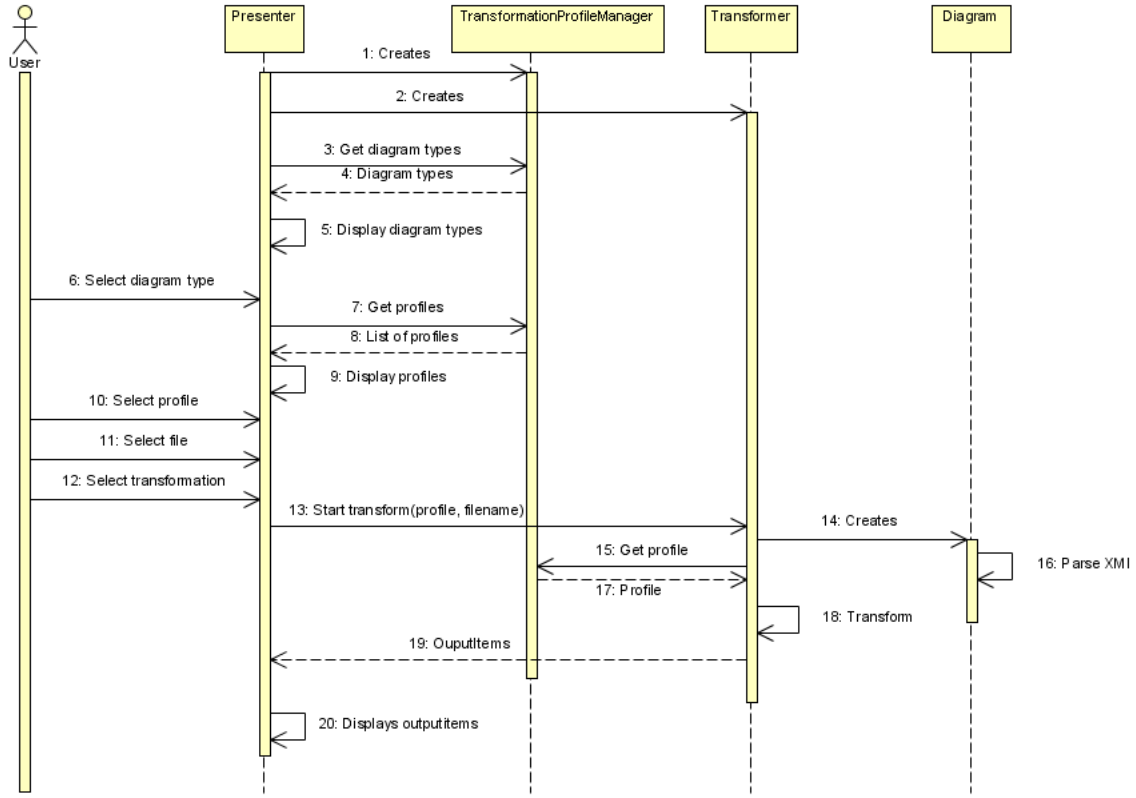

Figure 5.5: Transformer usage - Sequence Diagram.

5.4 The input

The transformer takes an XMI document as input, but it is parsed into an object representation of the diagram. Figure 5.4 shows the structure of these diagrams. There are four different kinds of diagrams, all of which extend the class *Diagram*. The Diagram base class contains methods that all diagrams share, that is, methods for extracting stereotype and tag value information from the XMI-documents.

Each diagram is decomposed into parts that extend the *DiagramElement* base class. This base class contains common methods for all UML elements

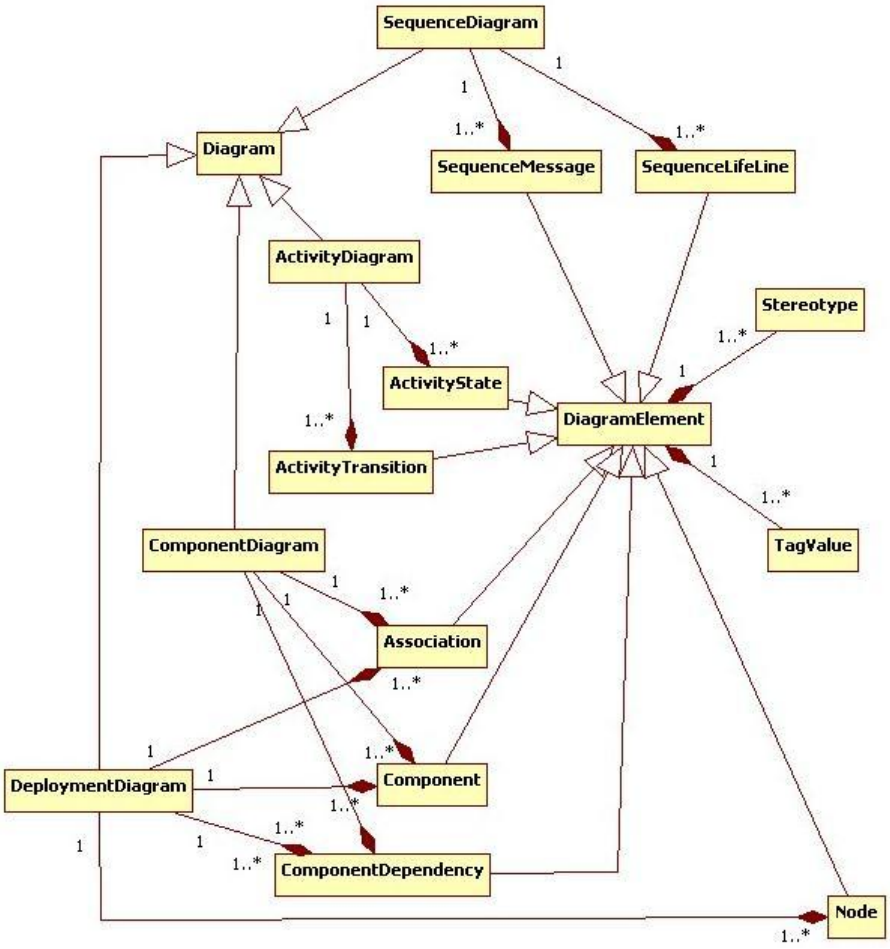

Figure 5.6: Diagram Structure.

such as methods for storing tag values and stereotypes.

A more detailed class diagram can be found in Appendix A.

5.5 Transformation Profile structures

Let us now see the difference between the two transformers. One transforms UML diagrams into PHA risk lists, and the other produces a HAZOP analysis of a UML diagram. The difference lies in the transformation profiles.

5.5.1 PHA profile

Figure 5.7 shows the structure of the PHA transformation profile as it is represented in the database file. Each of the entities depict a table in the database, and the connector represents the relations between them.

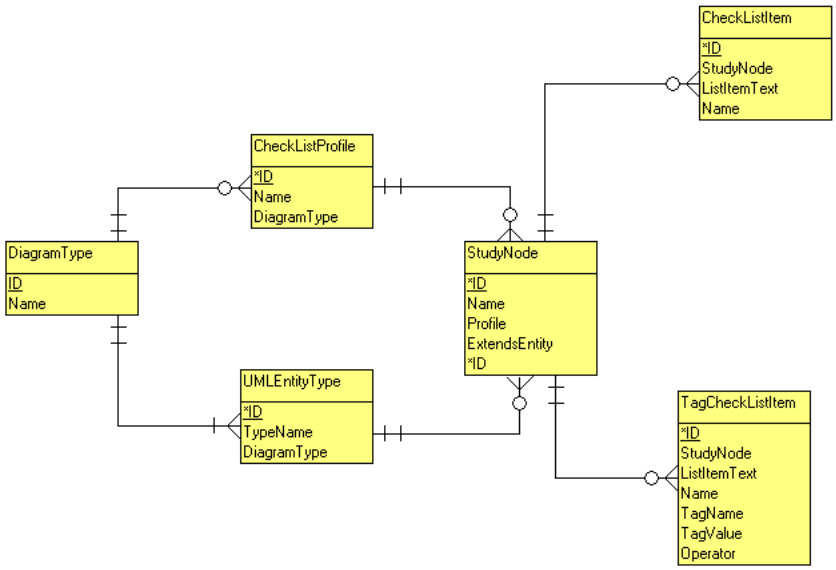

Figure 5.7: Database model of the PHA Transformation Profiles.

The PHA profile is stored in the *CheckListProfile*. Each profile is related to a *DiagramType*, which lets us sort profiles based on that given property.

A *CheckListProfile* has several *StudyNodes*. A *StudyNode* is a single part of the diagram. Therefore each *StudyNode* has a *UMLEntity* connected to it, which defines which entity the study node extends.

UMLEntity is associated with a *UMLDiagramType* so that we can ensure that the user doesn't create study nodes in a profile created for one diagram type from another diagram type.

The *name* attribute in the StudyNode table represents the name of the Stereotype that the given StudyNode is associated with.

The following example can illustrate the use. One study node could be of the UMLEntityType "Component", and the stereotype could be "ThirdPartyComponent" (See Figure 5.8).

Figure 5.8: An example study node.

For each StudyNode there can exist two kinds of associated piece of transformation logic.

1. Third party vendor goes out of business.
2. Third party vendor cuts support.

Pseudo code 22 $PHATransform$

Description: The code that PHA transforms a diagram into a list of risks.
Input: *diagram* - The diagram that is to be used.
Input: *profile* - The transformation profile that is to be applied.
Filename: PHATransformer.cs

```
1:  for each studynode in profile do
2:      for each diagramelement in diagram do
3:          if diagramelement equals studynode then
4:              for each checklistitem associated with studynode do
5:                  Add checklistitem to output
6:              end for
7:              for each tagchecklistitem associated with studynode do
8:                  if diagramelement constraint satisfies tagchecklistitem then
9:                      Add tagchecklistitem to output
10:                 end if
11:             end for
12:         end if
13:     end for
14: end for
```

The first is the CheckListItem, which is piece of information that is output if the diagram has the given study node. For example, if the diagram includes

a Component with the stereotype ThirdPartyComponjent, it could have the following associated CheckListItems

1. 3rd party vendor goes out of business.
2. 3rd party vendor cuts support.

The second kind is the TagCheckListItem. As with the CheckListItem, it is associated with a study node and output for each of the occurrences of that study node in the model. The difference is that its check list will only be output if the study node meets a constraint check on one of its tag values.

As described earlier, an entity in the UML diagram may not only be extended by creating new stereotypes, but one can also append a name/value pair called a tag value. This enables us to do a check on tag values. The supported operands are ">","<","!=" and "==".

The Studynode in Figure 5.8 can have the tag value "yearly license", which is either *true* or *false*. If this value is true, it means that the vendor has to be paid yearly for the use of the component.

E.g., one TagCheckListItem associated with this study node could then say

"If study node is a "ThirdPartyComponent", and has the tag "yearly license" with value *true*, then output "Forgetting the renewal of license"."

The PHA transformation is shown in Pseudo Code 22.

5.5.2 HAZOP profile

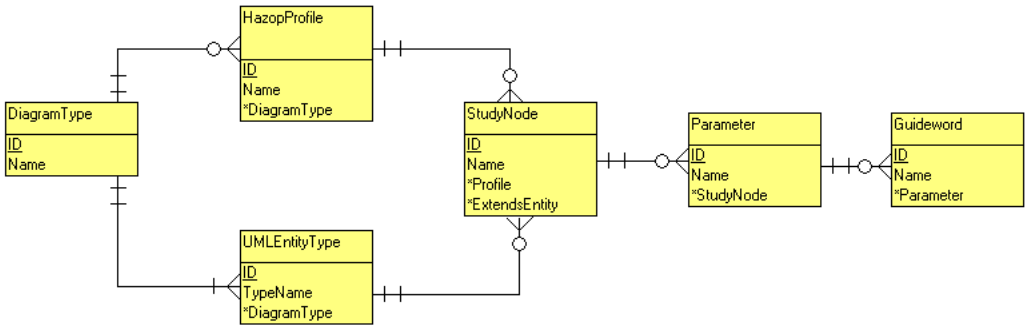

Figure 5.9: Database model of the HAZOP Transformation Profiles.

Figure 5.9 shows the structure of the HAZOP transformation profile. The

HAZOP follows the structure of the PHA, except in the structure of the logic connected to the study node.

Each study node has a set parameters connected to it. For example, if the study node is a Message with stereotype "Database Query", then a parameter could be "processing" or "communication". A parameter is a special view of the study node.

Each parameter has a set of guide words associated to it. As previously mentioned in the appendix, the HAZOP output is a study node merged with a parameter and a study node.

If "processing" had the guide words "no" and "less" associated with it, the HAZOP outputs to a "Data Query" named "GetInvoices", and its interpretation is shown in Table 5.5.2.

The transformation code is shown in Pseudo Code 23.

Study node	**Name**	**Parameter**	**Guide word**	**Interpretation**
Database Query	GetInvoices	Processing	no	The Invoice-DB does not process the query
Database Query	GetInvoices	Procesing	less	The Invoice-DB does not return all the data, or the processing is to slow

Table 5.1: Example HAZOP output.

Pseudo code 23 *HazopTranform*

Description: The code that HAZOP transforms a diagram into a list of risks.
Input: *diagram* - The diagram that is to be used.
Input: *profile* - The transformation profile that is to be applied.
Filename: HazopTransformer.cs

```
1:  for each studynode in profile do
2:      for each diagramelement in diagram do
3:          if diagramelement equals studynode then
4:              for each parameter associated with studynode do
5:                  for each guideword associated with parameter do
6:                      Add tagchecklistitem to output
7:                      Output studynode + parameter + guideword
8:                  end for
9:              end for
10:         end if
11:     end for
12: end for
```

Chapter 6

Enterprise UML Profiles

6.1 The UML parts

In this section I will sum up the four kinds of diagrams that was chosen to represent the enterprise and their subparts. When constructing a set of UML profiles, we may use these basic parts as extension points for new stereotypes which give the diagram more fidelity.

We draw our basic UML parts from two sources. The first is the Enterprise Analyzer which requires that the Enterprise is modeled by a set of basic UML parts. The other source is the Model Analyzer, which can do analysis on a range of basic UML types, many of which is not used in the to model the Enterprise Representation.

The challenge is then not to add too many parts that the Model Analyzer supports, because it will make the models harder to model. But yet add enough so that we can use the PHA and HAZOP analysis feature of the Model Analyzer usefully.

The Activity Diagram Figure 6.1 shows the parts of the UML Activity Diagram that are supported by the tools.

Only the Action and the Transition is supported by the Model Analyzer.

Sequence Diagram Figure 6.2 shows the parts of the UML Sequence Diagram that are supported by the tools.

Only the Lifeline and the Message is supported by the Model Analyzer.

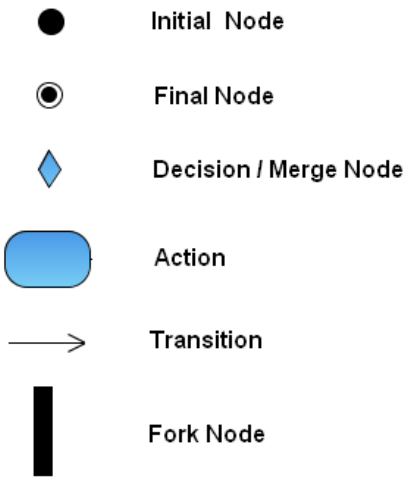

Figure 6.1: Activity Diagram - Parts.

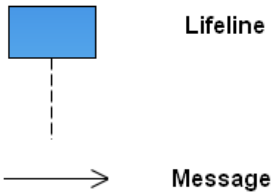

Figure 6.2: Sequence Diagram - Parts.

Component Diagram Figure 6.3 shows the parts of the UML Component Diagram that are supported by the tools.

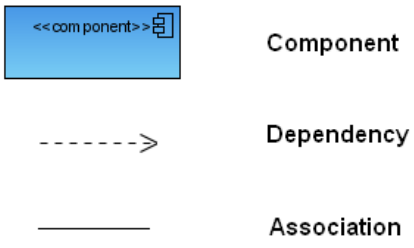

Figure 6.3: Component Diagram - Parts.

The Model Analyzer supports all of the parts in the figure.

Deployment Diagram Figure 6.4 shows the parts of the UML Deployment Diagram that are supported by the tools.

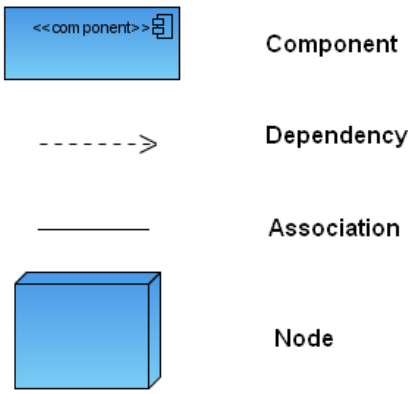

Figure 6.4: Deployment Diagram - Parts.

The Model Analyzer supports all of the parts in the figure.

6.2 Stereotype extensions

Given the parts in the previous section, I will extend the ones that make sense to extend.

6.2.1 Activity Diagram

At this level we are concerned with the business modeling level, and therefore the stereotypes should be types that make sense in the business domain.

The main parts of the activity diagram that is supported by the Model Analyzer are

- Action.
- Transition.

The Action represents a business process, and we can make the following extensions

1. Human.
2. Automated.
3. HumanAutomated.
4. NonCritical.

Within the enterprise, business functions can either be fully automated, fully done by a human, or done by a human interacting with an automated process. There is also a stereotype that indicates that the business function is NonCritical. It is much less work to mark all non-critical business processes "NonCritical", than marking all the critical with "Critical".

6.2.2 Deployment Diagram

At this level we are concerned with modeling of environments and systems in the enterprise, and therefore the stereotypes should be types that make sense in this domain.

The main parts of the Deployment Diagram that the Model Analyzer supports is:

- Node.
- Component.
- Dependency.

- Association.

Nodes represent the environments in the enterprise, and we can make the following extensions

- Mainframe.
- PC.
- Server.
- Operating System.
- Network.

The Component represents a system in the enterprise, and we can make the following extensions

- ApplicationSystem.
- DatabaseSystem.
- FileSystem.
- Firmware.

The Association represents a relation between nodes and components, in our domain this can be interpreted as how the parts communicate.

- Internet.
- LAN.
- WLAN.
- Serial.
- MessageBus.

I do not see a use for the *dependency* in our domain.

6.2.3 Component Diagram

At this level we are concerned with modeling of the internal structure of a system, and therefore the stereotypes should be types that make sense in this domain. The main parts of the Component Diagram is

- Component.
- Dependency.
- Association.

The Component represents a component in the enterprise, and we can do the following extensions:

- GUI.
- 3rdParty.
- CommunicationHandler.
- DatabaseHandler.
- Component

The Association between the components can be viewed as the means of communication between them, which gives us these extensions

- Internet.
- LAN.
- WLAN.
- Serial.
- MessageBus.

Likewise with the Deployment Diagram, I do not see a use for the dependency in this.

6.2.4 Sequence Diagram

At this level we are concerned with them modeling of interaction between systems and components, and therefore the stereotypes should be types that make sense in these two domains.

The main parts of the Sequence Diagram is

- Life Line.
- Message.

The Life Line represents a system or a component, at this modeling view it is not necessary to classify what kind of component or system it is, rather capture the interaction. The systems and components is more modeled with higher fidelity in the Component- and Deployment Diagram. Therefore, I see no need to extend this part.

The Message represents an interaction between two systems/components, or an action within a single system/component. Groth, [14], argues that the computer have four distinct operating modes.

1. IT systems process information.
2. IT systems store information.
3. IT systems communicate information.
4. IT systems can be configured.

Also, users may interact with the computer. We also need a stereotype that indicates if a SystemOperation or SubSystemOperation is non-critical, which gives us the following six extensions.

- Store.
- Process.
- Communication.
- Configuration.
- GUI Interaction.
- NonCritical.

6.2.5 Tag values

All the stereotypes that extend nodes (environments in the Enterprise Representation) and components (systems/components in the Enterprise Representation) may have the tags *MTBF* and *MTTR* which specifies the reliability and maintainability of the parts (See section 2.1.1). The stereotypes extending a UML component representing components in the enterprise may also has the tag value *parallel* to indicate that there are several independent components standing by to perform the functionality of the given component.

A DatabaseHandler could have the tags in Table 6.1.

Tag name	**Tag value**
MTBF	100
MTTR	1
Parallel	2

Table 6.1: Tagvalue examples.

Which indicate that there are two independent parallel DatabaseHandler components standing by, both which operate on average for 100 time units before failing and requiring one time unit to be repaired.

Chapter 7

Transformation Profiles

7.1 Transformation Profile - HAZOP

7.1.1 Business Scenario

Figure 7.1 shows the different extended stereotypes (the large bubbles) used to classify business functions, and beneath I have listed a set of parameters (the small bubbles) that I found relevant to the stereotype.

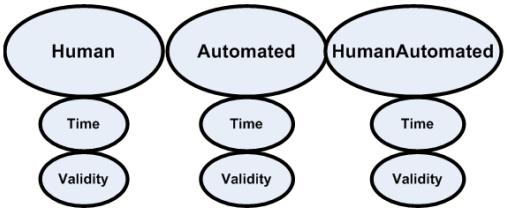

Figure 7.1: Action extended stereotypes and parameters.

7.1.2 (Sub) System Scenario

Figure 7.1 shows the different extended stereotypes used to classify (sub) system operations, and beneath I have listed a set of parameters I found relevant to the stereotype.

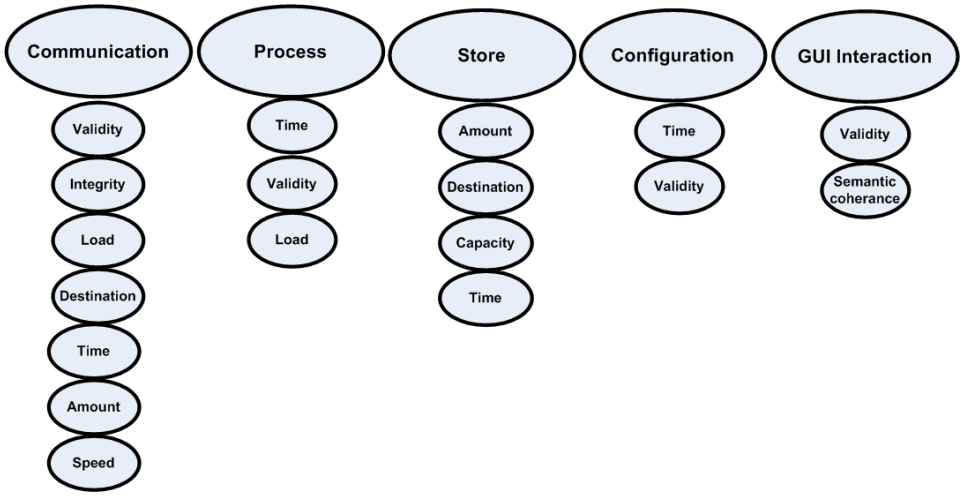

Figure 7.2: Message extended stereotypes and parameters.

7.1.3 Guide words

The parameters described previously are associated with the following generic guide words.

- No.
- More.
- Less.
- As well as.
- Before.
- Part of.
- Reverse of.
- Other than.
- Early.
- Late.
- After.

7.2 Transformation Profile - PHA

In the next sections I will propose a set of PHA profiles. Keep in mind that these profiles are mainly used for testing the PHA feature, and that, time permitted, a lot more time could have be spent making these profiles more comprehensive. These profiles will be extended by developing organizations that might use it for the purpose of analyzing enterprise solutions.

7.2.1 Business Scenario

The Transformation Profile for the Business Scenario is shown in Table 9.1.

Studynode	**Name**	**Description**
Human	Death	The human dies.
Human	Error	The human makes an error.
Human	No	The human doesn't do what he is intended to do.
Human	Wrong Time	The human acts at the wrong time.
Automated	Fail	The automated process fails.
Automated	Wrong Time	The automated process is performed at the wrong time.
HumanAutomated	Error	The human makes an error.
HumanAutomated	No	The human doesn't do what he is intended to do.
HumanAutomated	Wrong Time	The process is performed at the wrong time.

Table 7.1: System Structure - Action - PHA Check List Items.

7.2.2 Enterprise Structure

The PHA Transformation Profile for the Enterprise Scenario is found on Table 7.2, 7.3 and 7.4.

7.2.3 System Structure

The PHA Transformation Profile for the System Scenario is found on Table 7.5 and 7.3.

Studynode	Name	Description
Network	Down Time	The network is down.
Network	Overload	The network bandwidth overloads.
Network	Packet loss	The network has packet loss.
Network	Intrusion	Someone intrudes the network.
PC	Power	The PC loses power.
PC	Hardware failure	The hardware fails.
PC	Overload	The processing power is overloaded.
OperatingSystem	Halts	The OS halts.
OperatingSystem	Slows	The OS slows.
OperatingSystem	Leak	The OS suffers from a memory leak.
Server	Power	The server loses power.
Server	Hardware failure	The hardware fails.
Server	Overload	The processing power is overloaded.
Mainframe	Power	The server loses power.
Mainframe	Hardware failure	The hardware fails.
Mainframe	Overload	The processing power is overloaded.

Table 7.2: Enterprise Structure - Node - PHA Check List Items.

Studynode	Name	Description
LAN	Loss	The LAN has high packet loss.
LAN	Overload	The bandwidth is overloaded.
LAN	No	The LAN is breaks down.
WLAN	Low	The signal strength is too low.
WLAN	Range	The unit is out of range.
WLAN	Unsecured	The connection is unsecured and intruded.
WLAN	Loss	The WLAN has high packet loss.
WLAN	Overload	The bandwidth is overloaded.
Serial	No	The serial connection goes down.
Serial	Overload	The connection is overloaded.
MessageBus	Wrong	Wrong message is sent.
MessageBus	No	No messages are sent.
MessageBus	Few	The messages are sent to too few recipients.
MessageBus	Delay	The messages are delayed.

Table 7.3: Enterprise Structure/System Structure - Association - PHA Check List Items.

Studynode	Name	Description
DatabaseSystem	Down	The database is down.
DatabaseSystem	Overload	The database is overloaded.
DatabaseSystem	Corrupted	The database is corrupted.
DatabaseSystem	Intrusion	The database is intruded.
DatabaseSystem	Lost	Data in the database is lost.
FileSystem	Down	The file system is down.
FileSystem	Intrusion	The file system is intruded.
FileSystem	Corrupted	The file system is corrupted.
ApplicationSystem	Down	The application system is down.
ApplicationSystem	Denial	The application system denies service.
Firmware	Damaged	The firmware is damaged.
Firmware	Outdated	The firmware is outdated.

Table 7.4: Enterprise Structure - Component - PHA Check List Items.

Studynode	Name	Description
GUI	Wrong Input	The user enters wrong input.
GUI	Wrong Output	The GUI shows wrong output.
3rdParty	No Support	The 3rd party vendor cuts support.
CommunicationHandler	Overload	The component can't handle the communication load.
CommunicationHandler	Break down	The component breaks down.
DatabaseHandler	Wrong retrieval	The component retrieves wrong data.
DatabaseHandler	Wrong write	The component writes wrong data.
DatabaseHandler	Break down	The component breaks down.
Component	Break down	The component breaks down.

Table 7.5: System Structure - Component - PHA Check List Items.

Chapter 8

The Extended RUP Process

8.1 Introduction

8.2 The process

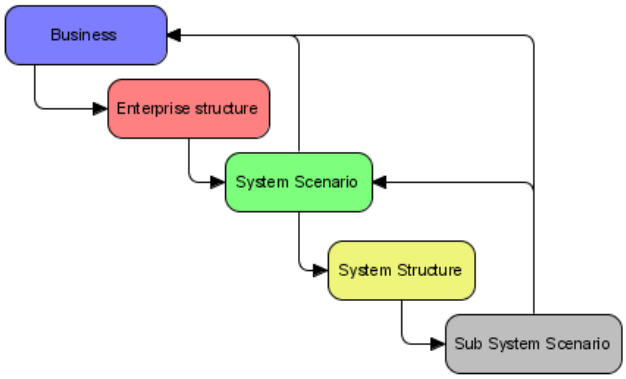

Figure 8.1: The main structure of the process.

The processes are divided into five distinct activities, which are shown in Figure 8.1.

The *Business* activity is the analysis and modeling of business processes, and is the first the activity in the process. When one or more business processes have been modeled, the next activity is the *Enterprise Structure*, which is the modeling and analysis of the systems and environments in the enterprise.

The next activity is *System Scenario*, where business functions in business processes are refined by creating system scenarios.

After this, one can return to the *Business* activity to create more business processes, or one can continue to refine the system scenario into sub system scenarios. The first step in this is the *System Structure* activity. It deals with the modeling of the structural composition of systems, which is followed by the *Sub System Scenario* activity which shows the interaction between the subparts (components) of the System. Then one can either continue by creating more system scenarios or business processes. The processes are repeated until the solution is satisfactory modeled and analyzed. Each of the activities will be described in the next sections.

8.3 Business activity

The Business activity is shown in Figure 8.2. The first step is to identify a business process and model it. When the model is defined, it is analyzed by the Model Analyzer tool. If this analysis calls for changes in the model, these are implemented. The processes go in a model/analysis-loop until the model is found satisfactory. The last step is to feed the model into the Enterprise Analyzer tool to start the creation of an enterprise representation. The whole procedure is repeated as long as there are more business processes to be modeled.

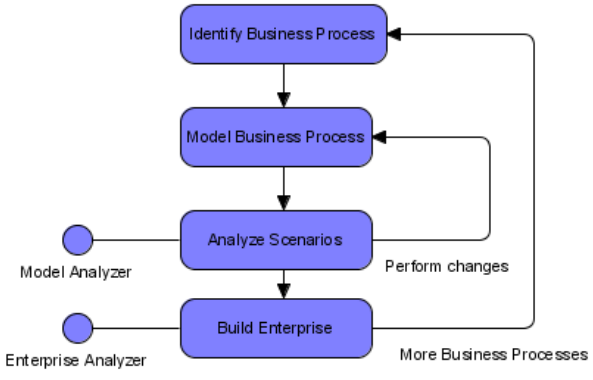

Figure 8.2: Business activity.

8.4 Enterprise Structure activity

The Enterprise Structure activity is shown in Figure 8.3. The activity is started by modeling the enterprise structure. This means identifying the environments and systems that are needed to realize the business functions. Once the structure is modeled, it is analyzed using the Model Analyzer tool. The processes go into a model/analysis cycle until they are found satisfactory. Lastly the model is fed into the Enterprise Analysis tool to append it to the enterprise representation.

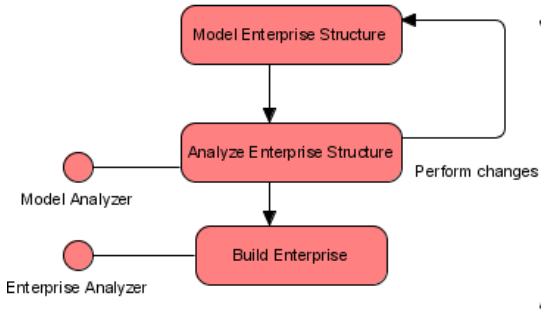

Figure 8.3: Enterprise Structure activity.

8.5 System Scenario activity

The System Scenario activity is shown in Figure 8.4. The first step is to select a business function that is going to the be refined by a system scenario. The scenario is modeled and analyzed and then fed into the Enterprise Analysis tool to grow the enterprise representation.

8.6 System Structure activity

The System Structure activity is shown in Figure 8.5. A system is selected and modeled by decomposing it into components. The result is analyzed and then fed into the enterprise representation with the Enterprise Analysis tool. This is repeated for as long as there are more systems to model and analyze.

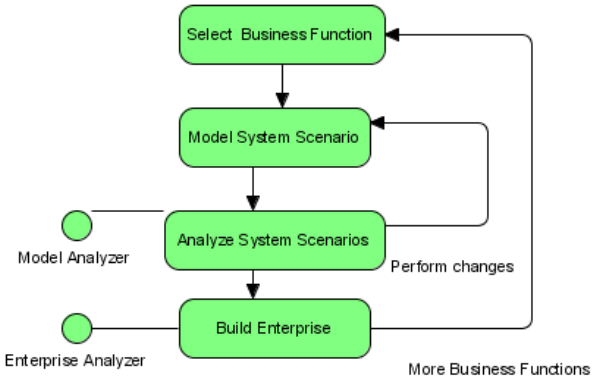

Figure 8.4: System Scenario activity.

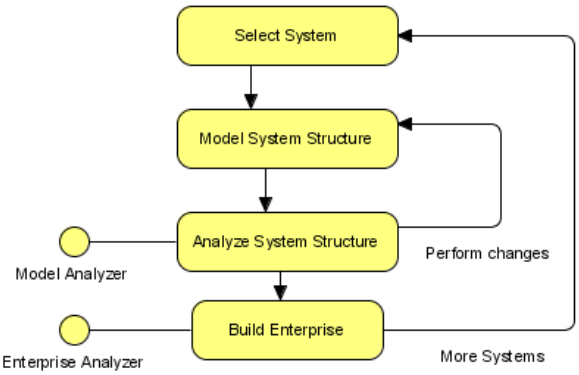

Figure 8.5: System Structure activity.

8.7 Sub System Scenario activity

The Sub System Scenario activity is shown in Figure 8.6. A system operation is selected to be refined by a sub system scenario. When the sub system scenario is fully modeled and analyzed, it is fed into the Enterprise Analysis tool. This procedure is repeated until there are no more system operations to realize.

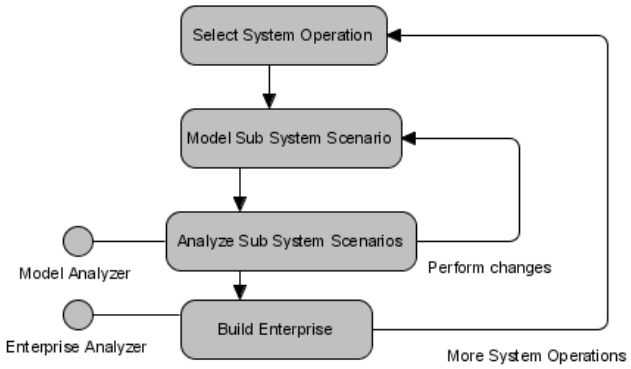

Figure 8.6: Sub System Scenario activity.

8.8 Risk analysis activity

Each use of the Model Analyzer should follow a simple process (Figure 8.7) that is taken from [11]. This process is more explained in more detail in Appendix B.1.3.

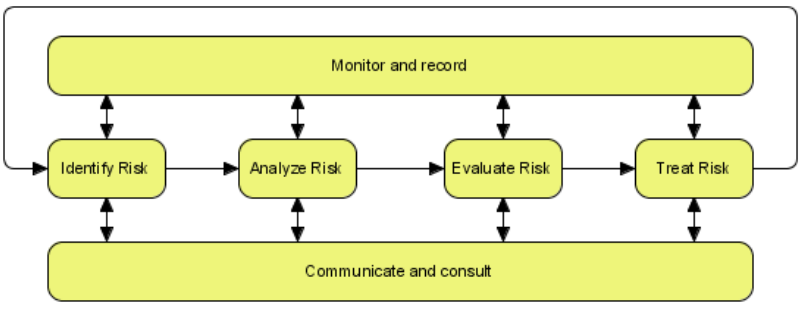

Figure 8.7: The Risk Analysis activity.

Chapter 9

Test of the Tools

9.1 The business processes

To demonstrate and test the methods, I have constructed a case scenario. The case is an enterprise system at a newspaper consisting of the following six departments.

Printing Department The main business process in the Printing department is the printing of the paper.

Inventory Department The main business processes in this department is administrating the inventory, this means tasks such as purchase, internal delivery etc.

Administration Department The main process in this department is to manage subscriptions.

Publishing Department The business process in this department is creation of the contents of the newspaper.

Transport Logistics Department The main process in this department is delivering the newspaper to drop off points.

IT Department The main processes of the IT department is hosting the online edition of the newspaper.

This gives us the seven business processes shown in Table 9.1. I have also assigned each process a criticality value.

#	**Description**	**Criticality**
1	Print paper.	300
2	Purchase supplies.	50
3	Add subscribers.	20
4	Remove subscribers.	10
5	Create newspaper content.	300
6	Deliver newspapers.	300
7	Host online edition.	75

Table 9.1: Business processes and criticality.

Based on the UML Profile defined in Chapter 6, I will now model this enterprise system, following the process described in Chapter 8.

The first step is to create business work flow diagrams for each of these business processes.

Print paper

Figure 9.1 shows the business scenario for the *Print Paper* business process.

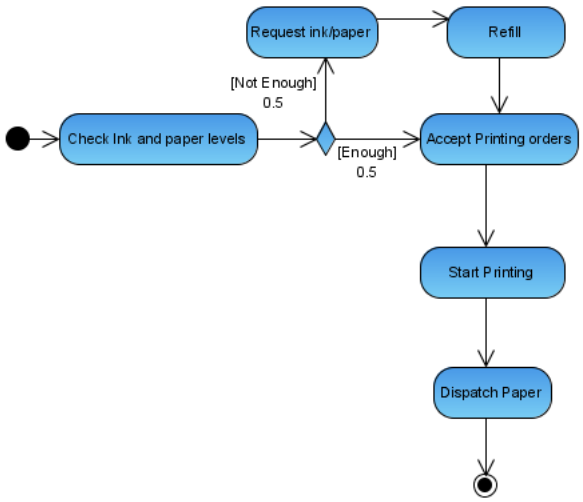

Figure 9.1: Print Paper Business Scenario.

The Print Manager starts by checking if there is enough ink and paper

in printing department to perform the print. If there is not enough, the manager requests more supplies from the Inventory Department, awaits their arrival, and then refills. The manager then awaits the order to start the printing from the Publishing Department. After the printing is finished, he dispatches the paper by noticing the Transport Logistics Department that the paper is ready for transport.

Purchase supplies

Figure 9.2 shows the business scenario for the *Purchase Supplies* business process.

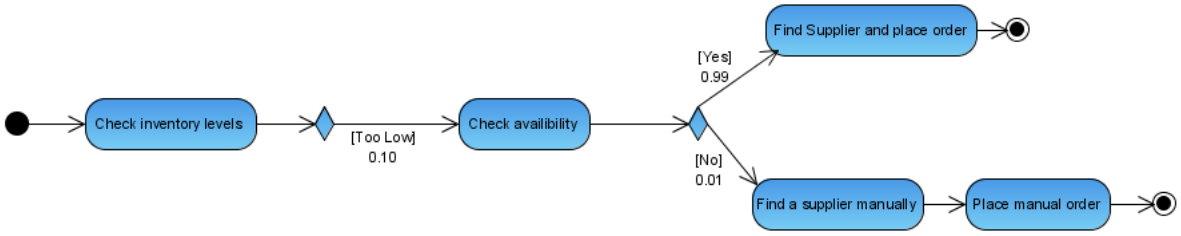

Figure 9.2: Purchase Supplies Business Scenario.

First, the stock levels are checked at a specified time interval. If some of the levels are to low, the availability of the supplies at electronic stores are checked. If the supplies are available, an electronic order is placed at the supplier with the lowest cost or delivery time. If there is no availability in the electronic stores, the workers need to find a supplier manually and place the order.

Add subscriber

Figure 9.3 shows the business scenario for the *Add Subscriber* business process. There are three ways subscribers can order a subscription, either through letter and telephone, where the subscription is added manually, or through the internet, where the subscription is added automatically.

Remove subscriber

Figure 9.4 shows the business scenario for the *Add Subscriber* business process. With background in the *Add Subscriber* process, it is self-explained.

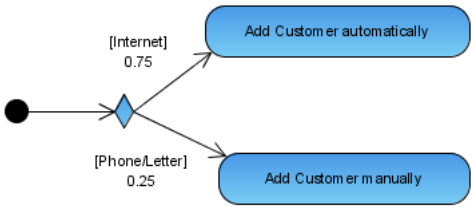

Figure 9.3: Add Subscriber Business Scenario.

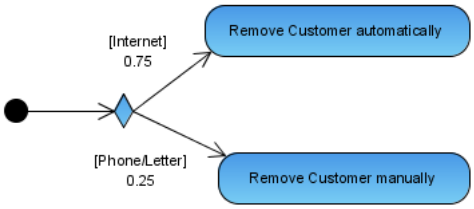

Figure 9.4: Remove Subscriber Business Scenario.

Create newspaper contents

Figure 9.5 shows the business scenario for the *Create newspaper contents* business process. The journalists start by writing the articles, after which layout managers merge these articles with ads to create the newspaper contents.

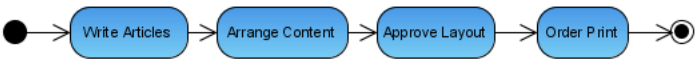

Figure 9.5: Create newspaper content Business Scenario.

The final layout is approved by the editor, and the Printing Department is told that they can start the printing of the newspaper.

Deliver newspaper

Figure 9.6 shows the business scenario for the *Deliver newspaper* business process.

Figure 9.6: Deliver newspaper Business Scenario.

The manager of the department receives a notice that the paper is ready for delivery, and a set of drop-off lists are created and printed. A drop-off list is a delivery instruction for a delivery driver. The lists are created in such a way that they minimize the amount of time needed to deliver all the newspapers to their designated recipients.

Host online edition

Figure 9.7 shows the business scenario for the *Host online edition* business process.

Figure 9.7: Host online edition Business Scenario.

It is very simple, it is only the function of hosting the online edition such that a remote web user may access it.

9.1.1 The Enterprise Structure

The next step is to define the structure of the enterprise that is going to handle these business processes. This is done in Figure 9.8.

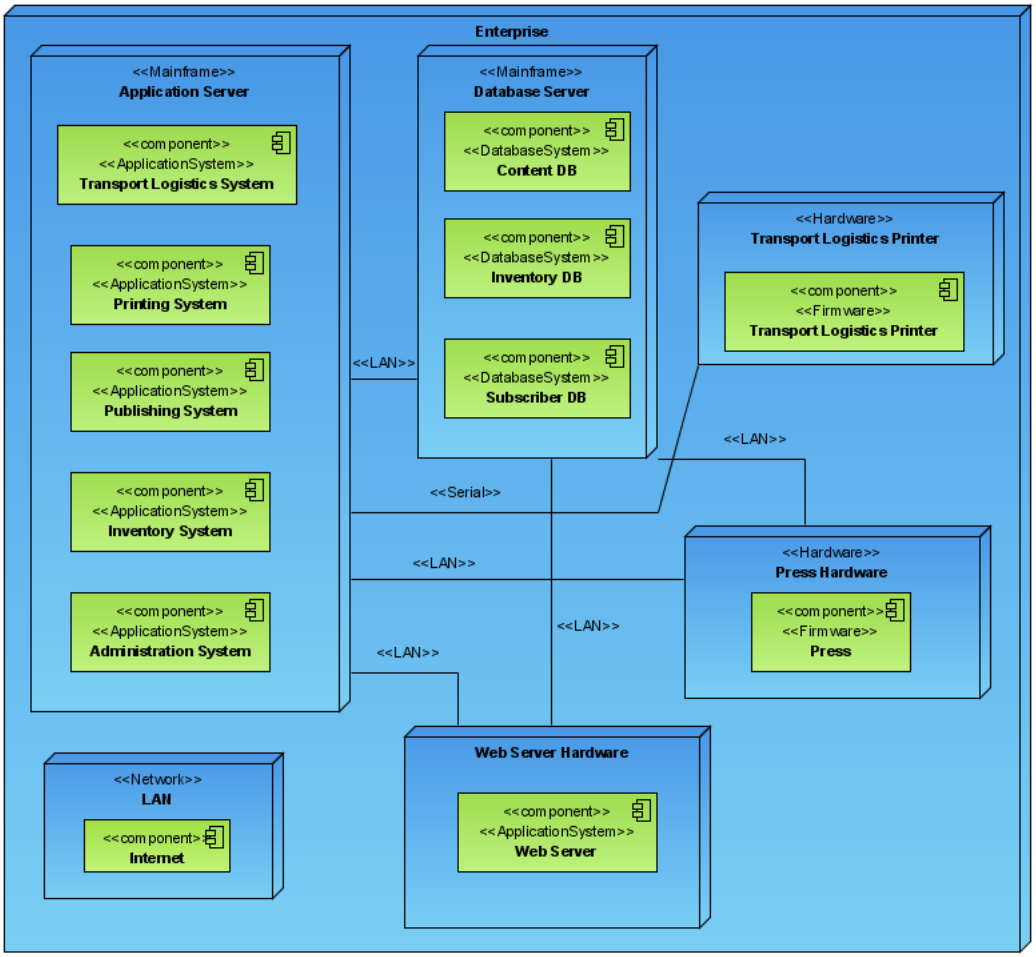

Figure 9.8: Newspaper Enterprise Structure.

Table 9.2 shows the reliability and maintainability in the enterprise.

I have not specified a MTBF or MTTR number to the systems as a whole, but will specify this information for each component in the systems.

Entity	MTBF	MTTR
Database Server	1000	4
Application Server	1000	4
Inventory DB	100	3
Content DB	100	3
Subscriber DB	100	3
Press Hardware	10000	10
Press	20000	50
Administration System	-	-
Inventory System	-	-
Printing System	-	-
Publishing System	-	-
Transport Logistics System	0	0
Webserver	1000	1
Webserver Hardware	1000	4
Internet	10000	1
LAN	30000	2

Table 9.2: Reliability and Maintainability In The Enterprise.

9.1.2 System Scenarios

Printing the paper The business functions in the "Printing the paper" process are refined in Figure 9.9.

Purchase Supplies The business functions in the "Purchase Supplies" process are refined in Figure 9.10.

Add Subscriber The business functions in the "Add Subscriber" process are refined in Figure 9.11.

Remove Subscriber The business functions in the "Remove Subscriber" process are refined in Figure 9.12.

Create newspaper content The business functions in the "Create newspaper content" process are refined in Figure 9.13.

Deliver newspaper The business functions in the "Deliver newspaper" process are refined in Figure 9.14.

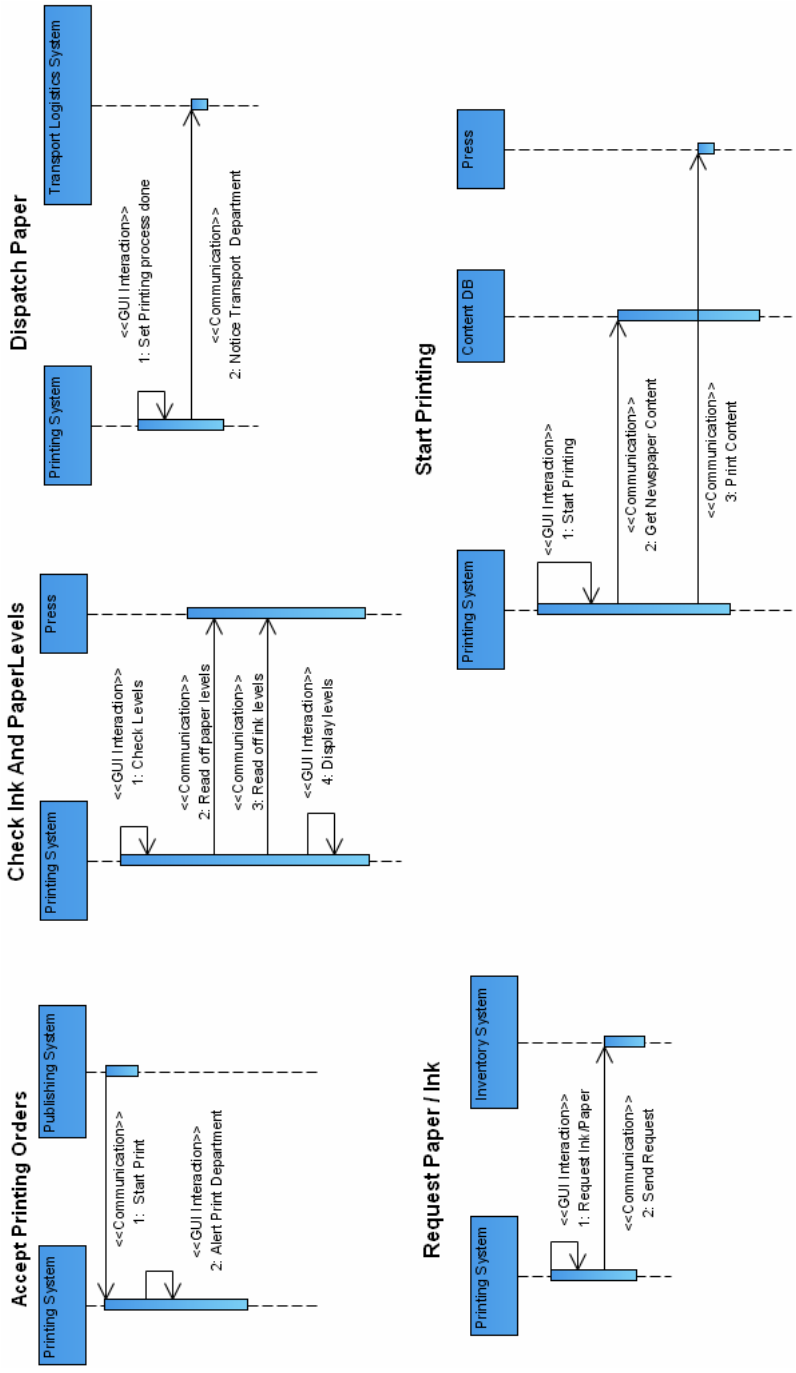

Figure 9.9: Printing the paper - Systems scenarios.

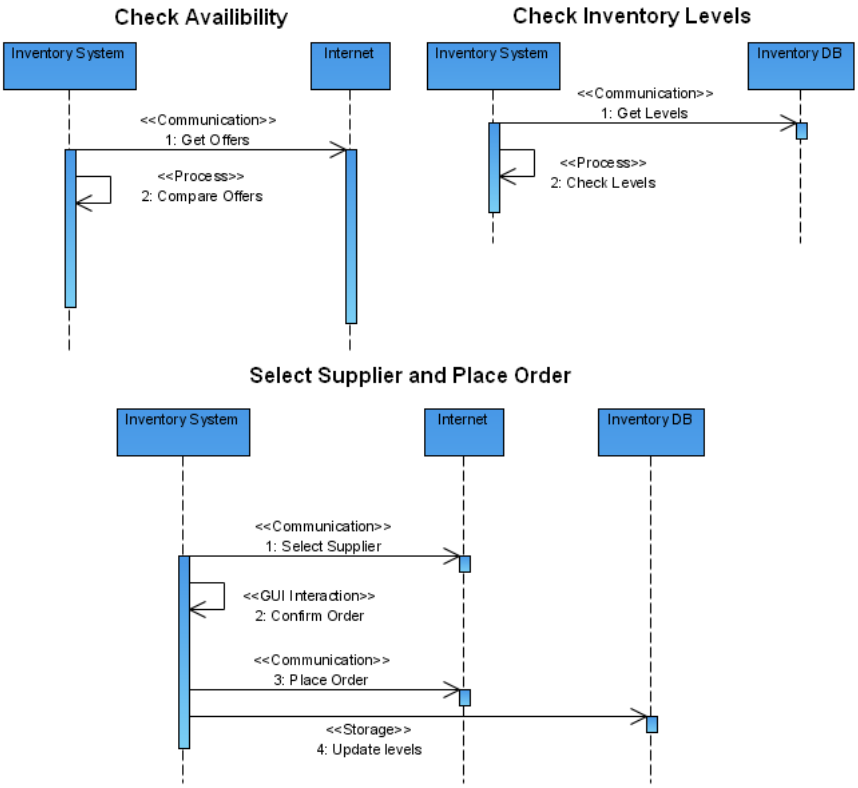

Figure 9.10: Purchase supplies - Systems scenarios.

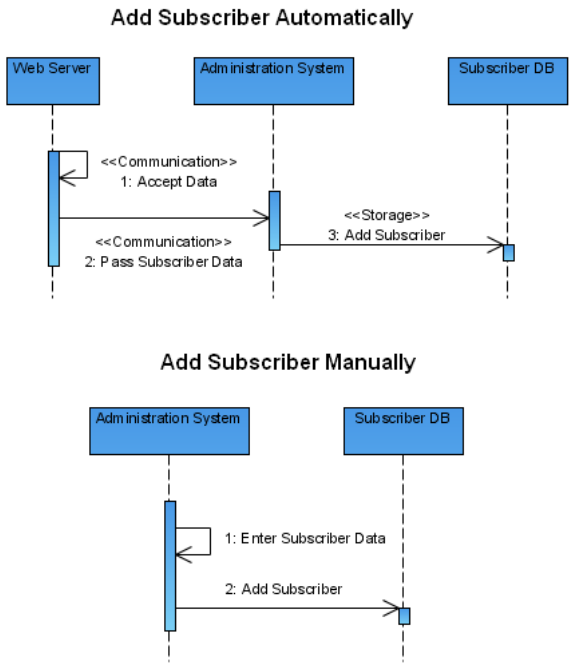

Figure 9.11: Add Subscriber - Systems scenarios.

Host online edition The business functions in the "Host online edition" process are refined in Figure 9.15.

9.1.3 System Structure and Sub System Scenarios

To limit the scope of the case enterprise, I will only model the enterprise at sub system level on one system operation. This system operation is the "Get Content", which refines the "Start Printing" business function. The sub system operations of "Get Content" and the system structure of the systems involved in this system operation are shown in Figure 9.16.

The reliability and maintainability of the components in the two systems is shown in Table 9.3 and 9.4,

Entity	**MTBF**	**MTTR**
Communication Handler	10000	1
Print Controller	20000	1

Table 9.3: Reliability and maintainability in Press.

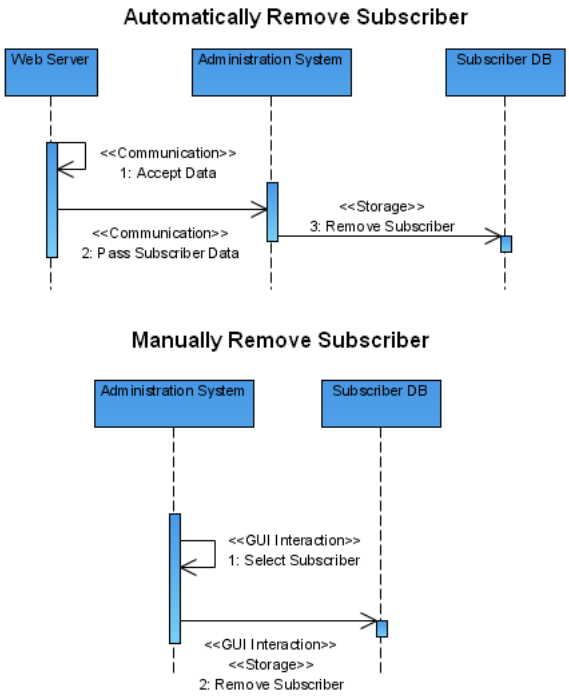

Figure 9.12: Remove Subscriber - Systems scenarios.

Entity	**MTBF**	**MTTR**
Message Handler	10000	1
DB Handler	20000	1
Press Handler	15000	1
GUI	20000	1

Table 9.4: Reliability and maintainability in the Printing System.

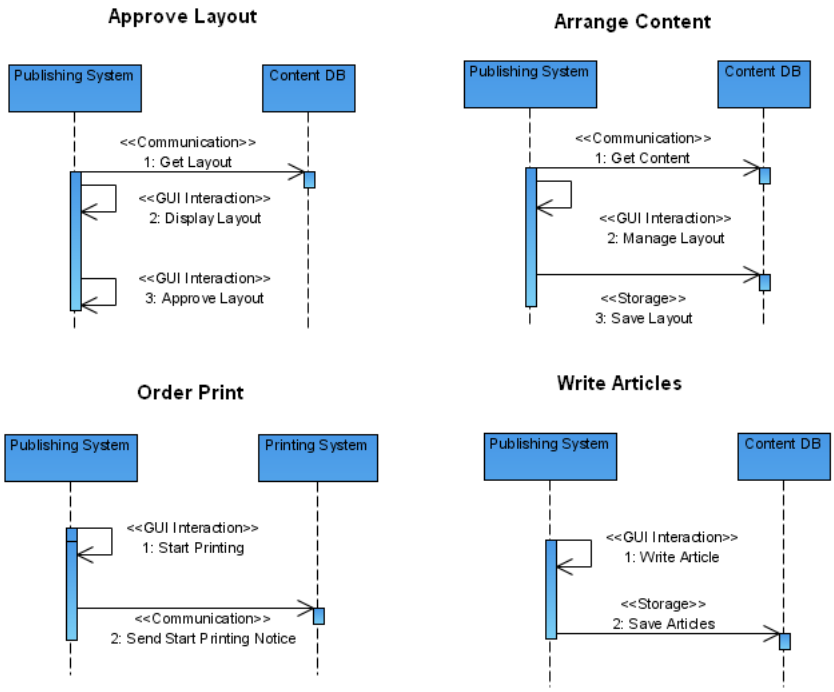

Figure 9.13: Create newspaper content - Systems scenarios.

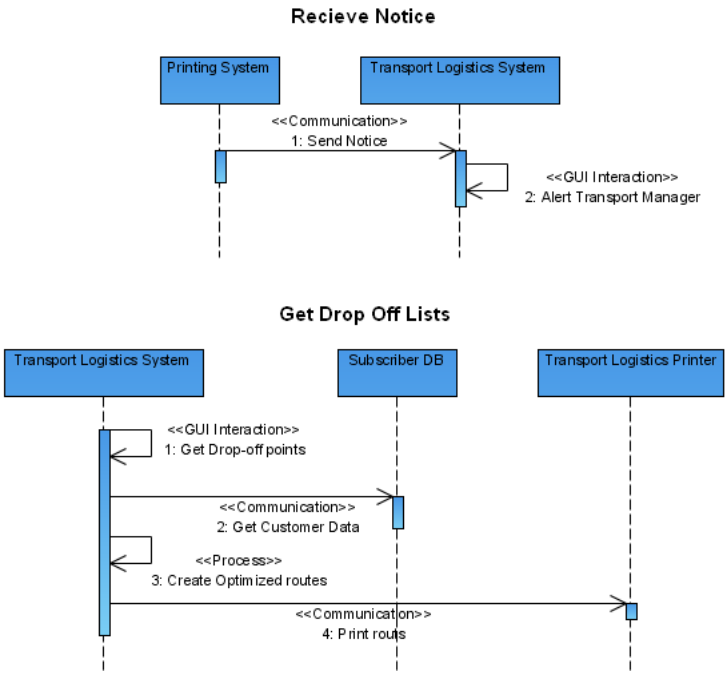

Figure 9.14: Deliver Newspaper - Systems scenarios.

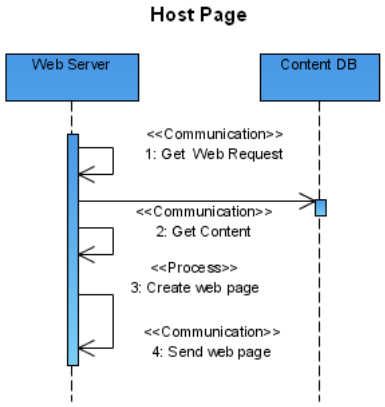

Figure 9.15: Host online edition - Systems scenarios.

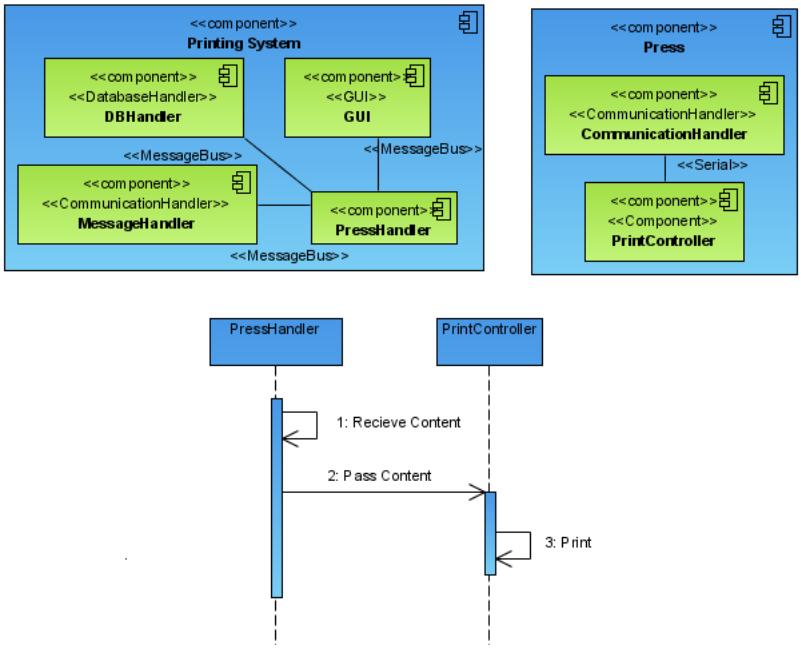

Figure 9.16: Example System Scenario and System Structure.

9.2 Testing the Enterprise Analyzer

I loaded all of the models above into the Enterprise Analyzer and got the analysis result shown in Figure 9.17.

9.3 Testing the Model Analyzer

9.3.1 Testing the HAZOP Analyzer

I tested the HAZOP analysis feature on the "Start printing" system scenario.

Table 9.5 shows the result. The output items that was not interpreted since a risk is omitted.

9.3.2 Testing the PHA Analyzer

I tested the PHA analysis feature on the Enterprise Structure (Figure 9.8). Table 9.6 and Table 9.7 shows the result.

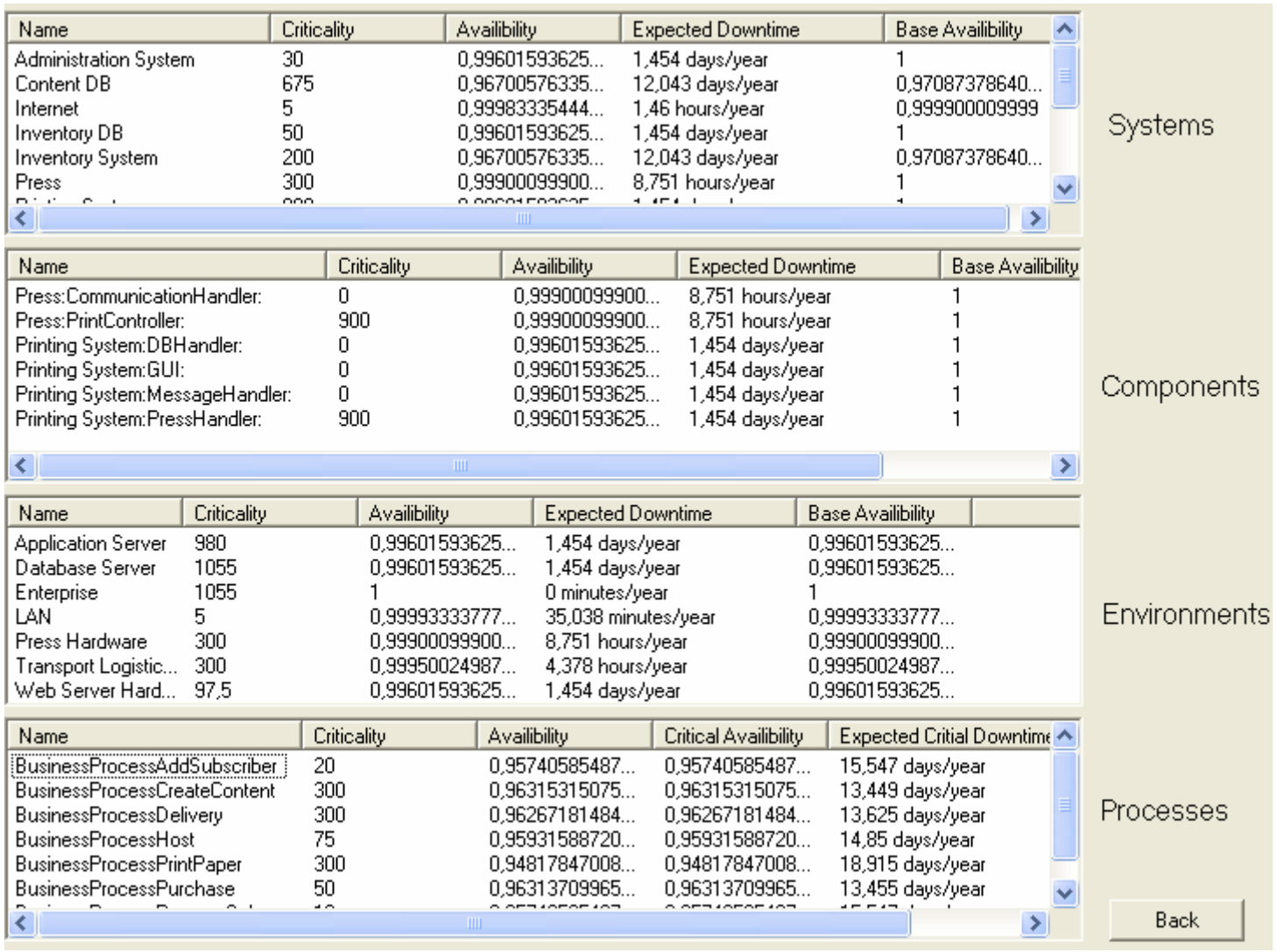

Figure 9.17: Enterprise Analysis result.

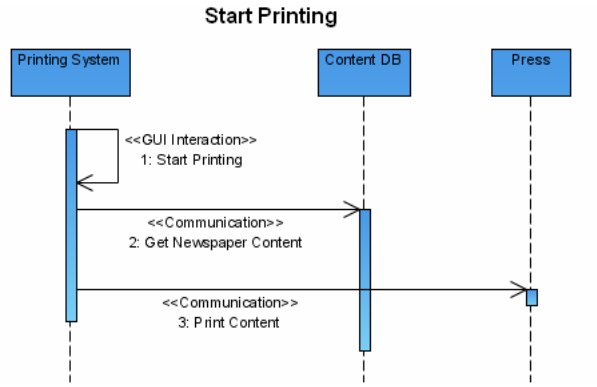

Figure 9.18: Start Printing - System Scenario.

Studynode	Type	Parameter	Guideword	Interpretation	Consequence	Likelihood	Solution
Get Newspaper Content	Communication	Integrity	Less	The data content data is not complete.	Moderate	Unlikely	Have the print manager look over the first newspaper that is printed.
Get Newspaper Content	Communication	Load	No	The data cannot be retrieved from the database.	Moderate	Possible	Transfer the content via a CD from the content database and the printing system.
Print Content	Communication	Validity	No	The printing is started before the content is ready.	Moderate	Possible	Have the print manager look over the first newspaper that is printed.
Print Content	Communication	Destination	No	The notice is never received.	Major	Possible	Have the Printing System send a confirmation message that is has received the notice.
Start Printing	GUI Interaction	Validity	No	The print manager starts printing without content being ready.	Moderate	Rare	Accept risk.

Table 9.5: HAZOP test on “Start printing” System Scenario.

Studynode	Name	Description
Database Server	Overload	The processing power is overloaded.
Database Server	Hardware failure	The hardware fails
Database Server	Power	The server loses power
Application Server	Overload	The processing power is overloaded.
Application Server	Hardware failure	The hardware fails
Application Server	Power	The server loses power
LAN	Down time	The network is down.
LAN	Overload	The network bandwidth overloads.
LAN	Packet loss	The network has packet loss.
LAN	Intrusion	Someone intrudes the network.
Printing System	Down	The application system is down.
Printing System	Denies	The application system denies service.
Publishing System	Down	The application system is down.
Publishing System	Denies	The application system denies service.
Inventory System	Down	The application system is down.
Inventory System	Denies	The application system denies service.
Transport Logistics System	Down	The application system is down.
Transport Logistics System	Denies	The application system denies service.
Administration System	Down	The application system is down.
Administration System	Denies	The application system denies service.
Web Server	Down	The application system is down.
Web Server	Denies	The application system denies service.
Content DB	Down	The DB is down
Content DB	Overload	The DB is overloaded
Content DB	Corrupted	The DB is corrupted
Content DB	Intrusion	The DB is intruded
Content DB	Lost	Data in the DB is lost.

Table 9.6: PHA test on Enterprise Structure - I.

Studynode	Name	Description
Inventory DB	Down	The DB is down
Inventory DB	Overload	The DB is overloaded
Inventory DB	Corrupted	The DB is corrupted
Inventory DB	Intrusion	The DB is intruded
Inventory DB	Lost	Data in the DB is lost.
Subscriber DB	Down	The DB is down
Subscriber DB	Overload	The DB is overloaded
Subscriber DB	Corrupted	The DB is corrupted
Subscriber DB	Intrusion	The DB is intruded
Subscriber DB	Lost	Data in the DB is lost.
Press	Damaged	The firmware is damaged
Press	Outdated	The firmware is outdated
Transport Logistics Printer	Damaged	The firmware is damaged
Transport Logistics Printer	Outdated	The firmware is outdated
From: Application Server To: Database Server	Loss	The LAN has high packet loss.
From: Application Server To: Database Server	Overload	The bandwidth is overloaded.
From: Application Server To: Database Server	No	The LAN is down.
From: Web Server Hardware To: Application Server	Loss	The LAN has high packet loss.
From: Web Server Hardware To: Application Server	Overload	The bandwidth is overloaded.
From: Web Server Hardware To: Application Server	No	The LAN is down.
From: Application Server To: Press Hardware	Loss	The LAN has high packet loss.
From: Application Server To: Press Hardware	Overload	The bandwidth is overloaded.
From: Application Server To: Press Hardware	No	The LAN is down.
From: Web Server Hardware To: Database Server	Loss	The LAN has high packet loss.
From: Web Server Hardware To: Database Server	Overload	The bandwidth is overloaded.
From: Web Server Hardware To: Database Server	No	The LAN is down.
From: Press Hardware To: Database Server	Loss	The LAN has high packet loss.
From: Press Hardware To: Database Server	Overload	The bandwidth is overloaded.
From: Press Hardware To: Database Server	No	The LAN is down.
From: Transport Logistics Printer To: Application Server	No	There is no connection.
From: Transport Logistics Printer To: Application Server	Overload	The connection is overloaded

Table 9.7: PHA test on Enterprise Structure - II.

Chapter 10

User Manuals

10.1 The Enterprise Analyzer

Figure 10.1 shows the main screen of the Enterprise Analyzer application. From this screen we can build a complete representation of the enterprise.

The first step in building the Enterprise is to load the enterprise environment. This contains structural information about the environments and systems in the enterprise. This is done by pressing "Add Environment" button in Figure 10.1.

You will presented by a dialogue for selecting the XMI-file where this information is stored (Figure 10.2). Select the appropriate file and press "OK".

This will take you back to the main screen, and the structural view of the enterprise will be updated and look something like Figure 10.3. It will show all the environments and their systems.

The next step is to load the components in the enterprise. This is done by pressing "Add Components" button in Figure 10.1. You will then be presented with a new dialog for selecting XMI-files as shown in Figure 10.2. You can select multiple files.

After this you will be taken back to the main screen, and the structural view is again updated and show the components relation with the systems (Figure 10.4).

The next step is to add a business processes. The first thing is to enter a name for the business process and assign a criticality score to it. See Figure 10.5. Then click "Add Functions" which takes you to a dialogue for selecting an XMI-file (Figure 10.6). Select the file that contains the business scenario for the given business process and press "Open".

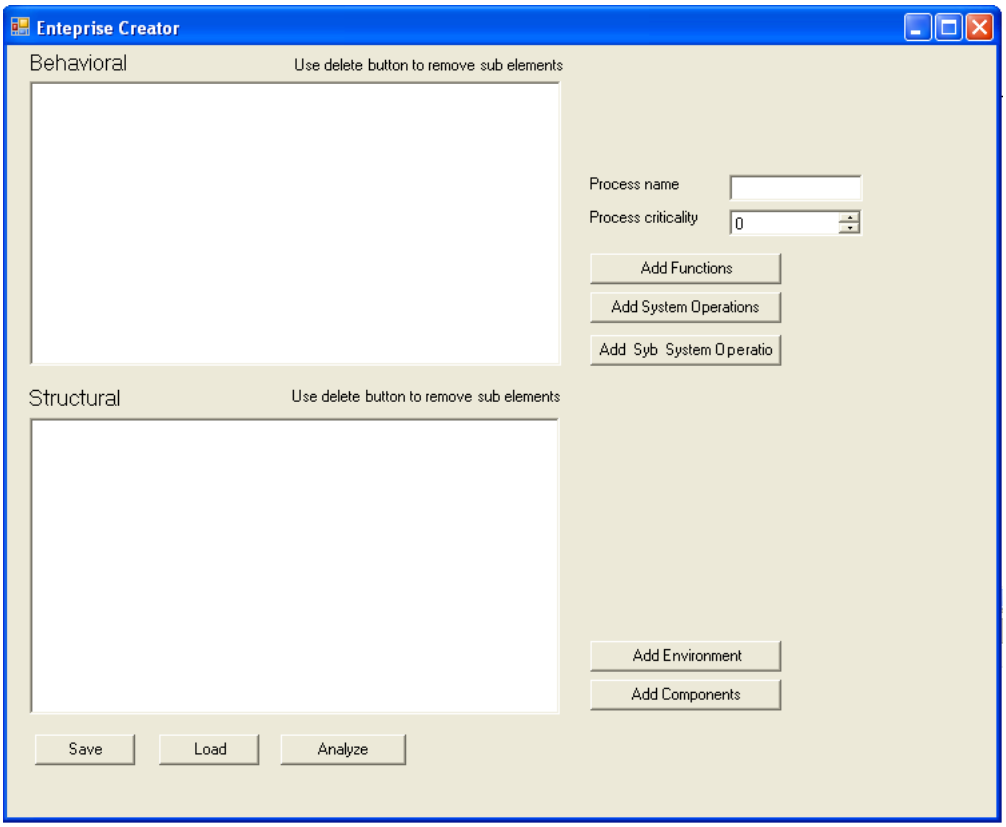

Figure 10.1: Enterprise Analyzer - Main menu.

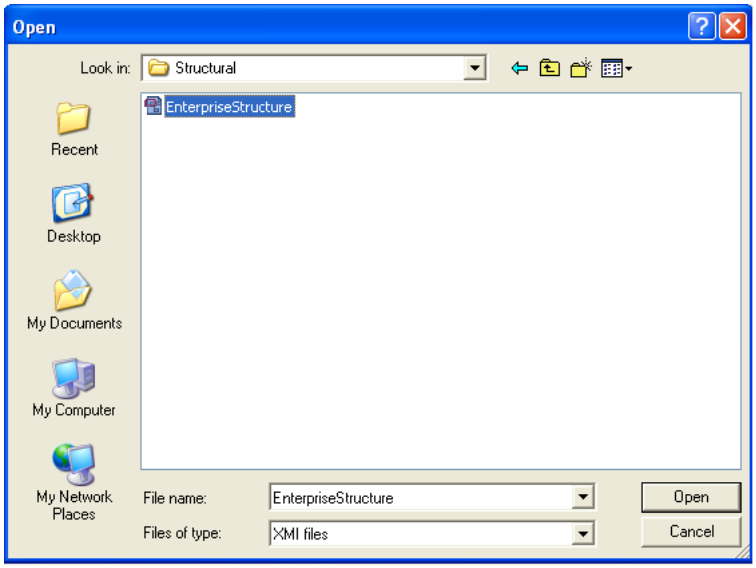

Figure 10.2: Enterprise Analyzer - Adding Environment.

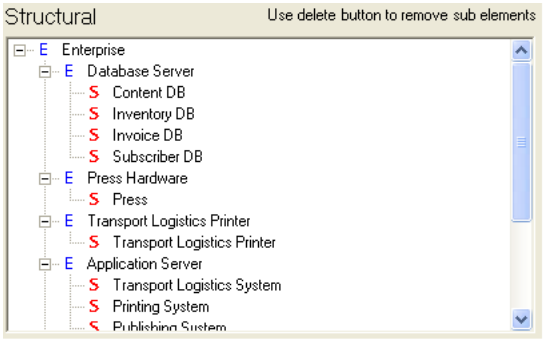

Figure 10.3: Enterprise Analyzer - Structural View.

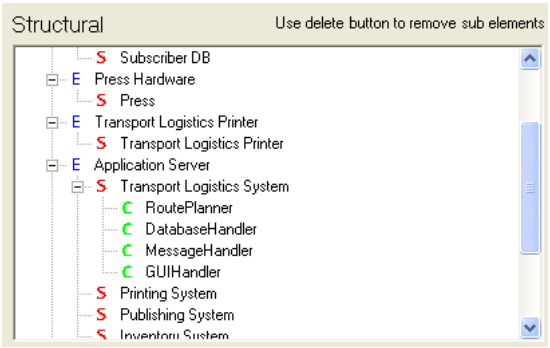

Figure 10.4: Enterprise Analyzer - Structural view II.

Figure 10.5: Enterprise Analyzer - Configuring a process.

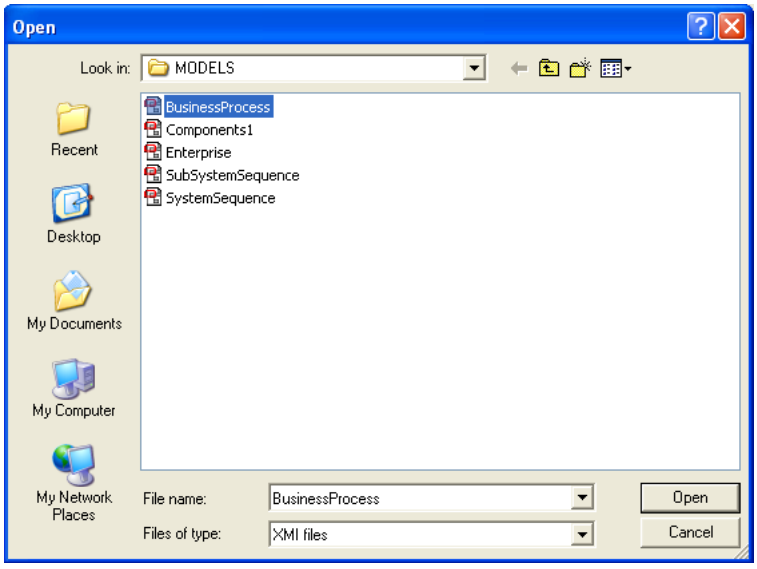

Figure 10.6: Enterprise Analyzer - Load Business Scenario.

Again, you will be taken back to the main menu and the behavioral view is updated and contains all the business functions that are used by the business process. See Figure 10.7.

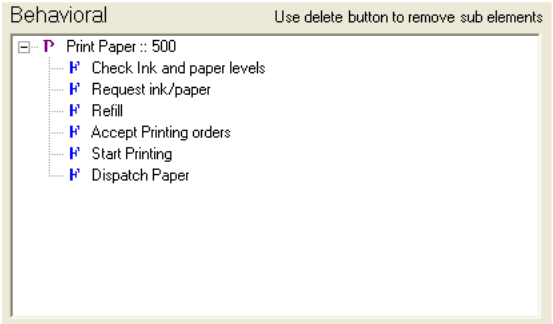

Figure 10.7: Enterprise Analyzer - Behaviorial view.

At any time, you can right click on a business process and select that you want to change the criticality of the process (Figure 10.8, 1). You will then be presented with a pop up dialogue that allows you to enter this value (Figure 10.8, 2).

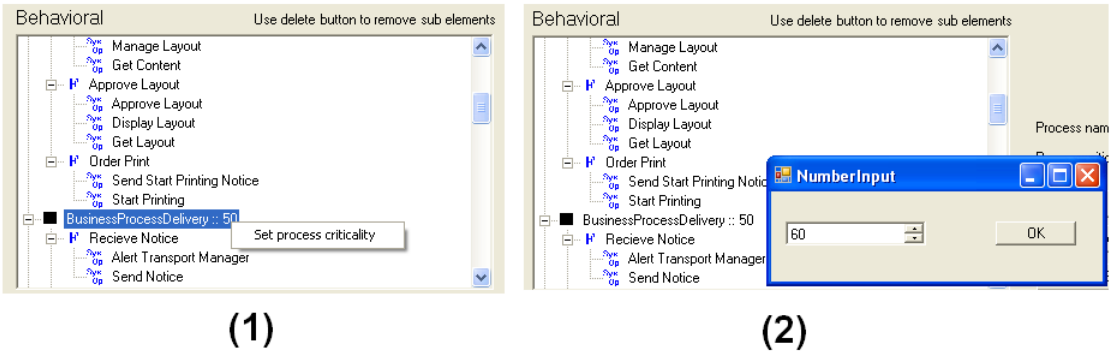

Figure 10.8: Enterprise Analyzer - Changing process criticality.

You can add system operations to the business functions by selecting one in the behavior view. When selected the business function is marked by a black square (See Figure 10.9). Press “Add System Operations” and you will be presented with a new file selecting dialogue. Select the XMI-file that contains the system operations to the given business function and press “OK”.

Again, you will be taken back to the main menu and the behavioral view is updated and contains all system operations you just added. See Figure 10.10.

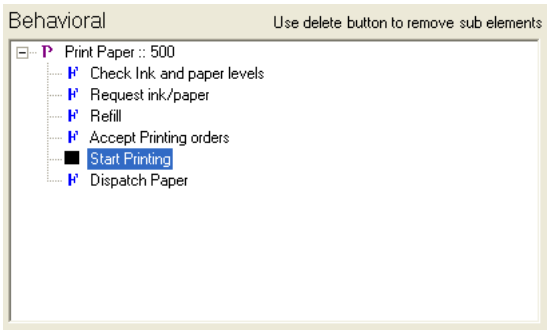

Figure 10.9: Enterprise Analyzer - Selecting a business function.

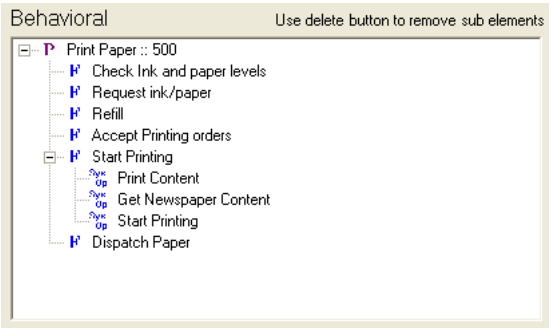

Figure 10.10: Enterprise Analyzer - Behavioral view II.

Likewise, one can select a system operation in the behavioral view (See Figure 10.11) and press the "Add Sub System Operations" button to add sub system operations to the given system operation. The result will look something like in Figure 10.12, showing the sub operations and their relation to the system operation.

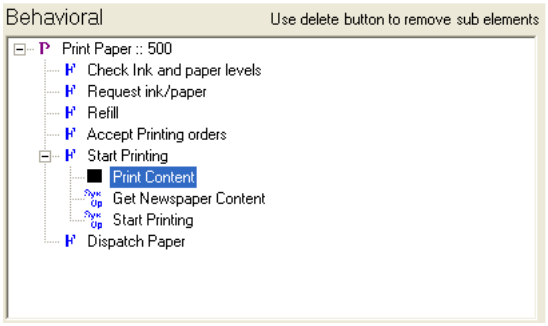

Figure 10.11: Enterprise Analyzer - Selecting a System Operation.

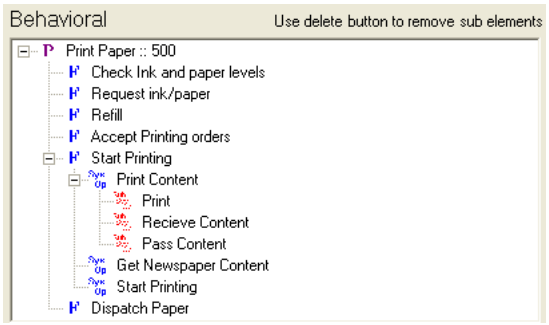

Figure 10.12: Enterprise Analyzer - Behavioral view.

The last steps (adding system and sub system operations) is repeated until the enterprise is fully built. At any time you can press "Analyze" in the main screen which will analyze the current enterprise representation. This will take you to the screen in Figure 10.13.

This screen shows the result of the analysis. The components, systems, and environments of the enterprise is shown in four lists, where each row is a part of the enterprise with the different columns showing the key figures of the analysis such as different types of availability and criticality. Pressing the "Back" button will take you back to the main screen.

By selecting an item in the behavior- and structure view we can delete all the subcomponents by pressing the "Delete" button on the keyboard. You will then be presented with a dialogue that asks you to confirm the deletion

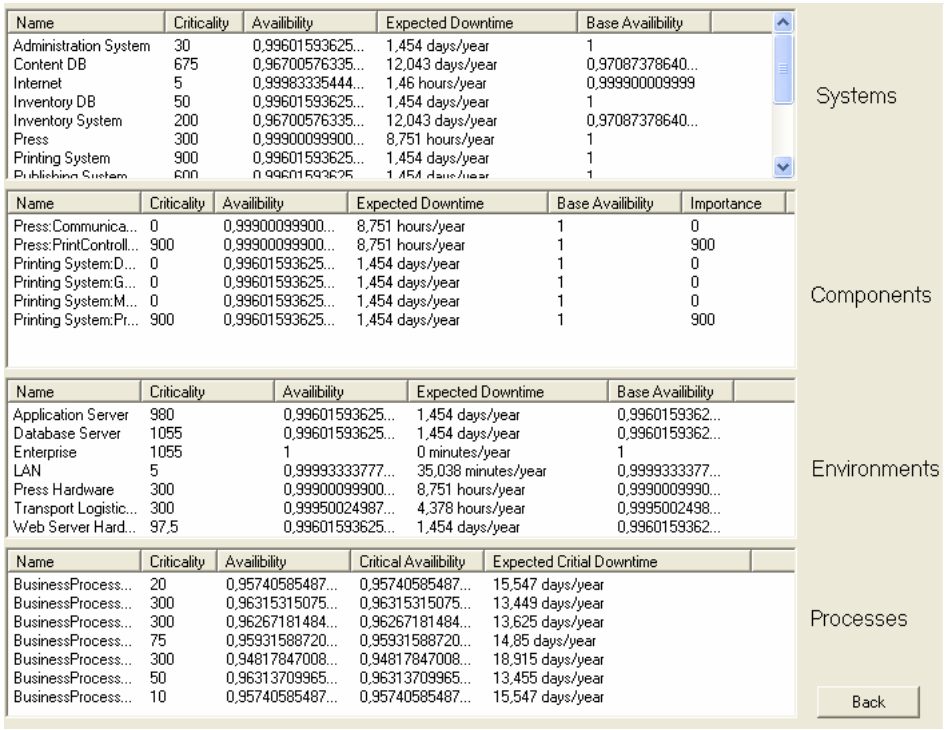

Figure 10.13: Enterprise Analyzer - Analysis result screen.

(Figure 10.14). This enables the user of the application to remove certain portions of the enterprise and reload it, which is necessary if a model has been changed.

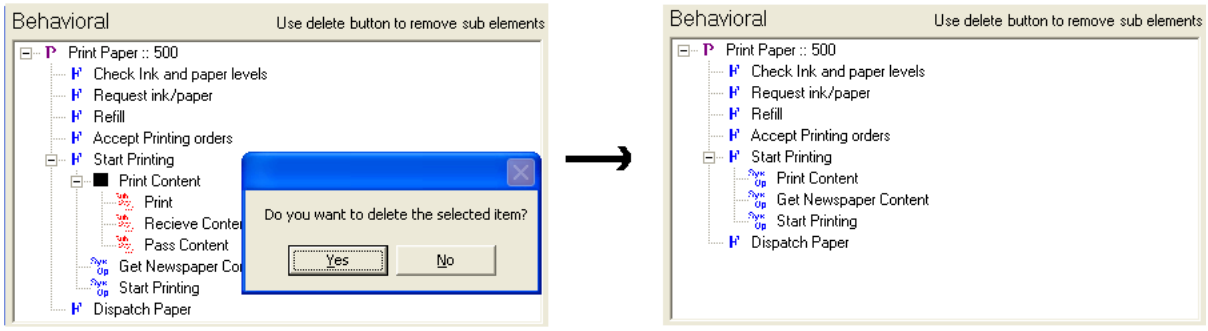

Figure 10.14: Enterprise Analyzer - Removing items.

You can also save the enterprise representation in a binary format by pressing "Save" on the main screen. This way you don't have to re-create the enterprise every time you restart the application. You will be presented by a file-saving dialogue (Figure 10.15).10

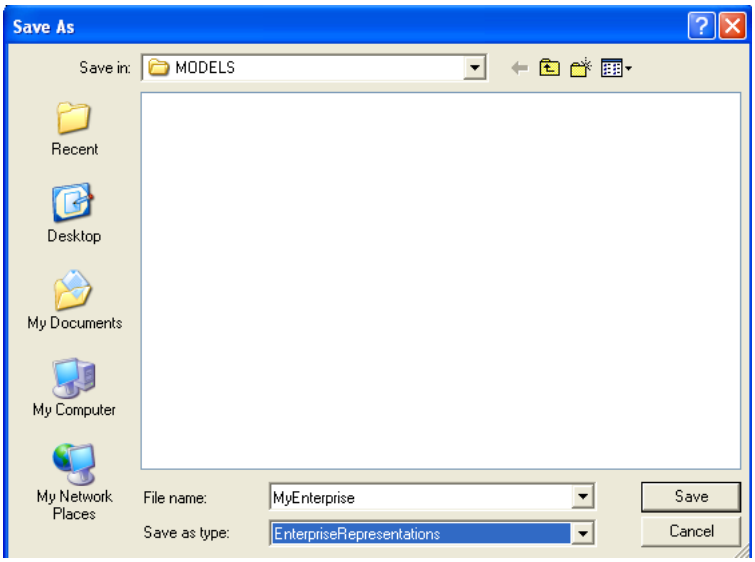

Figure 10.15: Enterprise Analyzer - Saving the enterprise representation.

Write the desired filename in the text box named "File name" and press "Save". You can later load this representation by pressing the "Load" button on the main screen. You will then be presented with a open-file dialogue

(Figure 10.16), which lets you select an enterprise representation with the ENT extension.

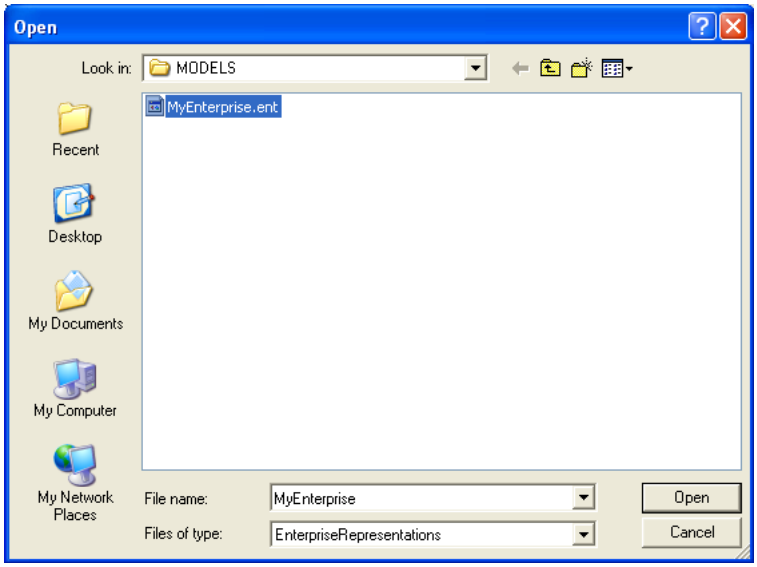

Figure 10.16: Enterprise Analyzer - Loading an enterprise representation.

10.2 The Model Analyzer

Figure 10.17 shows the main menu of the Model Analyzer, in the next sections I will describe each choice.

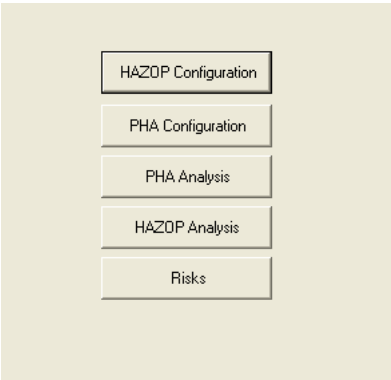

Figure 10.17: Model Analyzer - Main menu.

10.2.1 HAZOP configuration

The "HAZOP Configuration" takes you to the screen shown in Figure 10.18. From this screen you can create, edit, and delete HAZOP transformation profiles.

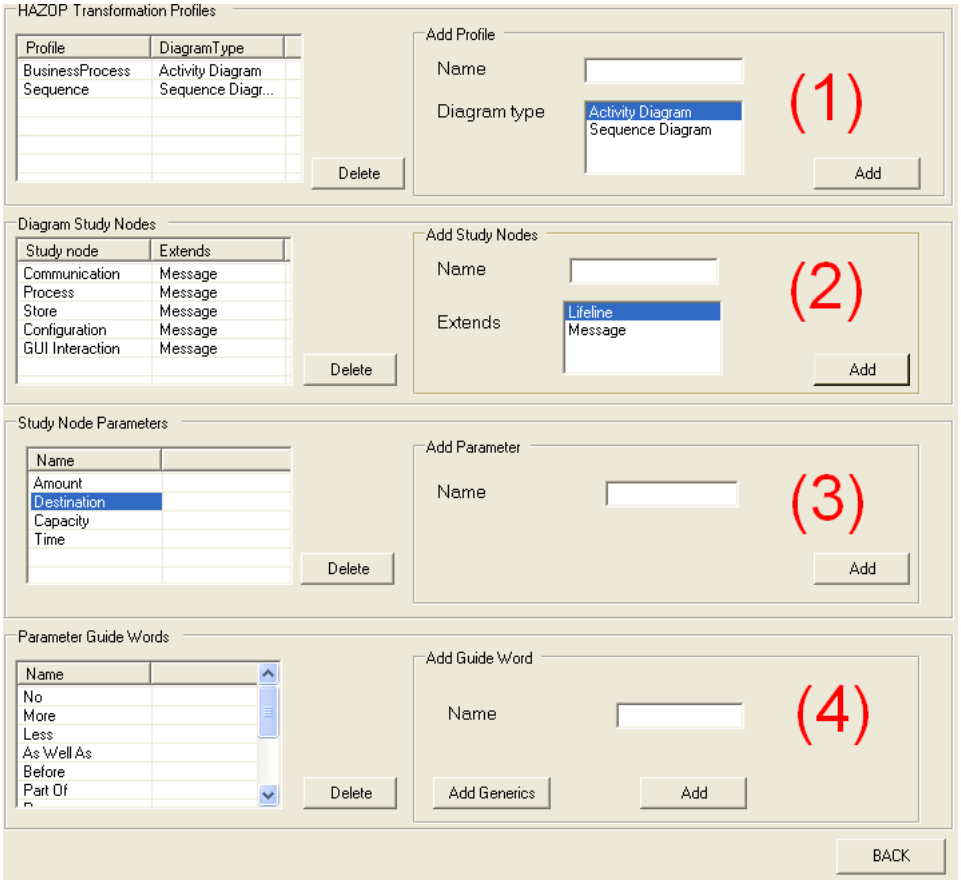

Figure 10.18: HAZOP Configuration screen.

The screen is divided into four sections which are marked on the screen. Section 1, right side, lets you add a profile. Select the type of diagram that the profile is going to transform and enter a name. Section 2, left side, lets you select or delete profiles.

When a profile is selected, you can add study nodes in Section 2. The principle is the same as in section 1. Choose which entity type the study node extends and enter a name. The list on the left side lets you select and delete study nodes.

When a study node is selected, you can add parameters in Section 2. Para-

meters only have a name. Further you can select a parameter and add guide words to it in section 4. You can either choose your own guide words, or click “Add Generics” and select from a list of common generic guide words. In the latter case, the dialogue in Figure 10.19 will pop up.

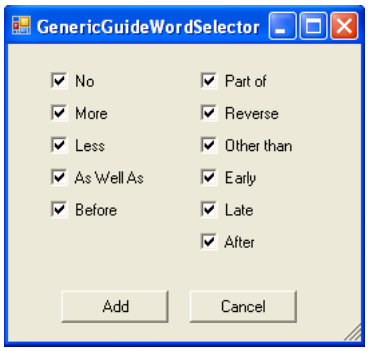

Figure 10.19: Adding generic guide words.

10.2.2 PHA configuration

The “PHA Configuration” takes you to the screen shown in Figure 10.20. From this screen you can create, edit, and delete PHA transformation profiles.

Section 1 and 2 is identical to the corresponding sections in the “PHA Configuration”-screen. Section 3 lets you add a stereotype dependent check list item, by giving a name and description. Section 4 lets you create a check list item with a tag constraint by supplying a tag name, operator, and a value.

10.2.3 HAZOP analysis

The “HAZOP Analysis” screen is shown in Figure 10.21. The list box marked 1, lets you select the type of diagram you want to analyze. When the selection is done, the list box marked 2 is filled with the transformation profiles that is associated with the given diagram type. By pressing the “Select File”, marked 3, you will be presented by the dialogue in Figure 10.22. Select the appropriate XMI-file, and then press “Analyze”, marked 4.

The result of the analysis is output in the list on the right side of the screen, see Figure 10.23.

You can select an output item and do one of two things. You can delete it

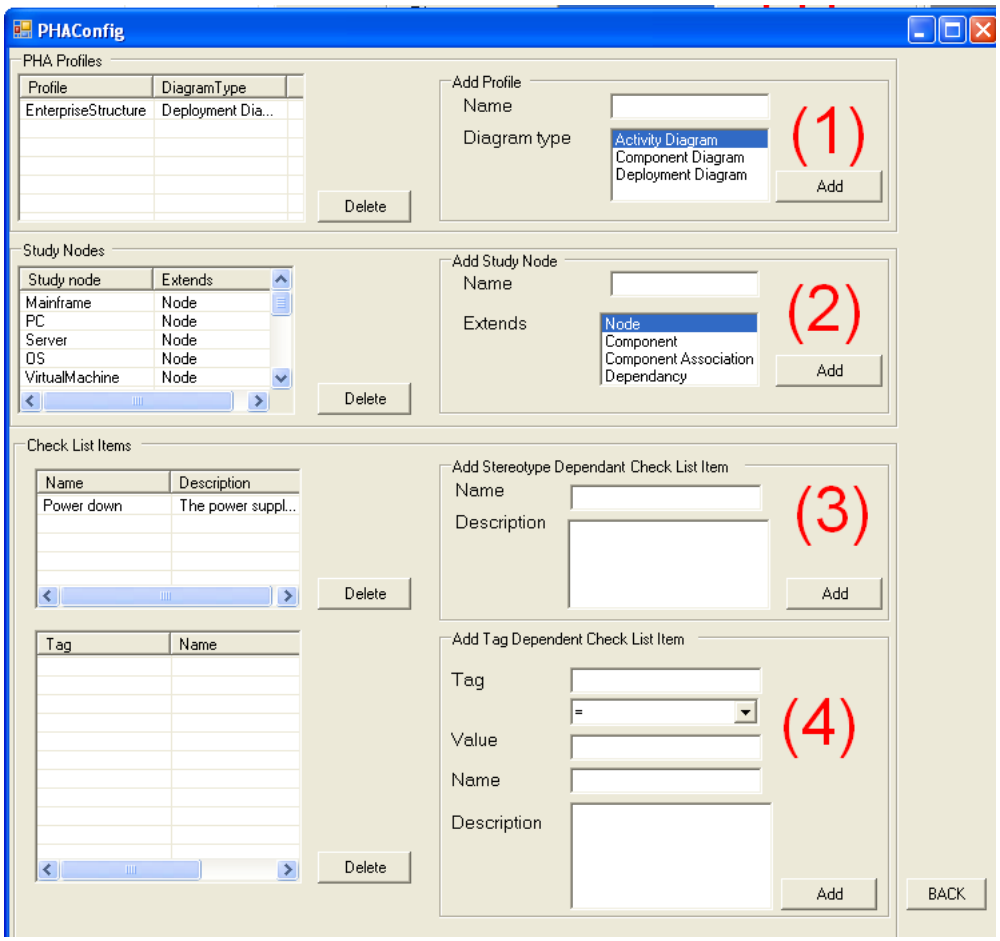

Figure 10.20: PHA Configuration screen.

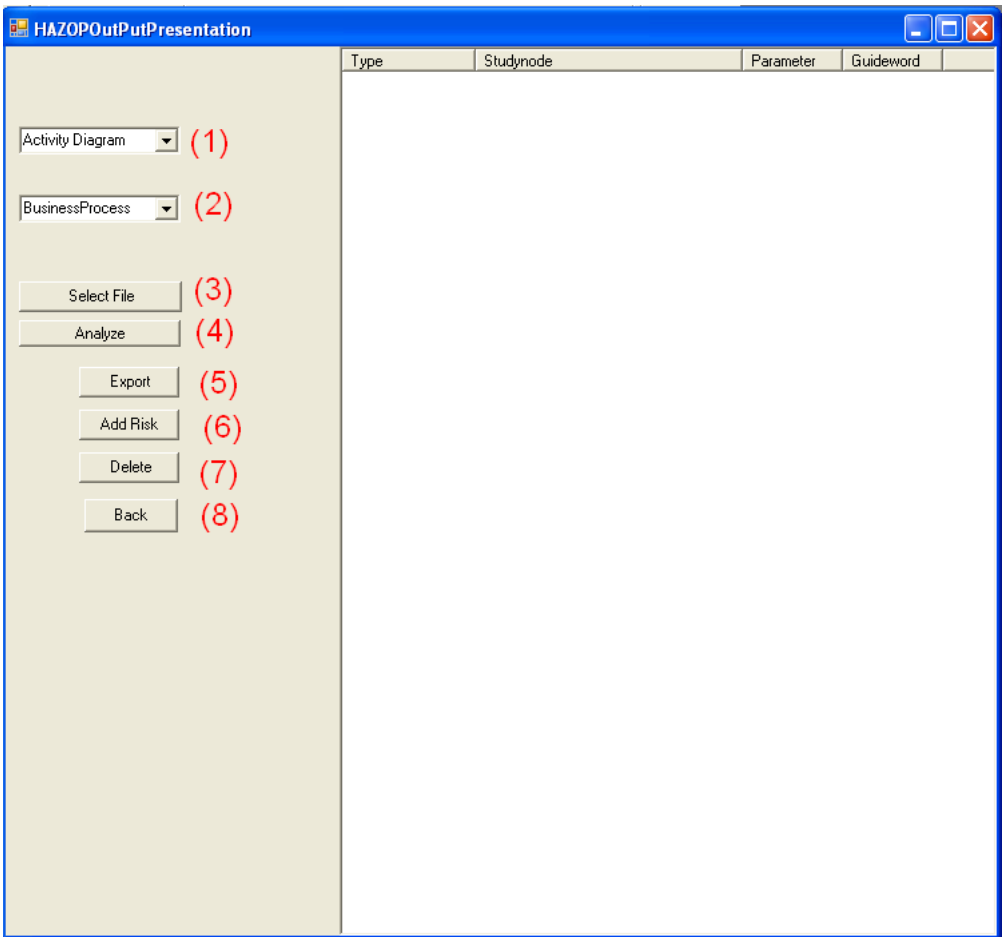

Figure 10.21: The HAZOP Analysis screen.

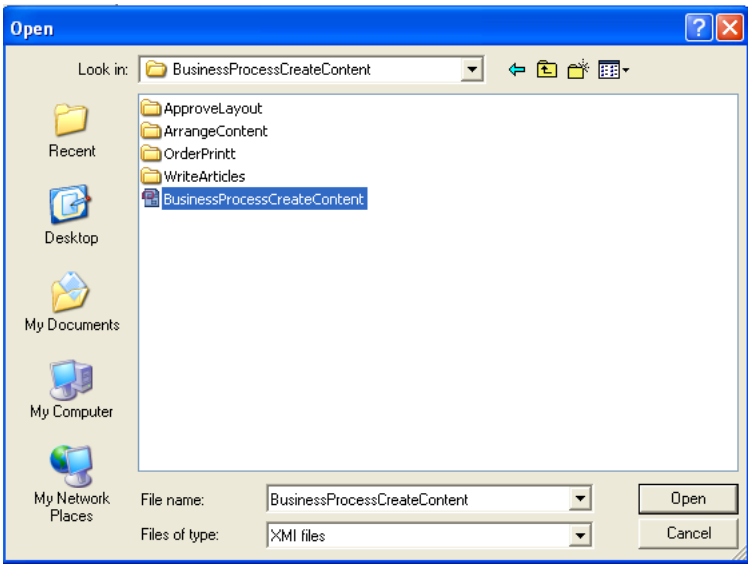

Figure 10.22: Selecting file.

by pressing the "Delete button", marked 7, or store the risk internally in the application by pressing the "Add Risk" button. When adding a risk, you will be presented by the dialogue shown in Figure 10.24.

By pressing the "Export" button, marked 5, you can export the list to an XML represented Office Word document.

10.2.4 PHA analysis

The "PHA Analysis" screen is identical to that of the "HAZOP Analysis" screen, except that it presents PHA output items instead of HAZOP output items.

10.2.5 Risks

The "Risks" shown in Figure 10.25 lets you view the risks stored internally in the tool, as well as adding new risks by pressing the "Add risk" button. You will then be presented with the dialogue in Figure 10.24.

Type	Studynode	Parameter	Guideword
Communication	Get Levels From:Inventory System To: Inventory DB	Validity	Less
Communication	Get Levels From:Inventory System To: Inventory DB	Validity	As Well As
Communication	Get Levels From:Inventory System To: Inventory DB	Validity	Before
Communication	Get Levels From:Inventory System To: Inventory DB	Validity	Part Of
Communication	Get Levels From:Inventory System To: Inventory DB	Validity	Reverse
Communication	Get Levels From:Inventory System To: Inventory DB	Validity	Other Than
Communication	Get Levels From:Inventory System To: Inventory DB	Validity	Early
Communication	Get Levels From:Inventory System To: Inventory DB	Validity	Late
Communication	Get Levels From:Inventory System To: Inventory DB	Validity	After
Communication	Get Levels From:Inventory System To: Inventory DB	Validity	No
Communication	Get Levels From:Inventory System To: Inventory DB	Validity	More
Communication	Get Levels From:Inventory System To: Inventory DB	Integrity	No
Communication	Get Levels From:Inventory System To: Inventory DB	Integrity	More
Communication	Get Levels From:Inventory System To: Inventory DB	Integrity	Less
Communication	Get Levels From:Inventory System To: Inventory DB	Integrity	As Well As
Communication	Get Levels From:Inventory System To: Inventory DB	Integrity	Before
Communication	Get Levels From:Inventory System To: Inventory DB	Integrity	Part Of
Communication	Get Levels From:Inventory System To: Inventory DB	Integrity	Reverse
Communication	Get Levels From:Inventory System To: Inventory DB	Integrity	Other Than
Communication	Get Levels From:Inventory System To: Inventory DB	Integrity	Early
Communication	Get Levels From:Inventory System To: Inventory DB	Integrity	Late
Communication	Get Levels From:Inventory System To: Inventory DB	Integrity	After
Communication	Get Levels From:Inventory System To: Inventory DB	Load	No
Communication	Get Levels From:Inventory System To: Inventory DB	Load	More
Communication	Get Levels From:Inventory System To: Inventory DB	Load	Less
Communication	Get Levels From:Inventory System To: Inventory DB	Load	As Well As
Communication	Get Levels From:Inventory System To: Inventory DB	Load	Before
Communication	Get Levels From:Inventory System To: Inventory DB	Load	Part Of
Communication	Get Levels From:Inventory System To: Inventory DB	Load	Reverse
Communication	Get Levels From:Inventory System To: Inventory DB	Load	Other Than
Communication	Get Levels From:Inventory System To: Inventory DB	Load	Early
Communication	Get Levels From:Inventory System To: Inventory DB	Load	Late
Communication	Get Levels From:Inventory System To: Inventory DB	Load	After
Communication	Get Levels From:Inventory System To: Inventory DB	Destination	No
Communication	Get Levels From:Inventory System To: Inventory DB	Destination	More
Communication	Get Levels From:Inventory System To: Inventory DB	Destination	Less
Communication	Get Levels From:Inventory System To: Inventory DB	Destination	As Well As
Communication	Get Levels From:Inventory System To: Inventory DB	Destination	Before
Communication	Get Levels From:Inventory System To: Inventory DB	Destination	Part Of
Communication	Get Levels From:Inventory System To: Inventory DB	Destination	Reverse
Communication	Get Levels From:Inventory System To: Inventory DB	Destination	Other Than
Communication	Get Levels From:Inventory System To: Inventory DB	Destination	Early
Communication	Get Levels From:Inventory System To: Inventory DB	Destination	Late
Communication	Get Levels From:Inventory System To: Inventory DB	Destination	After
Communication	Get Levels From:Inventory System To: Inventory DB	Time	No
Communication	Get Levels From:Inventory System To: Inventory DB	Time	More

Figure 10.23: HAZOP output item.

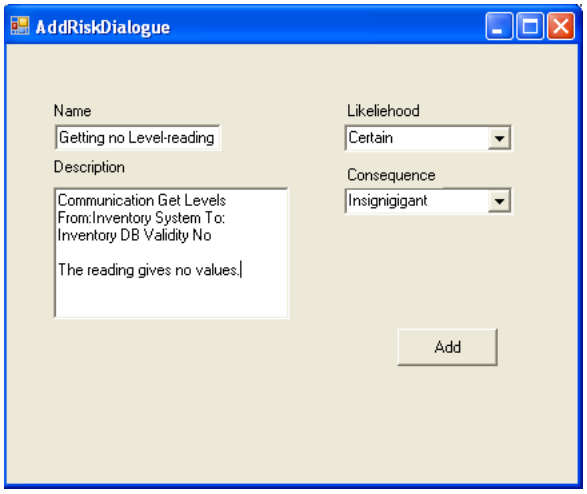

Figure 10.24: Add risk dialogue.

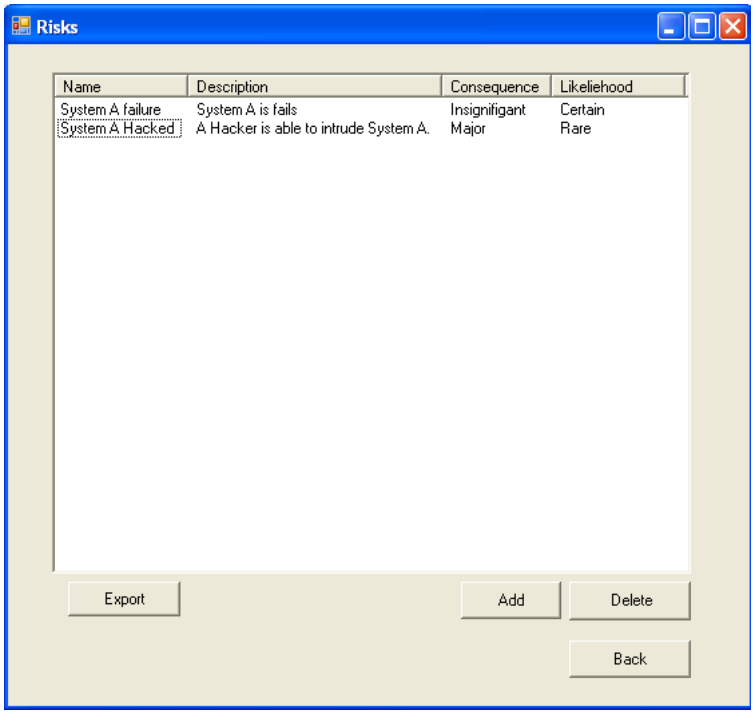

Figure 10.25: Risk list.

Chapter 11

Discussion

The work of this thesis can be divided into the following parts or products.

- Definition and development of an Enterprise Representation.
- Development of the Enterprise Analyzer tool.
- Development of the Model Analyzer tool.
- Definition of a set of Enterprise UML Profiles.
- Definition of a set of Transformation Profiles.
- Definition of a work process that uses these tools.
- Construction of a case enterprise and a test of the tools.
- The user manuals for the tools.

The Enterprise Representation is a model that encompasses both the behaviorial and structural parts of the enterprise at different levels. The model allows traceability between all the sub parts. I selected a set of UML models for creating this representation and developed a data structure that represented the Enterprise Representation and had functionality to access and build it.

I developed the Enterprise Analyzer, which has two functions. First, it allows the user to build an Enterprise Representation by loading digitized UML models. Second, it allows the user to take an Enterprise Representation and analyze the criticality and availability of the solution. This is done by tracing the availability from the structural parts to the behavioral and trace the criticality from the behavioral parts to the structural parts.

To better the criticality of the behavioral parts and the availability of the structural parts, I developed the Model Analyzer tool that lets the user analyze a single model based on a Transformation Profile.

A set of UML profiles to model the enterprise was defined. These are extensions to the UML language that allow a more precise description of the enterprise domain.

The Transformation Profiles describes how the Model Analyzer turns a model in a Enterprise UML Profile into a set of risks.

I have proposed a work process that describes how the tools should be applied when designing and analyzing an enterprise solution.

The tools and the process have been tested by modeling a case enterprise and taking it through analysis work process.

User manuals for the two tools are also contained within this document.

In the next sections I will discuss the work.

11.1 The Enterprise Representation

I believe that the Enterprise Representation captures the important parts of the enterprise without being too complex. The modelers need only refine the parts of the enterprise that are critical, thus limiting their work.

The purpose of modeling is mainly to communicate design to customers and developers. Modeling all critical interactions, means a significant increase in work load and cost. The question is then, is the extra modeling worth it in terms of what the criticality and availability analysis returns?

The strengths of this model driven analysis approach is the fact that no single person need to have complete overview of the enterprise. Therefore, the larger and more interconnected the enterprise is, the more should be gained by using this approach.

If conventional analysis is to succeed, there must also be good communication between the different modelers if they are to see the whole picture. This means that if the developing organization is dispersed in time and/or space, the model driven approach gives an advantage. Also, as the extra modeling translates into higher development costs, the approach should be applied to enterprise projects where the cost of failing is high.

The Model Driven Development (MDD) supporters envisions that nearly all software development will be done by modeling in the future, [23]. IBM pro-

mote the use of UML Profiles, while Microsoft promote the use of Domain Specific Languages (DSLs). In both cases the idea is that models are transformed into code. If the future is MDD, the developers have to model the whole solution, and therefore the modeling amount is no longer an argument against model driven analysis.

In the future there will probably also exist a set of industry standard models to develop enterprise systems that will replace my models, but these new models could easily be parsed into my Enterprise Representation. Many of these standard models are already proposed, [21], [22].

All the models are based on the v1.4 specification of UML, even though most of the current modeling tools support the v2.0 specification of UML. The reason for this is that the XMI specification for storing v2.0 UML diagrams digitally, the XMI v2.1 specification, is not implemented in most modeling tools. The v1.2 XMI specification (for storing v1.4 UML diagrams) is widely supported. I used the tool "Visual Paradigm for UML 5.2".

11.2 The Enterprise Analyzer

The Enterprise Representation proved to be an easy representation to analyze. The redundant structure and references made it easy to navigate and extract relevant relations and values. Calculating the key parameters was a very straightforward programming job.

The approach to building the enterprise representation, which is done by loading one model at a time, may be time consuming for larger projects. Information about which model the given model refines can be stored in meta data in the diagrams or possibly be defined in the filename. This will allow automated building of the enterprise.

A weakness in the implementation is that every business process is treated as independent of the others. If one business function A depends on B, the criticality of the latter cannot be less than the criticality of the one that depends on it. This can be worked around by modeling the two business processes as one. There is the possibility of letting users define the dependencies between processes in the tool, which allows the modeling of two interdependent business processes separately.

11.3 The Model Analyzers

The HAZOP analysis proved to be a good explorative method. It is easy to configure to the parameters and guide words.

The PHA analysis makes it easy for the developing organization to store dangers they have encountered as check list items in a PHA Transformation Profile. This way the tool serves as an simple, yet effective, way of collecting risk knowledge in the organization.

11.4 The Enterprise UML profiles

The Model Analyzer allows the developers to use their own UML profiles. For example, a company specialized in developing enterprise systems in the health care domain, could create their own health care specific stereotypes. This feature gives the tool large flexibility.

11.5 The Transformation profiles

The Transformation Profiles were created to test the Model Analyzer. The HAZOP Profile performed well, but the PHA did not show its full potential due to the fact that the profile was relatively small. We expect that PHA will become more useful as the amount of knowledge stored in the profiles grows.

The users of the Model Analyzer can create their own profiles. For example, the above mentioned health care specialized company, could easily create their own transformation based on their specialized UML Profile with logic that produced health care specific risks and considerations. This is a powerful feature of the tool.

11.6 The process

The process defined a simple and effective approach to using the tools during design and analysis of the enterprise solution.

Chapter 12

Conclusions and Future Work

12.1 Conclusions

The aim of this thesis has been to develop a set of tools and a work process that would allow model driven availability and criticality analysis of large scale enterprise solutions. I saw two possible approaches to model driven analysis, which resulted in two tools used in my process.

The Enterprise Analyzer, a tool that takes basis in a set of models with different views of the enterprise and merges them into a unified model (data structure) of the enterprise called the Enterprise Representation. Then it does a criticality and availability analysis of the total model.

I also developed the Model Analyzer which takes basis in a single digitized model and does an assisted analysis using methods from the field of System Safety.

The tools were tested on a hypothetical case. The Model Analyzer proved to be easy to use, customizable, and efficient in analyzing a part of the enterprise. Especially the PHA (Preliminary Hazard Analysis) analyzer within this tool made it easy to store the risk knowledge and transfer it into future projects.

The Enterprise Analyzer proved to overcome the obstacle of analyzing large enterprise solutions modeled in different views. The question that arose was whether the task of modeling the whole solution proved to be too cumbersome, or that the result of the analysis was worth the extra modeling.

I claim that it is useful when designing large and highly interconnected

enterprises. Also developing projects that are dispersed in time and space will benefit from this automatic model driven approach to analysis. I also claim that this kind of approach to analysis will become more common and cost effective in the future when Model Driven Development becomes more mature.

12.2 Future work

I have identified the following possible directions for future work.

Incorporate data and user modeling The current Enterprise Analyzer encompasses the structural and behavioral parts of the enterprise. It is possible to also incorporate data- and user modeling in the enterprise representation. This way we can also calculate the availability and criticality of data. This is possible with the current version by modeling data and user as structural parts of the enterprise, but separating the three allows greater flexibility in adding new analysis features.

More quality attributes There is the possibility of incorporating more quality attributes into the analysis process such as QoS (quality of service). If one introduces users as a modeling construct as proposed in the previous section, this could easily be incorporated. There is a connection between an enterprise's quality of service and the profitability, which in turn affects criticality.

Risk tracing As of today, the Model Analyzer and the Enterprise Analyzer are two separate tools. By linking risks uncovered by the model analysis to the different parts of the enterprise representation, we could have a dynamic indicator of each of the risks criticality. That is as the criticality of different parts of the enterprise change, so does that the criminalities of the risks associated with the given part. The Model Analyzer could be integrated into the Enterprise Analyzer and do a direct analysis of the Enterprise Representation.

More analysis method support The tools could incorporate more methods of analysis from the field of System Safety. Methods such as Fault Tree Analysis (FTA) and Fault Mode Effects Analysis (FMEA) could be included in this tool.

Industry Standard models As industry standard model for modeling enterprise systems for Model Driven Development purposes becomes available, the XMI-readers could be altered to parse these structures into an Enterprise Representation.

Testing the tools on an industrial case I tested the tools on a constructed case. It would be natural to follow up by a test on a large scale industry case enterprise.

Appendix A

Code Overview

This appendix gives a short description of the different files in the three projects and contains the class diagrams of the tools.

A.1 EnterpriseRepresentation

The EnterpriseRepresentation project folder contains all the classes needed to build a representation of the enterprise. The project is compiled into a DLL-file and used by Enterprise Analysis tool. Table A.1 and A.2 contain information about each of the code files in the project. Information such as filename, a description of the file, an ID that is used for refereing to the files. Also, table has a reference to which classes the given class extends (Column E) and the interfaces it implements (Column I).

A.2 EnterpriseAnalysis

The EnterpriseAnalysis project folder contains the classes and files that is needed to build the Enterprise Analyzer tool. The project is compiled into an executable. Table A.3 contain information about each of the code files in the project.

A.3 RiskAnalysis

The RiskAnalysis project folder contains the classes and files that is needed to build the Model Analyzer tool. The project is compiled into an exe-

ID	Filename	Description	E	I	Other
1	BusinessFunction.cs	A class used to represent a business function in the representation of a business scenario.	5	15	Serializable
2	BusinessFunctionDecision.cs	A class used to represent a decision node in the representation of a business scenario	5	14	Serializable
3	BusinessFunctionFork.cs	A class used to represent a forking in the representation of a business scenario	5	14	Serializable
4	BusinessFunctionMerge.cs	A class used to represent a forking in the representation of a business scenario.	5	-	Serializable
5	BusinessFunctionObject.cs	A base class that implements common functionality that all the classes that make up the business scenario representation shares	-	-	Serializable
6	BusinessProcess.cs	A class that keeps information about a business process and acts as a dummy-node in the start of the business scenario representation	1	-	Serializable
7	BusinessProcessReader.cs	A class that parses Activity Diagrams containing business process data and builds a BusinessProcess (see 6) object out of it	-	-	Serializable
8	Component.cs	A class that represents a software component in enterprise in the enterprise representation	11	-	Serializable
9	ComponentReader.cs	A class that parses a Component Diagram containing a structural representation of a system and populates a System (see 18) with Components (see 8)	-	-	Serializable
10	DeploymentReader.cs	A class that parses a Deployment Diagram containing a structural representation of the enterprise (environments and systems) and builds it using Environments (See 13) and System (see 18)	-	-	Serializable
11	EnterpriseEntity.cs	A base class that implements common functionality that all the classes that make up a part of the enterprise shares	-	-	Serializable
12	EnterpriseRepresentation.cs	A class that encompasses all the other classes and acts like the access class to the enterprise representation	-	-	Serializable
13	Environment.cs	A class that represents an environment in the enterprise representation	11	-	Serializable
14	IFunctionMultipleNext Objects.cs	An interface used to access the succeeding BusinessFunctionObject-extended classes from a class extending BusinessFunctionObject. This interface is used when the class has multiple next-objects	-	-	-
15	IFunctionNextAccessor.cs	An interface used to access the succeeding BusinessFunctionObject-extended class from a class extending BusinessFunctionObject. This interface is used when the class has a single next-object	-	-	-

Table A.1: Files in the EnterpriseRepresentation project - I.

ID	Filename	Description	E	I	Other
16	SequenceReader.cs	A class that is used to parse Sequence Diagrams into linked lists of System Operations (see 19) and Sub System Operations (see 17)	-	-	Serializble
17	SubSystemOperation.cs	A class that represents a Sub System Operation. A System Operation (See 19) is composed of a linked list of SubSystemOperation-objects	11	-	Serializble
18	System.cs	A class that represents a System in the Enterprise representation	11	-	Serializable
19	SystemOperation.cs	A class that represents a System Operation, Business Functions (see 1) are composed of a linked list of SystemOperation-objects.	11	-	Serializable
20	TagValue.cs	A class that represents a tag value, which is a name/value pair that stores information about a part of enterprise. This information is extracted from the UML diagram	-	-	Serializable

Table A.2: Files in the EnterpriseRepresentation project - II.

cutable. Table A.4, A.5, and A.6 contain information about each of the code files in the project.

A.4 Class diagrams

ID	Filename	Description
1	AnalysisPresenter.cs	This class is a .NET User Control that presents the results from the analysis of the enterprise.
2	AnalysisResult.cs	This class holds the analysis result from the analysis of the enterprise done by the EnterpriseAnalyzer (See 5)
3	ComponentAnalysisResult.cs	A class that holds the result of the component analysis done by the ComponentAnalyzer (See 4)
4	ComponentAnalyzer.cs	A class that analyzes the components in the enterprise and produces a ComponentAnalysisResult (See 3)
5	EnterpriseAnalyzer.cs	A class that analyzes the whole enterprise by using the analyzers (See 4,8,11 and 13). It returns a AnalysisResult object (See 2)
6	EnterpriseRepresentationCreator.cs	The class is a .NET User Control that lets the user create an enterprise representation by loading diagrams into the application
7	EnvironmentAnalysisResult.cs	A class that holds the result of the environment analysis done by the EnvironmentAnalyzer (See 8)
8	EnvironmentAnalyzer.cs	A class that analyzes the environments in the enterprise and creates an EnvironmentAnalysisResult (See 7)
9	MainMenu.cs	The MainMenu is a .NET Windows Form file that is the main menu of the application. The user controls AnalysisPresenter (See 1) and EnterpriseRepresentationCreator (See 6) is docked into this form.
10	ProcessAnalysisResult.cs	A class that holds the result of the process analysis done by the ProcessAnalyzer (See 11)
11	ProcessAnalyzer.cs	A class that analyzes the business processes in the enterprise and creates a ProcessAnalysisResult (See 10)
12	SystemAnalysisResult.cs	A class that holds the result of the system analysis done by the SystemAnalyzer (See 13)
13	SystemAnalyzer.cs	A class that analyzes the systems in the enterprise and creates a SystemAnalysisResult (See 12)

Table A.3: Files in the EnterpriseAnalysis project - I.

ID	Filename	Description	E
1	ActivityDiagram.cs	A class that represents a UML Activity Diagram.	10
2	ActivityState.cs	A class that represents the state in a UML Activity Diagram	11
3	ActivitiyTransition.cs	A class that represents the transition in a UML Activity Diagram	11
4	AddRiskDialogue.cs	A dialogue form that lets the user of the Model Analyzer tool to add a Risk to the Risk-list	-
5	Association.cs	A class that represents the association between nodes/components in UML Component/Deployment Diagrams	11
6	Component.cs	A class that represents a component in a UML Component/Deployment Diagrams	11
7	ComponentDependency.cs	A class that represents the depends-relation between components in a UML Component Diagram.	11
8	ComponentDiagram.cs	A class that represents a UML Component Diagram	10
9	DeploymentDiagram.cs	A class that represents a UML Deployment Diagram	10
10	Diagram	A base class that contains common functionality that all the diagram represents use. Mainly to extract stereotypes and tag-values	-
11	DiagramElement	A base class that contains common functionality that all the diagram elements in the diagram representations uses. Mainly to access stereotypes and tag-values	-
12	GenericGuideWordSelector.cs	A .NET User Control that lets the user select a set of generic guide words and add them to a given parameter in the HAZOP transformation profile	-
13	HazopConfigControl.cs	A .NET User Control that lets the user configure a HAZOP transformation profile	-

Table A.4: Files in the RiskAnalysis project - I.

ID	Filename	Description	E
14	HazopOutputItem.cs	A class that represents a output line of the HAZOP analysis.	-
15	HAZOPOutputPresenter.cs	A .NET User Control that presents the result of a HAZOP analysis.	-
16	HazopTransformationProfileManager.cs	A class that handles the creation, deletion and editing of HAZOP transformation profiles.	-
17	phaconfig.mbd	An Access-database file where the PHA transformation profiles are stored	-
18	HazopTransformer.cs	A class that does a HAZOP analysis (transformation) on a UML diagram.	-
19	MainForm.cs	The MainForm is a .NET Windows Form that is the main form of the application, in which the User Controls are docked.	-
20	Menu.cs	A .NET User Control that contain the main menu of the Model Analyzer tool	-
21	Node.cs	A class that represents the node in a UML Deployment Diagram	-
22	PHAConfigControl.cs	A .NET User Control that lets the user configure a PHA transformation profile	-
23	PHAOutputItem.cs	A class that represents a output line of the PHA analysis.	-
24	PHAOutputPresenter.cs	A .NET User Control that presents the result of a PHA analysis	-
25	PHATransformationProfileManager.cs	A class that handles the creation, deletion and editing of PHA transformation profiles	
26	PHATransformer.cs	A class that does a PHA analysis (transformation) on a UML diagram	-
27	ReportMaker.cs	A class that creates reports	-
28	Risk.cs	A class that represents a single risk. This class is serializble, so that the risks can be stored in a binary format in between the executions of the tool	-
29	NumerInput.cs	A .NET Windows Form that is shown to get the user to supply a number.	-

Table A.5: Files in the RiskAnalysis project - II.

Figure A.1: EnterpriseRepresentation Class Diagram.

Figure A.2: EnterpriseAnalysis Class Diagram.

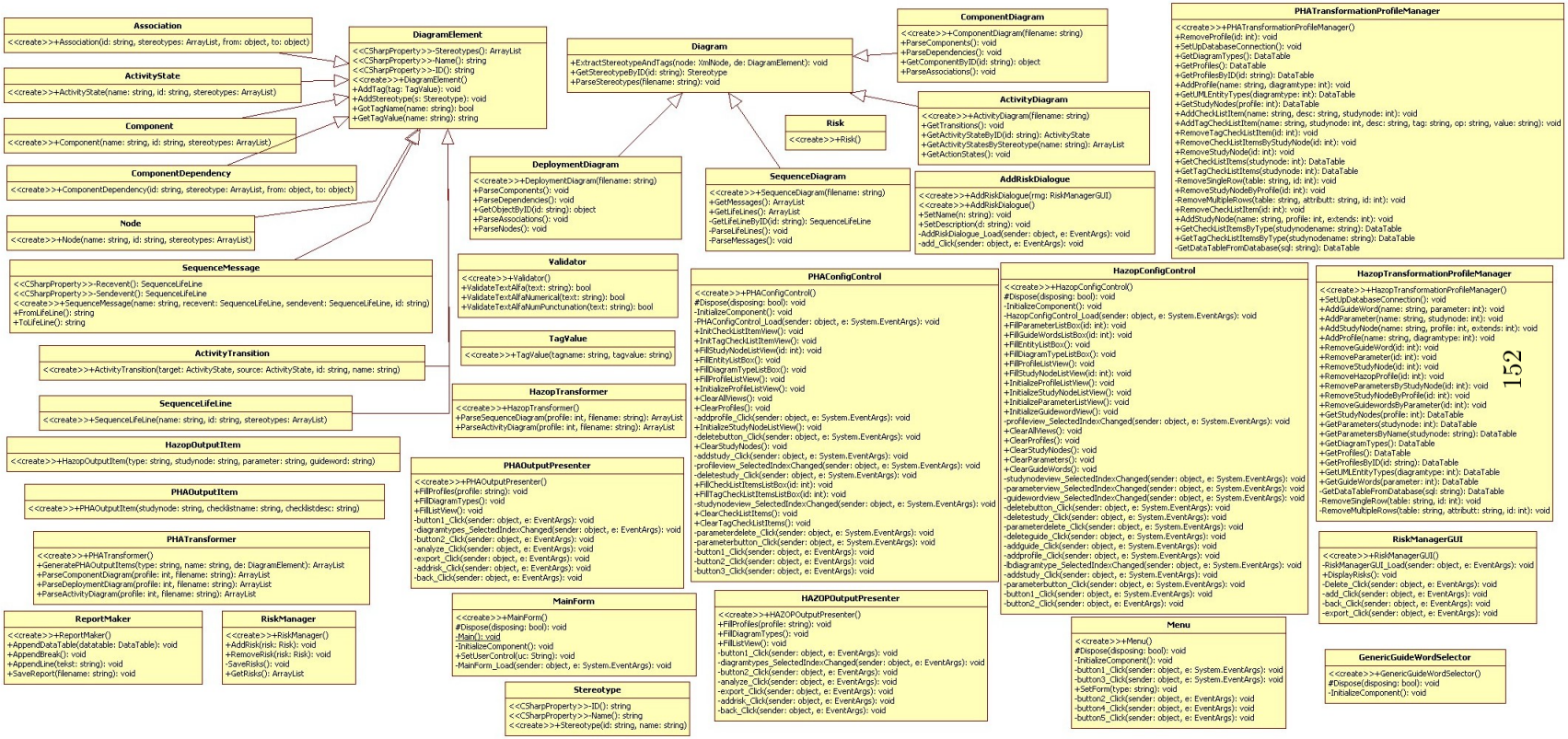

Figure A.3: RiskAnalysis Class Diagram.

ID	Filename	Description	E
29	RiskManager.cs	A class that the storage and retrieval of risks	-
30	RiskManagerGUI.cs	A .NET User Control that lets the user manage risks	-
31	SequenceDiagram.cs	A class that represents a UML Sequence Diagram	10
32	SequenceLifeLine.cs	A class that represents a lifeline in a UML Sequence Diagram	11
33	SequenceMessage.cs	A class that represents a message in a UML Sequence Diagram	11
34	Stereotype.cs	A class that holds a UML stereotype in a Diagram	-
35	TagValue.cs	A class that holds a UML tag value in a Diagram	-
36	Validator.cs	A class that is used to validate input that is going to be stored in a database	-
37	Hazopconfig.mbd	An Access-database file where the HAZOP transformation profiles are stored	-
38	Risks.dat	A binary files where the risks are stored in between executions of the tool.	-

Table A.6: Files in the RiskAnalysis project - III.

Appendix B

An Introduction to Risk

B.1 Risk

B.1.1 What is risk?

A *risk* is the possibility of something (usually negative) that may happen in the future, [6].

B.1.2 Risk terminology

A *hazard* or a *threat* is a set of conditions that may lead to a undesirable event. *Risk* can also be viewed as the probability of a threat or hazard to occur (its likelihood) multiplied with the consequence, also known as the criticality.

Risk management is the process of identifying, prioritizing, recording, treating, and monitoring risks, [7].

Risk managers must consider two dimensions of risk, its *consequence* and its *possibility*, and find a balance between these two. Risks can be divided into two groups, accepted and unaccepted risks. Risk managers will have to transform unacceptable into acceptable risks (Figure B.1). This is called risk migration or mitigation, and can be done in two dimensions. You can try to limit the consequences of the risk, and you can try to lower the possibility of the risk to occur.

A *fault* is a undesirable state of the system, a *failure* is the loss of functionality in a system. A fault may not result in a failure, but failure is always caused by faults. A *secondary failure* is a failure that is caused by

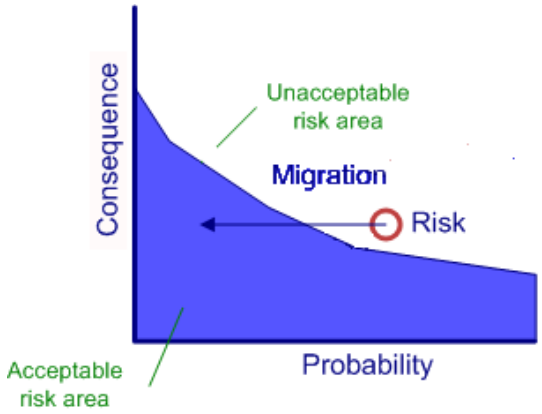

Figure B.1: Risk migration.

stress exceeding the levels in which the system-component is designed for. A *primary failure* is a failure the occurs even tough the component is not pushing its stress levels. That is a component which is incorrectly designed, selected, or installed.

B.1.3 Risk Management

Risk management can be divided into a set of activities, [10]:

Identify. The identification activity is to discover the risks that face the system. This is done by gathering data about about the system.

Analyze. The analysis of risks is taking the gathered risk data from the system and turning it into decision-making information.

Plan. The planning activity deals with how we handle the risk. That is to make decisions and actions out of risk information. There are four ways to deal with a risk:

1. Change the design.
2. Create a plan to handle the risk.
3. Accept the risk.
4. Transfer the risk.

The first possible action is to change the design of the system, that is to build a barrier into the system or rework the architecture to reduce the risk. The second action is to accept the risk and prepare

for it by creating plans. This includes plans that minimize the chance of occurrence of the risk (maintenance plans) and plans that tell the organization what to do when the risk is realized (emergency plans). This is an option when the cost of taking a well prepared risk is cheaper than to redesign the system.

The third option is to accept the risk and do no planning for it. In these cases the criticality of the risk is so low that neither planning or redesigning is worth the effort.

The last option is to transfer it, that is to pass the responsibility the another part. One example of risk transferal is insurance.

Track. Risk tracking is the activity of recording and monitoring the risks. All risks uncovered should be organized and available for the people who needs that information, and there should be some "watchdog mechanism" that alerts the organization when a risk has triggered.

Control. Risk control is the activity of implementing and controlling the plans and policies.

Communicate. The activity serves the need to communicate the risk to the involved parties, that being workers, customers etc.

It is important to remember that the job of the risk manager is not to minimize the occurrence of every risk, which is something that might seem rational for someone new to the field. Rather, the goal is to minimize the criticality. For most companies in the world, criticality is measured in currency. We do not need to construct the most sophisticated fire-prevention system in the world (likelihood reduction) when we can simply sign an insurance.

However, there is more to the job of the risk manager than considering dollar figures, the one factor that really complicates the work is the value of human life. By setting the value unlimited every possible safety feature concerning human safety considered would be implemented, this forces companies to put a value on human life.

B.1.4 Risk Management Process

The risk management process (Figure B.2) describes the interrelation of activities which is performed in the risk management discipline, [11]. It is the systematic application of these activities. It is this process we want to match as closely as possible with the RUP-process.

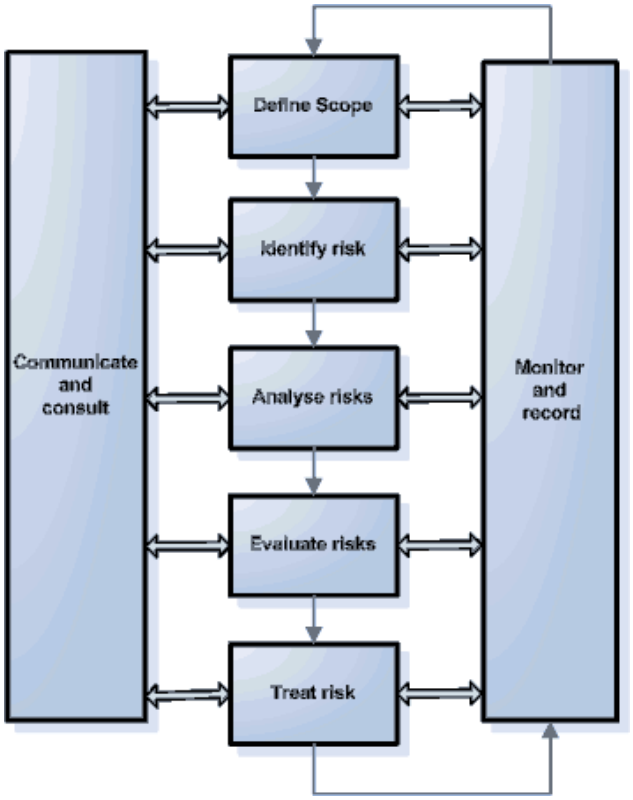

Figure B.2: The risk management process.

The first thing done before the analysis is the definition of the scope. The scope defines how deep into the system, or out of the system, we should go when finding risks.

Risk identification means verbalizing a possible threat: *"There could be a fire in the reactor"*. Analyzing the risks means finding the consequences of it: *"That would halt our production for days."*

Consulting means that we need to consult the people with specific competence to help us find the criticality of the risk: *"Call the chief engineer and ask him about the possibility of fire"*. It is not required, nor expected, that the risk analysts have complete knowledge about every aspect of the system.

Risk evaluation means giving the risk qualitative or quantitative data to evaluate the severity: *"A fire would happen every year on average, and there is a 75 percent chance that a fire would stop production for a week or more"*.

Treating the risks means taking appropriate actions to handle it: *"We need to install fire extinguishers near the reactors"*.

[15] states that the following is the safety order of precedence for risk treatment:

1. Design for minimum risk.
2. Incorporate safety devices.
3. Provide warning devices.
4. Develop training and procedures.

The first thing we must try to do is to try to design the system to minimize or eliminate the risk.

Those that cannot be minimized satisfactory (of financial or practical reasons) enough though design, we can incorporate safety devices (or safety features as they will be called from now) into the system.

If the safety features themselves cannot handle a risk, we must create warning devices which alerts a signal to the organization that something undesirable may happen, or already has happened.

The last option is to create safety procedures that can be engaged by the organization, and train the organization to follow them. In the case of the enterprise, this has its own field called "Business Continuity Planning".

Communication is about telling the people who is involved with the risks about it: *"Propose for the management that they order eight fire extinguishers and set up a training session for the workers."*

Risk monitoring is about keeping track of the risks, this is mainly done by recording the risks so they can be accessed at later stages. When recording the risks, it is usual to use the risk management matrix, which is a table that stores one risk for each row, and has multiple columns describing properties of the risk. These properties should include:

- A unique ID to represent the risk.
- A description of the risk.
- The source of the risk.
- The criticality of the risk (likelihood, consequence).
- Preventive measures.
- Who is responsible for the risk.

And of course, any other properties that might be relevant for the given system. In our case, working with software components and classes, it would perhaps be wise to link the risks with these entities.

Appendix C

An Introduction to the Rational Unified Process

C.0.5 What is RUP?

RUP is a process framework which takes basis in the iterative process of the developing software. The iterative process differs from the traditional Waterfall process.

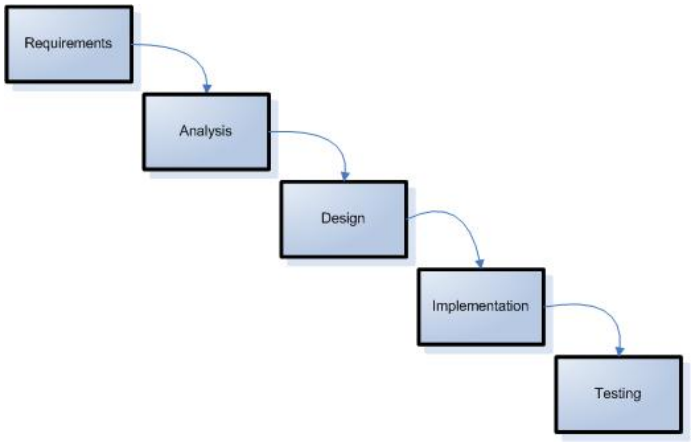

Figure C.1: The waterfall process.

The Waterfall process (Figure C.1) assumes that you first collect the requirements of the system, analyze the problem, then design the solution, implement it, test it and then you are done. Such a process may be effective for small software projects where the requirements are clear and the team fully understands the goals and challenges that exist within the given

project. However, when projects scale up and involve multiple end-users and subsystems, you may encounter a scenario where during the testing of the system, you find out that this system is not what the end-user wanted, or perhaps the system do not meet a specific quality requirement and therefore needs to be redesigned. At this point the system is well into its development, and such changes are costly and may very well produce new problems. In short, one can say that this process tend to delay the detection and treatment of problems.

Two faulty assumptions that this model is based on is that requirements given by the stakeholders will remain frozen during the development and that we can get the best design right on paper the first time, [1].

The iterative process, upon which RUP is built, deals with the problems introduced by the waterfall process by dividing the project into several waterfall processes or iterations (Figure C.2).

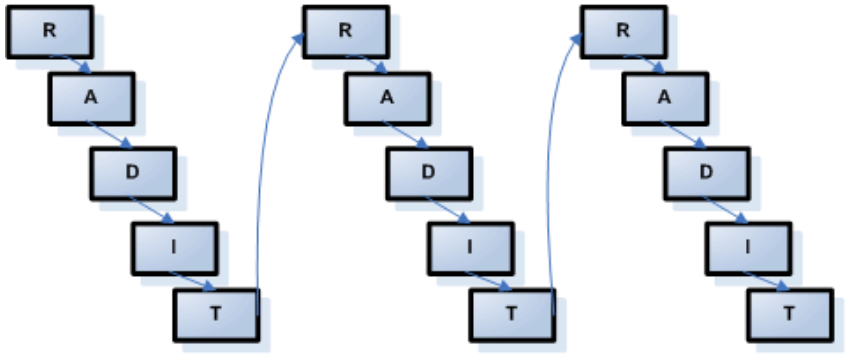

Figure C.2: The iterative method.

An iteration often ends in an executable release. By starting with a skeletal system or a prototype, the developers start fleshing out the system iteration by iteration with the most critical functionality being implemented first. Testing and evaluation is done along the way. In this way problems are not pushed forward in time, but can be dealt with at a time when it is easy and manageable to perform the change, with minimal effects on other components.

RUP is divided into four *phases*:

Inception. Stakeholders agree on requirements and estimate the cost, risk, and schedule of the project.

Elaboration. Establish the architectural base and create an architectural prototype.

Construction. Construct the prototype into the final system and test it.

Transition. Move the system from development to the end user. This includes activities like hardware installation, staff training etc.

Each of these phases consists of a number of iterations, which in turn are like small waterfall projects. The activities performed in the project are divided into *disciplines*. Each discipline creates and uses *artifacts*, which are central pieces of information. The RUP-process consists of the following disciplines:

Business modeling. This discipline include activities that analyze, create, refine, and extend the business processes in the organization. In most cases we do not create the business processes from scratch, but we need to understand them. The discipline is mainly performed early in the project lifetime. Artifacts include the *business model*, *business vision document* etc.

Requirements. This discipline includes analyzing and understanding the needs of the stakeholders and managing the changes in requirements. Just like business modeling, the majority of time spent in this discipline is done early the project, but the management of changes in requirement is a project-long activity. Central artifacts include Use Case models, requirements specification etc.

Analysis and Design. This discipline is concentrated around the analysis and design of the architecture of the system. Some central artifacts include system diagrams and user-interface prototypes. It is performed in all phases, but mainly in the Elaboration phase.

Implementation. The discipline consist of the planning of the integration, the development of components, and the integration of these components. Some artifacts include the integration plan and the builds. The discipline is mainly performed in the Elaboration and Construction phases, but plays a role in all phases.

Test. The Test discipline consists of planning, performing, and evaluating tests of the system. These tests ensure that the components are performing satisfactory with such aspects as quality attributes, user requirements etc. Typical artifacts are *Test Plan* and *Test Results*.

Deployment. This discipline consists of testing the software in its final operational environment. The packing, distribution, and installation of the software and training of end users.

Configuration and Change Management. This discipline ensures that the project artifacts maintain their integrity. When a change is performed it is ensured that this change is updated in all relevant artifacts, that is to maintain consistency between artifacts and the project. It also adds traceability to the project.

Project Management. This discipline consists of the activities of managing the project, such as resource planning, follow-up etc.

Environment. The Environment discipline are the activities supporting and providing the infrastructure, tools, compilers etc (the environment) needed to develop and test the system.

Each of the phases are divided into iterations. Figure C.3. shows the iterative process of RUP.

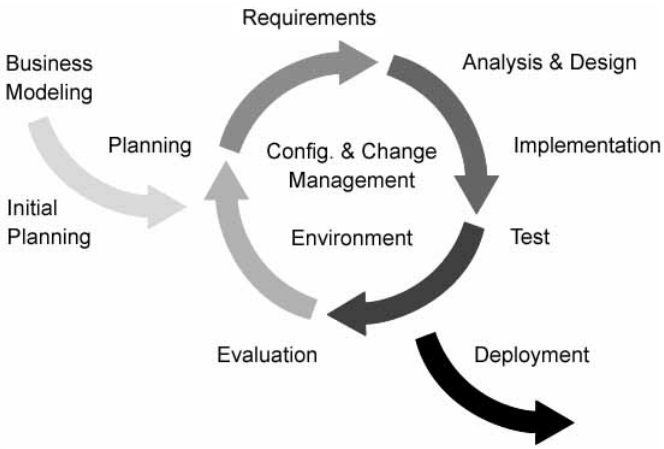

Figure C.3: The RUP iterative process (From www.rational.com).

Before the development is started, there is some business modeling and initial planning. Then the project enters an iterative process of Planning - Requirements - Analysis and Design - Implementation - Testing - Evaluation. When the development is complete, it is deployed. Typically the *inception* phase consists of a single iteration, *elaboration* consists of two, *construction* is divided into multiple iterations depending on the size of the project, and lastly the *transition* phase is often divided into two iterations, [1].

The amount of work spent in each discipline in each iteration varies over the lifetime of the project. Typically there is not spent a lot of time on the implementation discipline in the inception phase (the two first iterations). Figure C.4 indicates how the workload may vary from iteration to iteration

in a typical project. The height of the graphs is the amount of work spent in the given iteration in the given discipline.

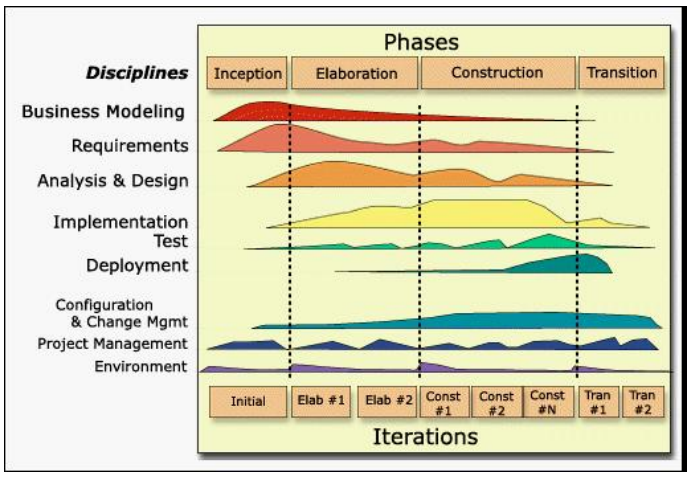

Figure C.4: The phases and disciplines of RUP (From www.rational.com).

RUP is based on the following six best practices, [1].

- Develop iteratively.
- Manage requirements.
- Use component architecture.
- Model visually.
- Continuously verify quality.
- Manage change.

Appendix D

Methods of Analysis

D.1 Preliminary Hazard Analysis (PHA)

PHA (Preliminary Hazard Analysis) is a method performed to find risks early in the planning of a system and is often the basis for later work in the risk analysis. The PHA is done by listing dangers associated with each of the elements in the system. For each danger found, the possible causes, consequences, and severity of the danger are recorded. The method is performed by three consecutive steps:

1. **Collection.** The collection will be done very early in the design of the system. Not every detail will be known about the system, but the different subsystems should be known, as well as the main interactions between them. If the method has been performed on similar cases before, the information from these cases may be of value for the current one.

2. **Analysis.** First we identify every danger that threatens the system. Then we list the possible causes that can produce the situation, their consequences, and a proposal for minimizing or removing the threat. All of these scenarios will then be rated by their severity. You may also add other factors that may be relevant at a later stage, such as cost to implement etc.

3. **Documentation.** The output of the method is a table that lists the results found in the analysis.

In the general method the following aspects should be considered, [2]:

1. Dangerous equipment and materials (such as poisonous materials and cutters)

2. Environmental risks (such as natural catastrophes)

3. Process-risks (such as fires etc)

4. Safety related equipment. (Fire extinguishers, protective wear etc)

5. Operation, maintenance and emergency procedures (Human errors, maintenance procedures etc)

These aspects are important in many engineering disciplines, but when looking at business critical software systems in the enterprise, we are better off by creating our own list of important aspects.

Danger	**Cause**	**Consequence**	**Severity**	**Prev. action(s)**
Database is down	Power failure	Loss of business	Critical	Purchase and install UPS [1] for database server
Database is down	Server is down	Loss of business	Critical	Keep database at two different servers

Table D.1: An example PHA output.

Strengths and weaknesses

The strength of the method is finding *what* could go wrong with the system, but not necessarily *how*. It covers the system wide, but shallow, given this It can uncover major design issues on an early stage.

Complexity.

The method can be performed by one person, but it is important that this person has expertise in all the technologies involved and knowledge about the domain in which the system is going to operate. Because of this, the test is usually performed by a group of people that complement each other with different expertise. The time required to perform the method varies with the size of the system and the scope of the analysis. The method produces a moderate amount of documentation which is also dependent on the size of the system and the scope.

D.2 Hazard and operability study (HAZOP)

HAZOP is usually used in process engineering, such as the development of chemical plants. HAZOP is a systematic method that investigates the sub components (often processes) of a system. HAZOP considers the hazards (external consequences) and operability (internal consequences) of the system. The system is divided into study nodes, which are analyzed using guide-words and parameters.

Take the example of engineering a chemical process with one container containing one liquid, substance A, and one containing another liquid, substance B. These two liquids are transported into a reaction chamber, where a new substance is created, substance C. The creation of Substance C requires the right ratio of mix of substance A and B (Figure D.1).

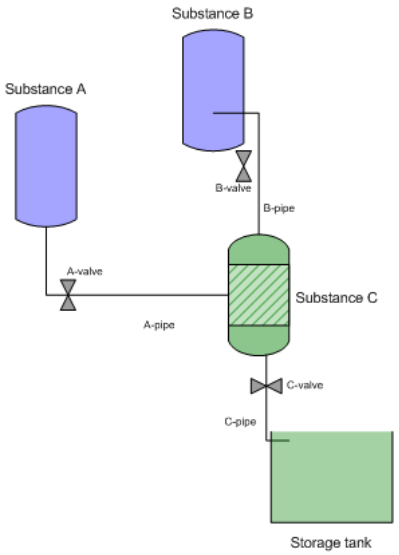

Figure D.1: An example process engineering system.

The system is divided into three *study nodes*, which are the parts of the system where something may go wrong and that we want to investigate into. In this example the three pipelines are the study node (A-pipe, B-pipe, and C-pipe). Then a set of *parameters* is defined. Parameters are properties of the study node we want to study, like flow and temperature. Finally a set of *guide words* is defined. Guide words are used to investigate how the system may differ from the desired state, typical guide words would be no, more, less etc. The HAZOP applies each guide word, on each parameter, for each study node to systematically consider all possible risks. A scenario would be "no flow in the pipe A" or "less temperature in Pipe B". Table D.2 shows

the input of this method applied on the example.

Study Node	Parameter	Guide word	Deviance	Consequence	Preventive Action(s)
Pipeline A	Flow	No	No flow in pipeline A	The production of substance C is stopped, and the substance C in the reaction chamber and storage tank is spoiled	Stop the C-valve and B-valve to stop the process and investigate deviance.
Pipeline A	Flow	More	Abnormal high flow in pipeline A	Overflow in the reaction chamber and the substance C in the reaction chamber and storage tank is spoiled	Try to regulate the A-valve to obtain normal flow, or close the C-valve and B-valve to stop the process.
Pipeline A	Flow	Less	Too low flow in pipeline A	Underflow in the reaction chamber and the substance C in the reaction chamber and storage tank is spoiled	Try to regulate the A-valve to obtain normal flow, or close the C-valve and B-valve to stop the process.

Table D.2: An example HAZOP output.

Relevancy

This method would be useful in the development of enterprise systems as these systems are highly process-oriented. It is not hard to see a similarity between the enterprise system and the process engineering example presented above: Business functions, human or automated, in an organization can be viewed as reaction chambers in which information from different parts in the organization are refined. A typical input would be a diagram that captures the process in the organization and systems, such as a sequence diagram or communication diagram. A set of specialized guide words and parameters will have to be created.

Complexity.

This method should use several specialists with knowledge about the system and the domain it is going to operate in. The meeting should be lead by

a HAZOP-leader which has experience in the method, [2]. The amount of time this method requires would depend on the size of the system. The input of this method is a diagram showing the processes of the system we consider, and the output would be tables of risks.

Appendix E

ZIP file contents

This appendix will describe the content of the ZIP file that is attached to this report.

E.1 Practical Information

The tools are developed on the .NET platform for Microsoft Windows operating systems family. Before running the tools you must have the .NET 2.0 Framework installed. As older versions of Windows may have an earlier version installed, you may have to visit Microsoft's web page to download the "Microsoft .NET Framework Version 2.0 Redistributable Package" to run the tools.

E.2 Executables

The tools are stored in the subdirectory "Tools".

E.3 .NET 2005 projects

The Visual Studio projects of the Enterprise Analyzer, EnterpriseRepresentation and Model Analyzer is stored in the "Visual Studio 2005 Projects" subdirectory.

E.4 Source

Even though the source files are stored within the .NET projects, I have also put the source in a separate folder, as the projects also contain a lot of IDE-related files and resources. This way it is easier to find and navigate the source files.

E.5 Models and enterprise representation

The models used in my test of the tools are stored in the sub directory of "Enterprise Models". The "EnterpriseRepresentation" subdirectory holds a saved enterprise representation (ENT-file) built by these models.

Bibliography

[1] P. Kruchten : *The Rational Unified Process - An Introduction, Third edition (2003)*, ISBN 0-321-1-19770-4.

[2] M. Rausand : *Riskoanalyse - veiledning til NS 5814*, ISBN 82-519-0970-8 (In norwegian).

[3] J.M. Myerson : *Enterprise Systems Integration*, ISBN 0-8493-1347-3.

[4] A. Targowski : *Electronic Enterprise: Strategy and Architecture*, Chapter 7, ISBN: 1-931777-78-0.

[5] C. Giacomo, E. Reggio, M. Iori, A. Salvarani : *Describing and Extending Classes with XMI: An Industrial Experience.*

[6] R. King : *Risk Management*, p. 15, ISBN 0948672722.

[7] A. Jolly : *Managing Business Risk (2nd Edition)*, ISBN 0749440813.

[8] Dr. J. Murphy, Dr. T.W Morgan : *Availability, Reliability, and Survivability: An Introduction and Some Contractual Implications*, The Journal of Defense Software Engineering.

[9] J. Musa. *Software Reliability Engineering.* McGraw-Hill, New York, N.Y., 1998.

[10] P. Higuera Yacov, Y. Haimes : *Software Risk Management (Technical Report CMU/SEI-96-TR-012 ESC-TR-96-012).*

[11] *AS/NZS 4360:1999 (Risk management standard).*

[12] *XML Metadata Interchange (XMI) Specification - An Adopted Specification of the Object Management Group* , Object Management Group Inc.

[13] M. Fowler : *UML Distilled: A Brief Guide to the Standard Object Modeling Language* , Third Edition ,Addison-Wesley Professional ISBN 0321193687.

[14] L. Groth : *Future Organizational Design : The Scope for the IT-based Enterprise*, ISBN 0471988936.

[15] D. Alberico, J. Bozarth, M. Brown, J. Gill, S. Mattern, A. McKinlay : *"SOFTWARE SYSTEM SAFETY HANDBOOK - A Technical Managerial Team Approach."*, Joint Software System Safety Committee.

[16] S. Uchitel, R. Chatley, J. Kramer, and J.Magee : *LTSA-MSC: Tool Support for Behaviour Model Elaboration Using Implied Scenarios*, In Proc. of 9th TACAS, Warsaw, Apr. 2003.

[17] S. Balsamo and M. Simeoni : *Deriving Performance Models from Software Architecture Specifications*, Dipartimanto di Informatica, Università Ca´ Foscari di Venezia, Italy.

[18] G. N. Rodrigues, D. S. Rosenblum, S Uchitel : *Reliability Prediction in Model-Driven Development.*

[19] J. Skene, W. Emmerich : *Model Driven Performance Analysis of Enterprise Information Systems*, Department of Computer Science University College London, London, United Kingdom.

[20] M. Barth : *Performance Assessment of Software Models In a Configurable Environment Simulator*, Institute of Computer Science, Ludwig-Maximilians-Universität, München, Germany.

[21] *UML Profile for CORBA, v 1.0*, formal/02-04-01, Object Managementr Group.

[22] R. Silaghi, F. Fondement, A Strohmeier : *Towards an MDA-oriented UML profile for distribution*, Enterprise Distributed Object Computing Conference, 2004. EDOC 2004, Proceedings. Eighth IEEE International.

[23] B. Selic : *The Pragmatics of Model-Driven Development*, IEEE SOFTWARE, September/October 2003, pages 19 - 25.

[24] L. Fuentes-Fernández and A. Vallecillo-Moreno : *An Introduction to UML Profiles*, European Journal for the Informatics Professional.

[25] I. Majzik, A. Pataricza, and A. Bondavalli : *Stochastic Dependability Analysis of System Architecture Based on UML Models*, In Architecting Dependable Systems, LNCS-2667, pages 219 - 244. Springer, 2003.

[26] T. Hermansen : *Creating more reliable business critical enterprise systems using RUP and System Safety*, Depth study project, Norwegian University of Science and Technology.