NTNU
Innovation and Creativity

Providing a Birds Eye View on the Execution of Distributed, Reactive Systems using Collaborations

Lars Erik Karlsen

Master of Science in Communication Technology
Submission date: June 2006
Supervisor: Peter Herrmann, ITEM
Co-supervisor: Frank A. Kramer, ITEM

Norwegian University of Science and Technology
Department of Telematics

Problem Description

Understanding what is happening during the execution of a telecommunication service can be quite difficult, as there are usually several processes involved that communicate with signals and change their connection over time. Looking at a trace of messages sent between the processes is possible, but for a system of realistic size the number of messages and processes makes it difficult to get an overall picture of what is going on. To tackle this problem, we started in a previous project thesis to use a notation similar to UML 2.0 collaborations for the visualisation of behavior. The realized prototypical implementation demonstrated the potential of the idea, but made also evident that additional presentation mechanisms have to be introduced to keep the visualisation comprehensible. In addition, the prototype had to be adjusted manually to each application individually, as it did not take the abstract specification in form of UML 2.0 collaborations into account.
In this work, the idea should be carried further and refined. Mechanisms to make it possible to keep the overview also in large systems should be suggested and implemented. In particular, the information stored in the collaborations of the system specification should be used automatically, so that a manual adjustment is not necessary anymore. This information can then be used to filter or to abstract the presentation.

Assignment given: 16. January 2006
Supervisor: Peter Herrmann, ITEM

Preface

This thesis was written at the Norwegain University of Science and Technology (NTNU) as part of the Master of Technology study. It was carried out at the Department of Telematics in the 10th semester, spring 2006.

I would like to thank my supervisor Frank A. Kraemer for his valuable advice and comments during this thesis work, and professor Peter Hermann for some final input.

Trondheim, June 2006

Lars Erik Karlsen

Abstract

This thesis studies the use of collaborations as a mean to visualize behaviour of an observed distributed reactiv system. The work is inspired by previous work where an approach using elementary collaborations, that is collaborations with only two participants, visualized a running system. This work investigates how the previous work can be enhanced by adding the possibility to visualize systsem behaviour at different abstraction levels and improve usability, scalabiliy and overview in the visualization.

A monitor for the visulization of distributed reactiv systems will be implemented. A system model and trace information from the obeserved system will be used in order to realize the visualization. By adding the possibility to visualize nested collaborations, a systems behaviour can be visualized at different levels of abstraction. The necessary constraint on the model of a system, and an algorithm for the detection of nested collaboration in the observed system, is suggested and implemented. An automatic approach for the loading of data from a model of the observed system is added to improve uasbility. Additional filtering and layout mechanisms are implemented in order to provide better overview in the visualization. A post-mortem approach to visalization is taken in order to generate a correct visualzation. The speed of the visualzation can be controlled by the user. The possibility to reverse the visualization is also implemented.

Contents

List of Figures

List of Tables

Abbrevations

APSM Association Point State Machines

EMF Eclipse Modeling Framework

GEF Graphical Editing Framework

JDI Java Debugger Interface

MVC Model-View-Controller

NTNU Norwegain University of Science and Technology

OMG Object Management Group

SV Software Visualization

UML Unified Modelling Language

Chapter 1

Introduction

'A picture is worth a thousand words.' - Unknown

As with the rest in life, this rule of thumb applies to software engineering as well. Graphical means in forms of diagrams have been used in software engineering from the very beginning. Today, the common way of software engineering is to first do some design in the form of class diagrams, sequence diagrams, activity diagrams, statecharts or whatever graphical facilities one prefere. Graphical aids are used because they can represent and convey information more effectivly than most textual descriptions. The usual approach to system development is to use vizualization during the design phase, and observation of code or log information when testing and debugging. Testing and debugging are perhaps the most important phases in the system development process. Depending on the tasks a system performs, potential errors or failures could be fatal. In order to get a high quality software, it should be continously checked and tested during development. The cost of correcting errors can be high, especially if discovered at a late time in the process [Vli00]. The cost of and time spent in testing and debugging depends on the size, complexity of the structure and behaviour of a system. A meaningful graphical representation can communicate information more effectivly than the inspection of log information or observing code in debug mode. Utilizing graphical aids in these processes could make them cheaper and faster. Despite the potenial benefits of a graphical approach, visualization is not in widespread use in testing and debugging of software.

In telematics, systems may have simple behaviour, but the number of elements involved can be very large. This make them hard to understand and complicated the task of testing and debugging. To ease the task of debugging and testing, a tool for tracing and monitoring of a running systems was developed in [Nes05]. The system observed consisted of state machines that communicated through asynchronous message passing. Trace information from the running system was used in order to visualize the behaviour. A diagram, similiar to UML sequence diagrams, visualized the behaviour in

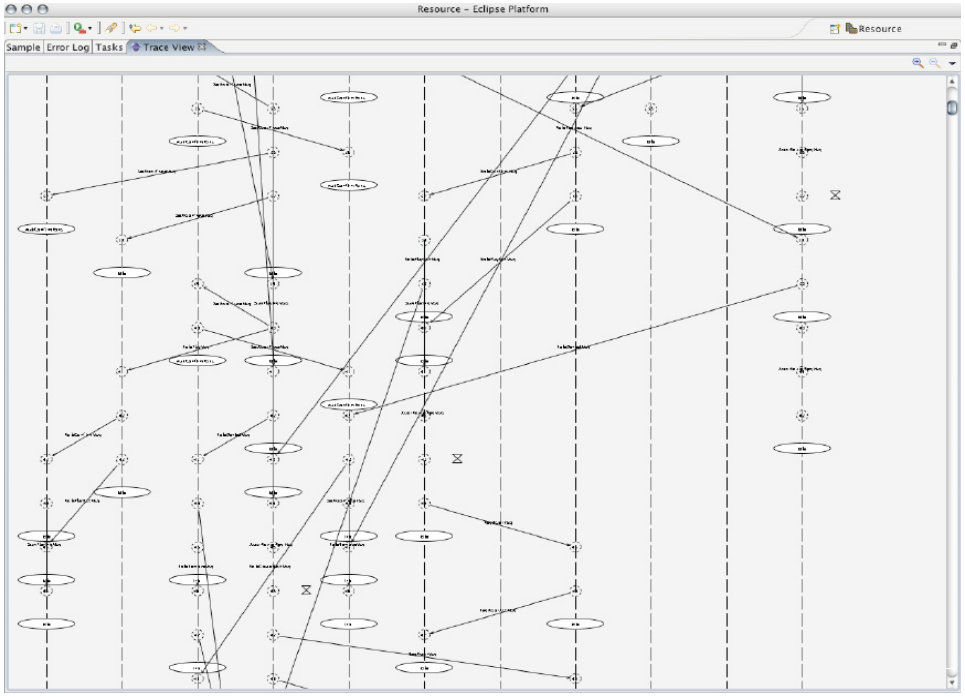

Figure 1.1: A sequence diagram from [Nes05] p. 64.

real-time. Compared to the previous approach of log inspection to decide if the system behaved correctly, the sequence diagram was a great improvement. A problem with the approach, as experienced in similar Software Visualization (SV) approaches [Meh02], was that the UML based diagrams did not scale well to the large amount of information. The sequence diagrams became too large for any ordinary display. In consequence, the overall functionality and behaviour of the observed system was lost in the details of the diagrams. This is illustrated in figure 1.1.

To be able to better visualize the overall behaviour, [Nes05] suggested a visualization based on UML 2.0 collaborations. A collaboration's primary purpose is to explain how a system works. It only includes details relevant to the explanation. In consequence, diagrams based on collaborations should limit the amount of information to the user, and show the behaviour of the system at a higher level of abstraction. [Kar05] studied the collaboration approach to visualization. A prototype of a collaboration monitor that visualized the system behaviour by using collaborations was implemented. The monitor was limited to elementary collaborations[1]. The system structure was visualized as a tree, and the collaborations were visualized as connections between nodes in the tree. An example of the visualization from the

[1] An elemenetary collaboration is a collaboration with only two participants. See section 2.5.3 for more information.

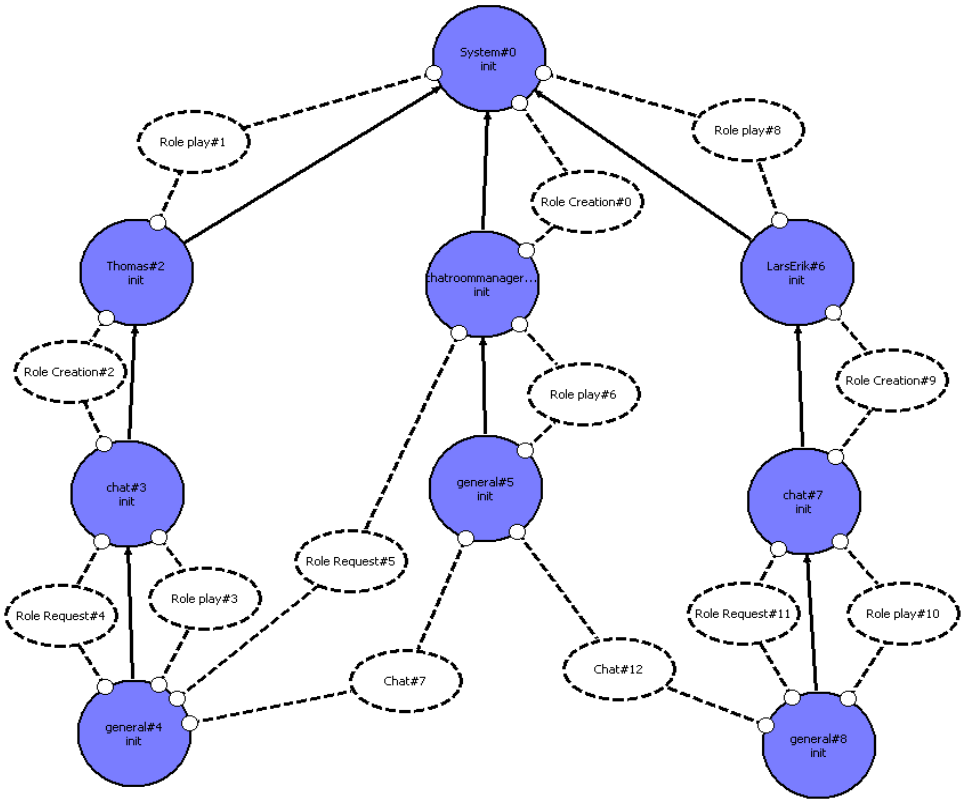

Figure 1.2: Previous implemented prototype visualization of processes and collaborations. From [Kar05] p. 39

prototype in [Kar05] can be seen in Fig. 1.2. The visualization relied on trace data from the running system and information about the parts of the system, known prior to the visualization, in order to create the graphical representation. In addition to help in debugging and understanding of system behaviour, the tool could be useful in demonstration of systems and for educational purposes.

The visualization based on elementary collaborations did reduce the amount of information compared to the diagrams in [Nes05]. The prototype also had some fuctionality (filtering, zooming and layout) to improve the usability and overview. By only showing elementary collaborations, the prototype showed the behaviour at the lowest level possible with collaborations. [Kar05] suggested the addition of nested collaborations in order to visualize the system behaviour at a higher level of abstraction, and improved filtering mehcanisms to deal with problems of scalability.

In this thesis the idea from [Kar05] will be studied further and refined. How additional information in the UML model description of an observed system can be utilized to improve overview and usability of the visualization will be explored. The previous prototype will be discussed in order to find mechanisms to increase usability, scalability and overview. To improve usability, an automatic approach to loading of data from a systems UML model will be examined. In the prototype in [Kar05], this had to be done manually. To provide better overview and system behaviour abstraction, the additions of nested collaborations in the visualization will be investigated. With nested collaboration the behaviour of a system can be specialized at different abstraction levels. The necessary system model constraints and an algorithm for the detection of collaborations in a running system will be suggested and implementet in order to visualize nested collaborations. To provide additional overview, improved layout, filtering mechanism for collaborations, framework messages and processes will be suggested and implemented. A sample application, in accordance with the system model constraints defined in the thesis, will be designed and used for testing. Based on the result of the testing, the value of the implemented mechanisms will be discussed, and possible solutions to existing weaknesses that could improve the monitor will be suggested.

1.1 Outline of the Thesis

The rest of this thesis is structured as follows:

Background Study

Presents the background study for this thesis.

Overview and Scalability

Analyzes the previous approach to visualization and suggest possible solutions to existing problems.

Design and Implementation

Present the design and implementation of the visualization tool. Necessary algorithms and constraints are explained.

Testing

Demonstrates the use of the visualization tool with a sample application.

Discussion

Discusses the implemented visualization tool and the solutions chosen during development.

Conclusions

Presents the main achievements in this thesis and proposes some future works related to this thesis.

Chapter 2

Background Study

This chapter presents the work preceeding this thesis. Some of the contributions and observations from these works will be used in this thesis. Different aspects of SV in general together with a look at some UML based SV tools and the different technologies used in the realization of the collaboration monitor is presented. Finally, the Ramses tool suite and the specification of the UML model in Ramses is explained.

2.1 Previous work

This section gives a presentation of the previous work in [Nes05] and [Kar05]. [Kar05] were based on the part of the contributions in [Nes05]. This thesis will use some of the contributions in [Nes05] and is based on the observations and experiences from [Kar05].

2.1.1 Trace Visualisation for distributed State Machines

[Nes05] studied the use of sequence diagrams as a way to monitor and visualize the execution behaviour of distributed systems. The observed system consisted of finite state machines that communicated with each other through asynchronous message passing. Trace information from the system was used in order to produce the sequence diagram. This section will cover the main contributions and conclusions drawn in [Nes05], that inspired and are important to the work in [Kar05] and this thesis.

Use of Lamport clock to capture the partial order of events in a distributed system.

In order to monitor a system, trace information from the system has to be collected. A distributed system add additional complexity to that task because it has no global clock so the total order of events in the trace data is unknown. Unreliable communication channels and delay of messages may

also affect the order of events as they appear in the trace data. The mechanism used in [Nes05] to capture the order of events is a concept known as logical clocks. Logical clocks were first introduced by Lesli Lamport in [Lam78]. Logical clocks can capture the 'happened-before' relationship (denoted as $\rightarrow$) defined in [Lam78] as:

- If **a** and **b** occure on the same process, $\mathbf{a} \rightarrow \mathbf{b}$ if the occurrence of event **a** preceded the occurence of event **b**.
- If event **a** is the sending of a message and event **b** is the reception of the message sent in event **a**, $\mathbf{a} \rightarrow \mathbf{b}$.
- For three events **a**, **b** and **c**, if $\mathbf{a} \rightarrow \mathbf{b}$ and $\mathbf{b} \rightarrow \mathbf{c}$, then $\mathbf{a} \rightarrow \mathbf{c}$.

Events with the same lamport time stamps are treated as adjacent events.

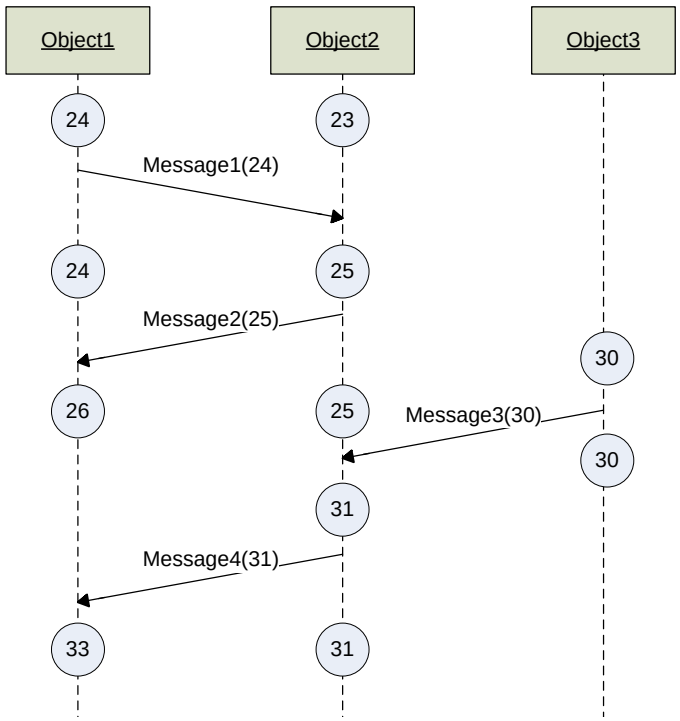

Figure 2.1: An example of lamport clocks.

In [Nes05] an implementation of Lamport clocks were added to the runtime system. A Lamport clock is a monotonically incremented software counter. Every process in a system has its own clock which is incremented according to the following rules [Lam78]:

1. The counter is incremented before each event is issued at the process.

2. When a process sends a message, it piggybacks the value on the message.

3. On receiving a message a process computes the counter as the maximum between its own value and the received value and then applies rule 1 before timestamping the event as received.

This behaviour is examplified in the sequence diagram in figure 2.1. The current value of the Lamport clock is shown in the circles and the piggybacked clock stamp is contained in the messages. 'Object1' sends 'Message1 to 'Object2' stamps the signal with the clock value 24, and sends it to 'Object2'. When 'Object2' receives Message1 it compares the piggybackes stamp with it own clock and saves the highest of the two and increments the new value of the clock by one so the resulting clock value is 25.

Other achievements

A trace monitor, for distributed systems, that visualized trace data as a sequence diagram was implemented. The visualization was created in real-time. The implemented Lamport clocks guaranteed the partial order of the trace events which made it possible to create an accurate sequence diagram. A notation, based on a subset of the UML notation for sequence diagrams, was suggested and implemented in the monitor. The notation is not explained here because it is not relevant to the work in this thesis. An example of the sequence diagram with the notation can be seen in Fig. 1.1 and Fig. 2.2.

Conclusions of the work

[Nes05] concluded that using sequence diagrams as a mean to observe distributed systems may be helpfull to the programmer. The sequence diagrams can make specific parts of the system behaviour more understandable. A drawback was that the amount of information in the diagrams was too great to reason about the overall functionality of the system behaviour. The sequence diagram showed the system at a too low level. This is examplified in Fig. 1.1. [Nes05] suggested that improved filtering and layout capabilities could improve the monitor and reduce the problem of too much information. Addding additional diagrams i.e., collaboration diagram was also suggested.

2.1.2 Collaboration-Oriented Visualization of Communicating State Machines

The work in [Kar05] studied the use of collaborations to visualize the behaviour of a monitored system identical to the once in [Nes05]. The goal of collaborations is to explain how a system works, therefore it only incorporates the necessary details. This property should decrease the amount

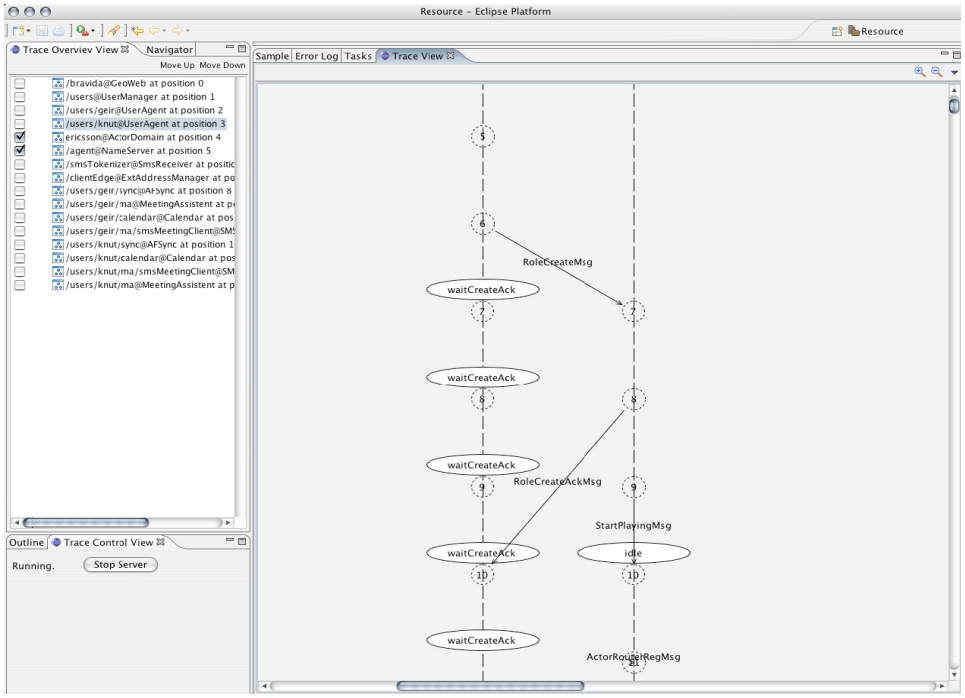

Figure 2.2: The visulization of the system with sequence diagram and process filtering applied.

of information, and could make it easier to understand overall functionality of a system. The goal of the project was to explore the value of such an approach and to what extend it was feasible. Due to time constraints, the work was limited to the visualization of elementary collaborations[1], that is collaborations with only two participants. This section will present the main contributions and findings in [Kar05].

Model constraints and algorithm for the detection of elementary collaborations.

[Kar05] revealed that additional contraints on the UML model of an observed system had to be introduced in order to be able to visualize the elementary collaborations. The defined constraints made it possible do detect elementary collaborations in the running system from the trace information without adding additional tracing information. An algorithm for the detection of elementary collaborations and system structure, based on model and trace information, were suggested and implemented. The defined constraints and the details of the algorithm will be explained, refined and further developed in section 4.6.

Design and implementation of visual monitor using elementary collaborations.

A prototype that used the defined notation and detection mechanisms was implemented and tested with a small test system. The prototype visualized the system structure and the elementary collaborations between processes in the system. The system structure was visualized correctly. The collaborations were visualized sufficiently, but with some inaccuracies. The inaccuracies are discussed in section 3.2.3. Figure 1.2 shows a screenshot of the monitor visualizing the process hierarchy with the collaborations they take part in. Figure 2.3 shows the monitor visualizing the process hierarchy and the signals exchanged between the processes. The screenshots shows that the amount of information were reduced by using collaborations.

The prototype design reused parts of the system designed and implemented in [Nes05]. The following parts were resued:

- **Logging framework -** The logging framework used in [Nes05] was reused in the prototype. The data logged by the framework was sufficient to visualize the different elements, with the added system design constraints.
- **EJBActorFrame -** The patched runtime system in [Nes05] was used as framework for the code generator in order to provide the necessary

[1] See section 2.5.3 for further details on elementary collaborations.

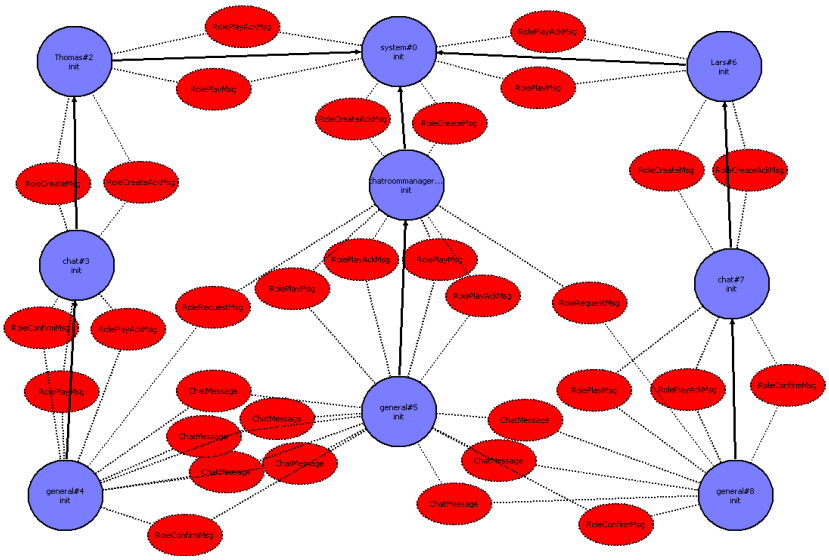

Figure 2.3: Previous implemented prototype visualization of processes and signals.

support for ordering of events in the trace data through the Lamport clock implementation.

- **Overall Model-View-Controller (MVC) structure -** The overall MVC[2] structure of the plug-in with different model levels was reused. The information was stored in different models to be able to support multiple views, and to separate view and system data.

In addition to the visualization, some filtering mechanism and functionality to increase the usability of the monitor were added.

- **Manual layout of figures** - The user could organize the figures on the screen by selecting one or more objects and move them around to get a better overview.

- **Automatic layout of figures** - Support for automatic layout of figures was implemented. The automatic layout was however not very sophisticated and as a result the user often had to use the manual layout tool to really get a good overview.

- **Filtering of collaborations and processes** - A filtering mechanism for collaborations and processes was partially implemented. The filtering mechanism was realised by making the system hiearachical tree structure collapse and expandable. By collapsing a part of the system

[2]See section 4.1.1 for more about MVC in this thesis

tree the user could hide the processes and collaborations in certaint parts of the system and focus on the visible part of the system. Although not fully implemented, the filtering mechanism proved effevtive as a way to reduce the amount of information.

- **Zoom support** - The user could use a zoom function in order to get an overview if the diagram got too large to fit the screen.

Limitations in the prototype

The Ramses II UML model, which the test system was specified in, did not support the specification of collaborations. As a result, the collaborations had to be added manually, grouping the signals into meaningful collaborations. This information was then hard coded into a static structure and used when analyzing the trace information from the logging framework. There were no defined interfaces for accessing the system information that was part of the UML model, so data could not be loaded programaticly when starting the monitor. The same static approach as with the collaboration specification was used to store the system information. The necessary system information was hard coded into a model. As a result of the static approach, the models for collaboration and system information had be manually recoded if other systems were to be monitored. Elementary collaboration did reduce the amount of information compared to using signals (see Fig. 1.2 and 2.3), but they did not reduce the amount of information sufficiently. The monitor did not scale well to an increasing number of processes. The collapse/expand filtering approach could not limit the number of elements at the same level in the hierarchy. Adding support for nested collaborations to increase the level of abstraction, and additional filtering mechanism (i.e., filtering of framework messages, grouping of processes (see chapter 3)) could improve the monitor.

Conclusions

[Kar05] concluded with that the collaboration based approcah to visuliazation could be usefull, but in its current form was somewhat limited. The level of details, which had been the problem with the approach in [Nes05], were reduced but not sufficently. For the visualization to be useful, the collaboration monitor had to be able to support nested collaboration. This would make it possible to visualize the system at a higher level of abstraction. The sample application used in testing was also very simple, perhaps too simple to make a real judgement of the usefullness of the monitor. Most of the signaling were framework specific and did not really say anything about the functionality of the application. The sample application did not have much functionality to visualize. As a result, how well the monitor could visualize the behavior of a observed system, was somewhat unclear. Larger

systems had to be designed and tested and abstraction and scalability problems had to be solved in order to see the full usefullness of collaboration based visualization.

2.2 Related work

The use of collaborations to visualize system behaviour has to the authors knowledge not yet been done by any other. In consequence, there is no previous experiences with such an approach we can make use of. In the area of Software Visualization (SV) however, a lot of work has been done. The main issues concerning the design of SV systems will be presented together with an example of a existing UML based visualization tool.

2.2.1 Considerations in software visualization

In [PBS98] a taxonomy for SV is introduced. The taxonomy classifies different SV systems based upon how a system addresses different aspects of software visualization. The taxonomy can be used as a basis when designing a software visualization system. It contains the most general areas that need to be addressed when designing such systems, and provides some possible solutions. In this thesis the taxonomy will be used as a basis for the evaluation of the final system. The following sections presents the main categories in the taxonomy.

- **Scope -** When designing a SV system the scope of the system has to be considered. Scope of SV system defines the range of programs that the SV system can take as input for visualization. The scope of a system can be further divided into generality and scalability. Generality concern to what degree the the system can handle a generalized range of programs or if it displays a fixed set of examples. Scalability means to what degree the system can handle large examples.

- **Content -** One of the key issues of a SV systems is what data it actually visualizes. In most cases, only a subset of the information about a system is visualized because only certain aspects of a program is interesting. How and when this data is gathered must also be considered.

- **Form -** The form of the visualization is another key issue in developing a SV system. A visualization can be designed for different mediums (i.e., monitor, paper, virtual reality) and the style of presentation and appearance can vary. A system can use different graphical elements, colors, sound, multiple dimensions and animation to convey information. If a system can visualize fine-grained details, it may be just as important to be able to filter out fine-details in order to see the bigger

picture. Temporary hide sections that are not of immediate interest is another way to limit information. By supporting multiple views, a system might offer both a coarse-grained and fine-grained visualization.

- **Method -** SV systems can employ different ways of creating the visualization. It can be coded from scratch (i.e., a user writes a special program in order to visualize a particular program), or it can be build from existing libraries. The user ability to customize the visualization, the intelligence of an automatic approach and how the visualization is specified is also important. The connection between the actual program code and the visualization can affect the useability and generality of a SV system.

- **Interaction -** A SV system should provide the user with different ways of interacting with the visualization. Filtering information, control temporal aspects, speed and direction of the visulization are all desirable features in a SV system.

- **Effectivness -** The effectivness of a SV system must be measured in the context of it's purpose. There might be several aspects of effectivness, but how well the SV system communicate information to the user is the most important measurment of effectivness. An experimental evaluation might be necessary to get an accurate estimate.

2.2.2 JaVis

JaVis [Meh02] is an environment for visualizing and debugging concurrent Java programs. JaVis uses tracing for information gathering and UML for visualization. Java Debugger Interface (JDI) is used to collect the trace information. The JDI allows to trace remote and running programs [Mic06]. By using the JDI, no additional modification of the source code is necessary. The tracing mechanism is non-invasive, and the visualization is generated post mortem when the tracing is finished. JaVis automatically analyzes the trace for deadlocks.

The visualization was made possible through integration with the standard UML CASE tool Together. The visualization was based on UML sequence and communication diagrams[3] (see Fig. 2.5 and 2.4) which had been extended to model thread sychronization and deadlocks. The conclusion in [Meh02] were that using the JDI to collect trace information was effective. A problem with the visualization was that the UML diagrams did not scale well with huge amount of information. However, by focusing on errors the

[3]Since the creation of JaVis the UML specification has been updated. The diagrams referred to as collaboration diagrams in [Meh02] are now known as communication diagrams and are not the same as the collaboration diagram in [Gro04]

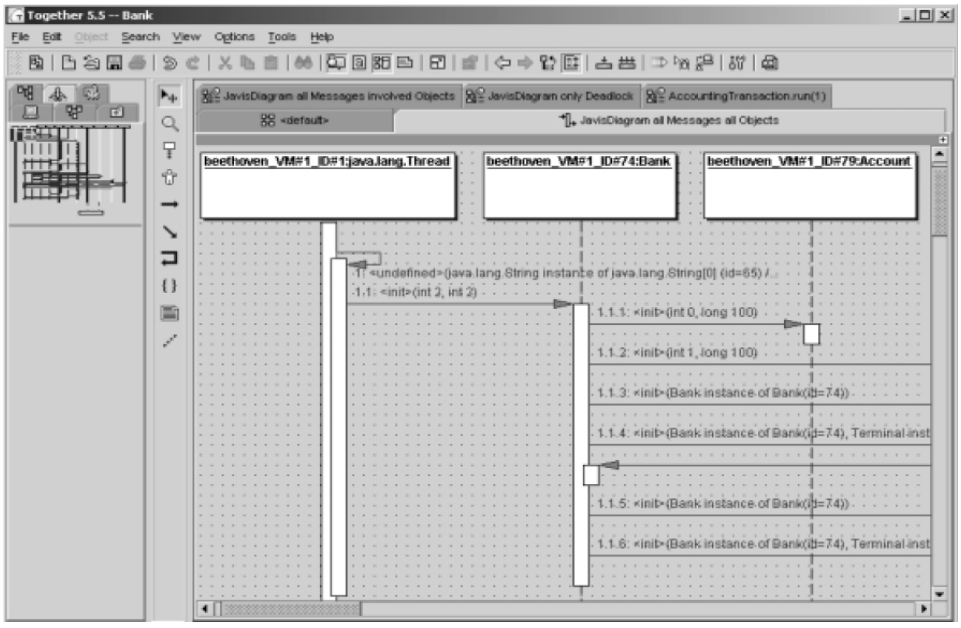

Figure 2.4: A sequence diagram visualized in JaVis, from [Meh02] p. 169

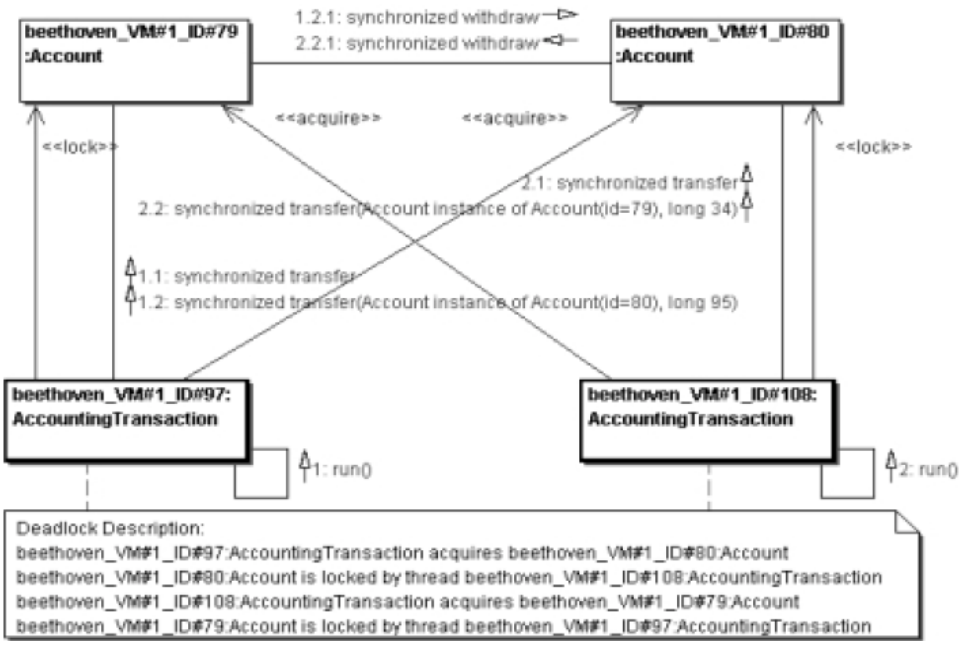

Figure 2.5: A communication diagram from JaVis, from [Meh02] p. 171

amount of information was reduced. The choice of using a standard tool for the visualization was found to be inconvenient. By using a standard tool, the visualization was locked to the framework that the tool provided which was insufficient and led to compromises. In a future version JaVis would have its own visualization tool.

2.2.3 What can we learn?

The JaVis framework did not try to solve the exact same problem as the collaboration approach in [Kar05]. Still, there is a lot of common issues that both JaVis [Meh02] and the work in this thesis has to deal with. Both JaVis and the collaboration monitor uses tracing for information gathering. The prototype monitor relied on logging information from the running system which was able to send these because of additional code added to the observed system, thus the trace mechanisms used in the prototype was invasive. Using the JDI might be an alternative. JaVis experienced the same problem with sequence diagrams as in [Nes05]. The approach in [Kar05] and this thesis will based on collaboration that should limit the amount of information. The monitor also has its own implmeneted notation, so it should not be limited in the visualization like JaVis that used a standard tool. If the current notation used in the monitor is limited, it can easily be expanded.

2.3 Technologies

This section gives a brief introduction to the different technplogies used in the realization of the collaborations monitor in this thesis.

2.3.1 Eclipse Frameworks

The Collaboration monitor is implemented as an Eclipse plugin. The Eclipse plugin environment is used to add extensions to the Eclipse workbench. With the extensions points in Eclispe you can for example add views and menu items to the workbench [DFK+05].

- **Eclipse Plugin Framework -** Used to add extensions to the Eclispe workbench in order to control the functionality of the collaboration monitor and to add views/editors.
- **Eclipse Modeling Framework (EMF) -** Is a modeling framework and code generation facility for building tools and other applications based on a structured data model, from a model specification described in XMI [Gro06a]. In this thesis it is used to realise the models that store the data from the trace and different views.

- **Graphical Editing Framework (GEF) -** Make it possible to create rich graphical editor from an existing application model [Gro06b]. GEF consist of two plug-ins. The org.eclipse.draw2d -plug-in provides layout an rendering toolkit for displaying graphics. The org.eclispe.gef-plug-in is an interactive MVC framework built on top of Draw2d. Through this plug-in, interaction from mouse, keyboard and the workbench is handled. It also handles display and drawing of underlying models by using draw2d figures. It also adds editing support on top of draw2d and provides tools such as selection. GEF is used to realize the graphical editor and figures used in the visualization.

2.3.2 Log4J

Log4J is a Java-based logging utility and is used primarily as a debugging tool. With Log4J it is possible to enable logging at runtime without modifying the application binary. The Log4J package is designed so that these statements can remain in shipped code without incurring a heavy performance cost. Logging behavior can be controlled by editing a configuration file, without touching the application binary. Log4J is used to capture the logging information sent from the running system durring simulation. For more information on Log4J the reader is refferd to [Fou06].

2.4 Ramses Tool Suite

Ramses is a tool suite for the creation of reactive systems [Kra06]. It is developed at the department of Telematics at NTNU. The tool suite contains different parts all implemented as Eclipse plug-ins. At the time of writing the tool suite consist of the following parts:

- **Model editors** - The model editors lets you create and edit the functionality of your application. The application is stored as an Unified Modelling Language (UML) model. The behaviour of the application are modeled with state machines and composite structures like classes, ports and connectors describes the structure. Different diagrams also visualize the system structure and state machine behaviour. It is also possible to describe system behaviour with collaborations that describe how the different parts of the system interact with each other. The UML model is explained in more detail in 2.5.

- **Code generators** - Code generators can generate code from the UML model for different plattform. At the time of writing the available code generator translates the models to EJBActorFrame, a Java based framework from Ericsson. Both a standard edition for servers and a compact edition for mobile phones exist.

Figure 2.6: Overview of the Ramses tool suite for rapid model drive development of distributed reactive systems.

- **Model checks** - Durring the development of an application, the model is continuously checked for errors by checking mechanism. Different tests are applied to the model to identify error situations.

- **Runtime trace support** - To make it possible to observe the system durring simulation, different trace mechanism have been developed. Currently there exist a working trace monitor that displays the system in a diagram similiar to sequence diagrams, and a prototype of a trace monitor using a collaboration approcah to visualization.

The Ramses tool suite is developed continuously and the latest addition is the collaboration specification mentioned above. Durring the development of the first prototype of the collaboration monitor no such support existed. In consequence the implementation had be simplified and compromises made. The trace support of the tool suite is presented in more detail in 2.1. How the additional information in the Ramses UML model can be used to improve the collaboration monitor will be explored in chapter 4.

2.5 PAX Ramses III

The PAX Ramses III specification [Kra06] gives a complete description of the UML model of a Ramses project. For those unfamiliar with the different UML concepts of the model, a brief introduction is included as the different elements are discussed. For further details on UML the reader is reffered to [Gro04]. The focus will be on the elements loaded from the model at the

start up of the monitor. The remaining elements are not covered in detail and the reader is reffered to [Kra06] for further details.

2.5.1 Classes and Packages

The top level element (project, see Fig. 2.5.1) in the model contains the UML resources in the project. The resource contains three packages which cointains the classes, signals and collaborations. The 'class-package' contains all the classes in the system and their components. The 'signal-package' contains all the signals[4] defined durring the design of the system. The 'collaboration-package' contains the defined colaborations in the project.

```
Ramses Core
 |- project
    |- resource
        |- package
        |   |- class
        |       |- state machine (classifier behavior)
        |       |- property (variable)
        |       |- property
        |       |- port
        |- package
        |   |- signal
        |- package
            |- collaboration
```

Figure 2.7: Ramses Core

Classes

A class describes a set of objects that share the same specifications of features, contraints and semantics [Gro04]. The behaviour of a class can be specified using a state machine. In the Ramses model, a class contains an instance of all collaborations it plays a role in. This instance is a collaboration use of an elementary collaboration and it contains a role binding between the specfic role and a particular port in a class that is the communication path for the information exhange in the specific collaboration.

[4]The programer can also use pre-defined signals from the EJBActorFrame package that is included in all Ramses projects. These signals reside in their own package that det programmer can import.

```
- project
   |- resource
       |- package
           |- class
               |- state machine (classifier behavior)
               |- property (variable)
               |- property (part)
               |- port
               |- connector
                   |- connector end (1)
                   |- connector end (2)
               |- collaboration use
                   |- dependency
           |- primitive type
```

Figure 2.8: Classes

2.5.2 State Machines

A state machine can be used to model discrete behaviour through finite state-transition systems [Gro04]. The UML 2.0 specification contains two types of state machines, behavioral and protocol state machines. The state machines in Ramses are all behavioral and describes the behaviour of the different class instances. The systems designed in Ramses are reactive so all transitions between states are triggered by signals. The transition can contain acitvities such as sending signals or calling a procedure [Kra06]. Figure 2.5.2 shows the different parts of a state machine in [Kra06].

Association Point State Machines (APSM)

An APSM is a state machines that describes the behaviour at a certain association point of a state machines. All collaborations contains an APSM. APSMs are modelled as state machines, but the actions and triggers are limited to send signal action and signal triggers on the transitions. A transition can only contain one of the two. The APSM represent a certain visible behaviour or protocol of a state machine seen from the rest of a system. When declearing signals for the system, the signals them self do not know anything about the order of how they are used. This information is stored in the APSMs. The start and ending signals of a collaboration can also be extracted from this state machine. Start and end signals are defined as:

- **Start-signal -** A signal that triggers a transistion from the initial state of the APSM.

```
|- state machine (classifier behavior)
   |- region
        |- transition
            |- signal trigger
            |- activity effect
                |- call behavior action
        |- transition
            |- time trigger
            |- activity effect
                |- send signal action
                |- send signal action
   |
   |-Activity
      |- call operation action
      |- send signal action
      |- send signal action
   |- Signal Trigger
```

Figure 2.9: State Machine

- **End-signal -** A signal that triggers a transition to the final state of the APSM.

Depending on the APSM description, there can be one or more end signals, but only one start signal. The signals can also be classified as sent or received from the specific APSM point of view.

2.5.3 Collaborations

A collaboration describes a structure of collaboration elements (roles), where each element has a special function, that together accomplish a certain functionality [Gro04]. The roles are played by participating instances and communication paths between them are defined by connectors. The collaboration specifies what properties instances must have to be able to participate in the collaboration. The roles specifies the required features the participating instances must have. A collaboration's main purpose is to explain how a system works and it typically only contains the details necessary to do so. An object can simultaneously play roles in several different collaborations, but the collaborations would only represent the different aspects of the object that are relevant to its purpose. A collaboration can be nested and contain other collaborations. The system behaviour can be described with collaborations in Ramses. Ramses collaborations can further be divided into different types of collaborations (elementary and composite) that can be used together to model the behaviour at different abstraction levels.

Elementary Two-Way Collaborations

A two-way collaboration has exactly two participating collaboration roles. An elementary collaboration does not contain any collaboration uses. A signal in the signal package can only be assigned to a single elementary collaboration. The order of the signals are specified in the APSMs. The collaboration contains one APSM for each of the two contained roles. Only one elementary collaboration can occupy a certain port of an instance of a class at any time.

```
-Project
|- resource
|- package
|- collaboration
     |- collaboration role (1)
         |- collaboration role (2)
         |- state machine (owned behavior) (APSM)
        |- state machine (owned behavior) (APSM)
```

Figure 2.10: Elementary Two-way Collaboration.

Composite Collaborations

A composite collaboration is a collaboration that contains collaboration uses. A collaboration use is an instance of a collaboration and represents a particular use of a collabroation. The collaboration use describes a collaboration in a certain context, and bindes entities to the roles of the collaboration in that specific context. The entities can be structural features of a classifier, instance specifications or roles in some containing collaboration. In the Ramses UML model, the entities will allways be ports that represents the binding between instances of classes and the roles of the collaboration. A composite collaboration can contain collaboration uses of elementary collaborations. The elementary collaborations are the building blocks for composite collaborations. A composite collaboration can also be nested. It can contain collaboration uses of other composite collaborations. The composite collaboration does not have any roles on its own, but rather it contains the roles of the contained elementary collaborations.

```
|- package
    |- collaboration
        |- collaboration role (1)
        |- collaboration role (2)
        |- ...
        |- collaboration role (n)
        |- collaboration use
            |- dependency
```

Figure 2.11: Composite Collaboration

Chapter 3

Overview and Scalability

From the conclusions in [Kar05] we know that the previous prototype design and functionality need refinements in order to make the collaboration approach usefull. This chapter will discuss the problems of scalability and overview experienced in the previous prototype in more detail and suggest possible solutions. The design and the implementations details of the proposed solutions will be explained in chapter 4.

3.1 Improving Scalability

The experiences from [Kar05] showed that additional mechanism should be added in order to improve the overview and deal with the increasing amount of information that could be visualized when large systems are considered.

3.1.1 Nested collaborations

The prototype in [Kar05] supported elementary collaborations. The elementary collaborations did reduce the amount of information compared to the alternativ of visualizing all signals (see Fig. 2.3 and 1.2). The elementary collaborations showed the system behaviour at the lowest level of abstraction in the context of functionality. In order to increase the level of abstraction and improve the presentation of overall functionality, the monitor should support nested collaborations. With nested collaborations, elementary collaborations that in its self represent a very simple functionality, can be part of a nested collaboration that performs a more complex or higher level function. By using both elementary and nested collaborations, the behaviour or functionality of the system could be described and visualized at different abstraction levels. There are no restrictions in [Gro04] on how collaborations can be nested. In theory, the level of abstraction could be infinite. To realise a nested collaboration visually, the existing notation for collaborations in [Kar05] could be used, and the nested collaborations could be represented

as tree structures. The root collaboration would represent the high level function and child nodes (elementary- or other nested-collaborations) would represent the sub-tasks in the collaboration. With collaborations organized as trees, a similar approach to the one used for the system hierarchy filtering (see section 2.1.2) could be applied to collaborations.

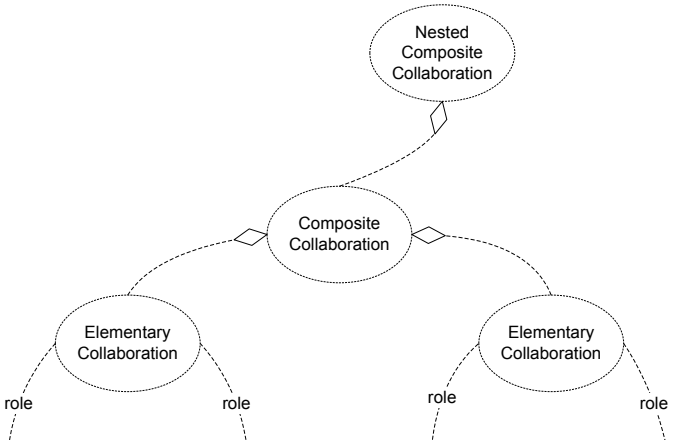

Figure 3.1: A skecth of a nested collaboration.

3.1.2 Grouping of processes

Filtering of processes was supported in previous prototype by expanding and collapsing of the process tree. This filtering mechanism made it possible to filter out child processes. This mechanism however did not scale well with a increasing number of process of the same type that would occure at the same level in the process tree. This problem could be resolved by adding the possibility to group a certain number or all processes of the same type into a single object in the visualization. This would limit the number of visible objects on the screen. The user could focus on the group as a single object, or group a certain number of processes together and only focus on the process not part of the group. The functionality is illustarted in Fig. 3.2. The type and number of elements in the group could be visuliazed together with the group object.

3.2 Improving the Overview

In order to provide better overview the problems of layout, framework messages, control and accuracy of the visualization in [Kar05] will be addressed.

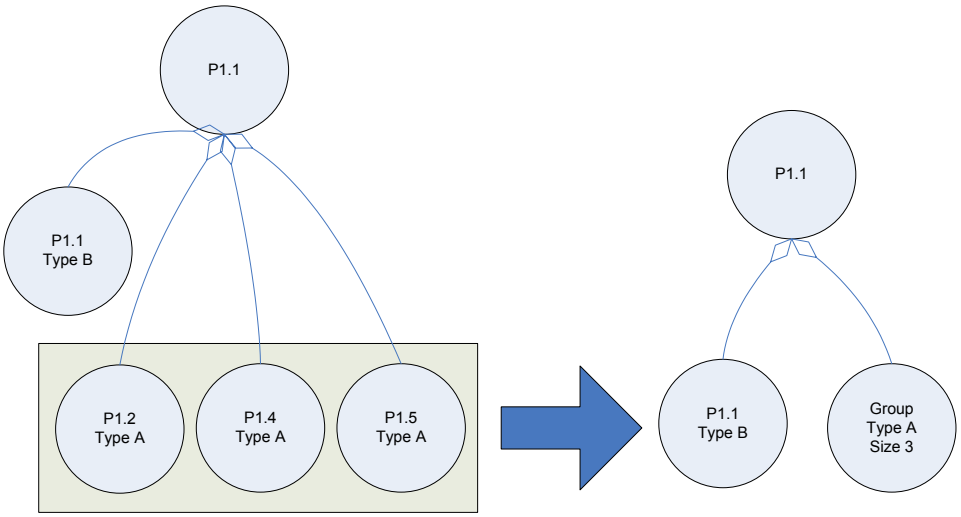

Figure 3.2: Illustration of the grouping of processes feature.

3.2.1 Improved automatic layout of figures

The automatic layout of figures was very limited in [Kar05]. The processes were organized in a tree using a custom layout algorithm implemented by the author. The layout was not very flexible and was only performed the first time a process was added to the view. The collaborations were placed in the view based on the coordinates of the involved processes. The layout algorithm was not very sophisticated and collaborations were usually cluttered together. As a result the user had to do most of the layout by moving the figures around in order to be able to fully observe the system. When the number of processes and collaborations increased this was very tedious work and it impaired and reduced the usefullness of the collaboratin monitor. An improved automatic layout of figures should improve the overview of the system by presenting the objects on the screen in a more orderly way. It should also improve the usability and effectivness of the monitor.

3.2.2 Filtering of framwork signals

With the current code generator in Ramses, the UML model is translated into Java code for the EJBAactorFrame. A lot of framework specific signaling take place i.e., when new processes in a system is created. The signaling might not be very interesting to a user because it does not say anything about the actual behaviour or functionality of the system observed, but is more framework specific. Filtering out these signals should reduce the amount of information and a user of the monitor could focus on the actual behaviour of the system, and completly ignore the framework specific signals.

3.2.3 Event based visualization control

The prototype in [Kar05] visualized the system in real-time. The trace data was analyzed as soon as it was received from the logging server, and added to the visualization. This may work fine for the sequence diagram implemented in [Nes05]. The sequence diagram shows all events that has occured over time and no information is ever removed. The collaboration monitor does not show present and the past like the sequence diagram. It shows the state of the system observed after a certain number of events have occured. Processes or collaborations that is no longer an active part of the system will be removed when they are in-active. The time a part of the system is active can span from a infinite number of events to a single event. If the system is visualized in real-time, the latter example will not be observable to the user, because the object would be added and removed from the visualization to fast to notice. In the previous prototype, the Lamport stamp of the events were used to guarantee that objects would be part of the visualization for a certain number of events. An object in the visualization would remain visible until a pre-defined number of events had occured. This approach was unsatisfactory because it assumed that a system would behave the same way every time. The pre-defined value also had to be tweaked before execution of the system in order to work somewhat sufficient. The problem is illustrated in Fig. 3.3. The terminated collaborations in grey, pollutes the visualization with its presence. When objects are not removed from the view when they are terminated, the visualization becomes inaccurate. This can result in misunderstandings about the behaviour of the system.

To deal with this limitation, a buffered approach to visualization is suggested. Instead of adding the data from the logging events in real-time, the events should be stored in a buffer and added at the request of the user. The objects terminated as a result of the added event, could now be removed from the visualization when they are terminated. There would be no need for a approach like the one used in the previous prototype because any object would allways be visible for at least one 'frame' in the visualization. The user could control the visualization down to event level in similar way that you control a dvd movie at the frame level. Adding an event to the visualization, would skip the visualization one event forward, subtracting an event from the visualzation would skip one step backwards. This mechanisms should solve the problem of removing terminated objects, and would give the user a complete control over the visualization.

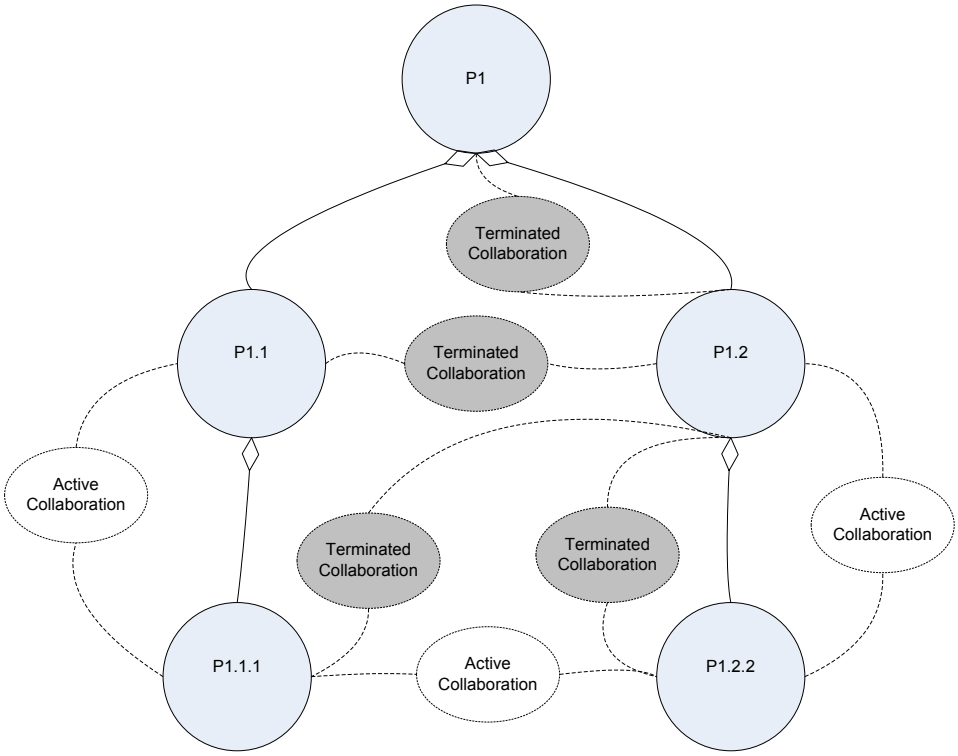

Figure 3.3: Illustration of the problem in the previous prototype.

Chapter 4

Design and Implementation

This chapter will explain the design and implementation of the application. The design of previous prototype will be used as a foundation. It will be expanded and updatet in order to incorporate the additional requirements. Certain implmentation details will be omitted but can be found in the included source code in appendix B.1.

4.1 System Structure

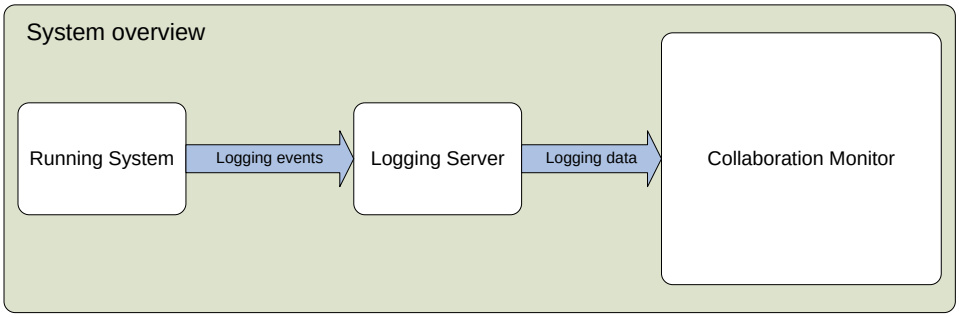

Figure 4.1: An overview of the system structure

Figure 4.1 shows a block diagram of the system structure. The basic design is the same as in the [Kar05]. The 'running system' block represent the observed application specified in Ramses. The details of the sample application used in this thesis can be found in chapter 5. Through the logging capability the observed system sends logging events to the logging server, which forwards them to the collaboration monitor. The collaboration monitor then processes the data and creates the necessary elements that are visualized in a view.

4.1.1 Collaboration Monitor

Figure 4.2 shows a block diagram of the monitor as it was designed in the previous work. The algorithm maintaining the system structure and elementary collaboration depended on data from the static hard-coded system and collaboration description, and trace data from the logging server. Based on the result of the processing, new objects were added or removed from the different models and visualized in a view. The details of the data processing will be refined and explained in section 4.6.

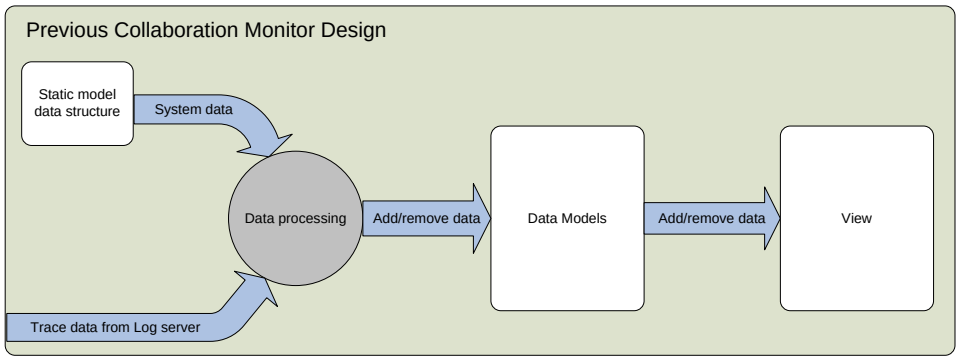

Figure 4.2: An overview of the previous collaboration monitor.

The design of the monitor follows the model-view-controller (MVC) paradigm [Ree79]. The goal of this paradigm is to separate the data model, user interface and the controll logic in three distinct components so that modifications to one component can be made with minimal impact to the others. In MVC design, the controller is often the only connection between the view and the model. The controller is responsible for maintaining the view, and for interpreting UI events and turn them into operations on the model. The different roles in MVC are represented accordingly:

- **Model -** The different information models (data and view information) realized in EMF.
- **View -** The visible objects realized as figures using the draw2d part of GEF.
- **Controller -** The controllers are represented by editparts which are a part of GEF. Editparts are the link between the view and the model. In GEF the editparts display their view in EditPartViewers.

In traditional graphical editors implemented with GEF, the user is the one responsible for editing the model. The collaboration monitor does not work that way. The editing on the model is done by adding or removing trace events from the data model, which in turn affects the view model and

visulization. The user will be able to control progress of visualization, but will only indirectly affect the data model. The user interaction will mainly consist of reorganizing or filtering the visualized objects, which only affects objects in the view model and above.

Information Models

The data in the monitor is stored in different models. The data model contains all data about the system observed as it is seen through the processed logging data. The view model contains all data concerning the visualization. The controllers and figures in GEF does not know anything about the underlying models. The data and view model is the only thing that is persisted and stored [Hud03]. Controllers and figures may be garabage collected or recreated over time. As a result, information such as coordinates, color, line width for example will be stored in the view model. By separating the informations in a data model and a view model, it is possible to add additional views to the monitor without changing any of the existing architecture. An additional view model would simply be added on 'top of' the data model.

The data and view model are realized in EMF and the notification mechanism of EMF is used to notify objects in the view model of changes in the data model. The EMF models are created by specifying interfaces which can be turned into models. From the models, code can be generated with the EMF code generation tool. EMF adds notification to the model objects during code creation. The objects that want to listen for changes in other object must implement the adapter-interface of EMF in order to get the notifications. Objects in the view model listens to changes in the data model and adapts to these changes. If new objects are added in the data model, the corrsponding objects will be created in the view model. If objects are removed or marked as terminated in the data model, the corresponding view objects will be removed. The view model objects also adapts to changes in state information. The same way the view model adapts to the data model, the graphical parts adapts to changes in the view model. The graphical objects represents the controllers for the view model objects and adapts to changes in the view model. If state information is changed in a view model object the graphical part updates its figures. I.e., if state information of a process is changed, the correct data is set in the figure. Another example is setting figures invisible or visible due to filtering mechanisms. As illustrated in Fig. 4.3, the different models are dependent on the model 'below' them in hierarchy. They rely on notifications and adapts to chagenges in that model. There is not necessarily a one-to-one relationship between the different models. For example an object in the data model can be represented by one or more objects in the view model. This complicates the task of mainting consistency between the different models.

Though the overall structure of the different models is more or less iden-

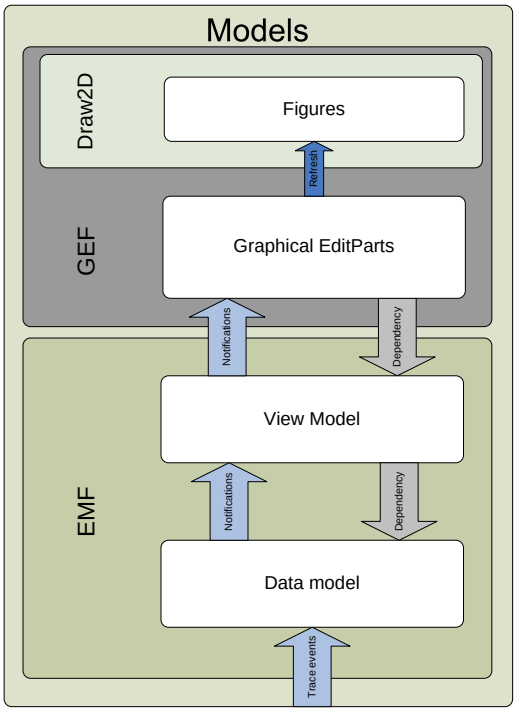

Figure 4.3: An overview of the different models in the system.

tical to the one in [Kar05], the code behind models has been more or less completly rewritten. The previous implementation was incomplete. The consistency between the different models was not properly implemented. This problem had not been detected in the previous work due to the inaccuracies in the processes of removing terminated objects (see section 3.2.3). A re-implementation was necessary in order to incorporate the additions suggested in chapter 3. Especially the notification mechanism, creation and removing of objects between the different models has been improved and should work correctly.

4.2 Design and Implementation Constraints

The existing frameworks (EJBActorFrame, Ramses and the logging framework from [Nes05]) will not be changed in order to realize the monitor. They will be used 'as is'. All solutions, designed and implemented, will not change any of the code of the existing works used in this thesis.

4.3 Loading and Accessing the Model

The prototype in [Kar05] had no support for dynamically loading of data from the UML model of the system observed. As a result, a static approach (see 2.1.2) was taken, which meant that part of the code of the monitor had to be rewritten everytime a new system was to be observed. In this thesis we will solve this obvious weakness by automate the process of collecting the data needed directly from the UML model description of a system. The data will be loaded from the UML model at start-up automaticly, and there will be no need for continues manual adjustments in the code.

The monitor access the UML model through interfaces defined in the Ramses core. At start up the user initates the necessary UML data needed from the UML model of a Ramses project by clicking a button in the menu. In addition to the interface for project access, an UMLHelper package provides some additional helper methods for getting specific information and accesing different part of the UML model.

The loading sequence loads the different packages of the projects and the data contained. From the Ramses UML model specification (see 2.5) we know that the project consist of three main packages. Signals, collaborations and the parts of the system all reside in their own packages. All data from the UML model is stored in hashmaps for fast and easy access durring the processing of trace data. From the signal package all signals are loaded in a hashmap keyed on their name. The signal information in the trace data, only contain the name of a signal as string, so in order to get the right UML signal, the signals are keyed on their string names. Also the signals in EJBActorFrame package is loaded. From the collaboration package all collaboration and the elements they contain is loaded. For elementary collaboration, all signals they contain (described by the APSM) is registered with the specific collaboration. The end signals (defined in 2.5.2) are also registered in a own map. The APSM and role information is also loaded. Composite collaboration are registered with all the collaboration uses of elementary and composite collaborations they may contain. The different components of the system is loaded from the main system package. All classes are registered with their components such as ports and collaboration uses. A overview of the system structure with the addition of loading of UML data included is illustrated in 4.4.

4.4 Parsing Trace Data

For every transistion in any of the processes[1] of the system observed, a logging event is sent to the logging server. The same same logging framework as in [Nes05] is used and the format of the logging events is the same. The

[1] By process we mean state machine and vice versa.

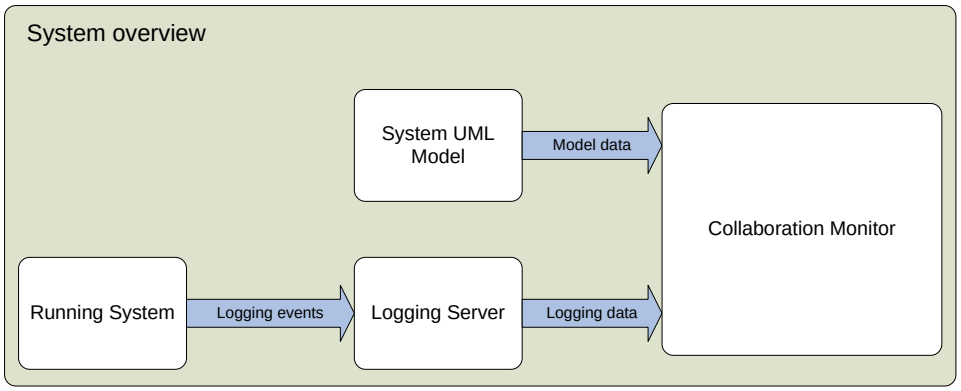

Figure 4.4: An overview of the expanded system structure

complete description of the format can be found in appendix A.1. The following information in the logging events are used by the monitor:

- **ActorID -** Contains the actor address of the class instance of the state machine reporting this transition. Actor address is the unique id of a class instance in EJBActorFrame [MH05]. It is used to indentify processes and the system structure as described in section 4.6.1. The format of the actor address is shown in Fig. 4.5. The actor address consist of three parts:

$$\overbrace{/.../grandparentName/parentName/processName}^{ID} : \overbrace{port}^{Port} @ \overbrace{actortype}^{Type}$$

Figure 4.5: The actor address format.

  - The id consist of the name of the process and the name of all process this process is a child of in top to bottom order.
  - The port field can contain the name of the port a signal is sent or received through. In its current version Ramses and EJBActorFrame does no support the use of port properly so it can not be assumed that the port information in the address is correct.
  - The type of the process. This name corresponds to the class and the name of it's state machine in the model.

- **Current State -** Contains the state of the state machine before the transition took place.

- **New State -** Contains the state of the state machine will enter at the end of this transistion.
- **Received event -** Contains the message(s) received durring the transistion. A message in the receive event also contains the type of the message, and the actor address of the sender and the receiver.
- **Destroy event -** Contains the actor id of a process that has been terminated.

From the trace data we are only conserned with received events. The received event will allways contain the largest lamport stamp because of the happened-before relation. It is allways is the last incremented stamp at the current time in the system. The only information lost by not using the send signal event is signals lost durring transfer. We will however assume reliable transfer of signals and that no signals are lost.

The details on how the information is used can be found in 4.6. The information in the logging events, not listed above, is saved but is not used by the collaboration monitor. At the time of writing it is not included as part of the collaboration monitor, but if found usefull it could easily be added.

4.5 Notation for visualized Processes and Collaborations

The notation used in the monitor is in part based on the notation used in [Kar05], which was in part based on UML. Som additional information will be added to the figures and notation for composite collaborations is added.

4.5.1 Visualizing the system structure

As in previous work [Kar05] the system structure will be visualized as a tree. Using graphs for software visualization is one of the most common and practical forms and it is used in many commercial programming environments [Nor98]. In addition graphs are perhaps the most intuitiv way of visualizing a hierarchy and therefor should be easy to understand for both experts and novice users of the monitor. The notation used for processes are shown in Fig. 4.6. The child-parent relationship is visualized as a connector. The parent end of the connector has a diamond decoration, similiar to the UML containment notation (see Fig. 14-122, p. 324 in [RJB04]), symbolizing the containment. Figure 4.7 shows an example of the process hierarchy visualized in the previous prototype.

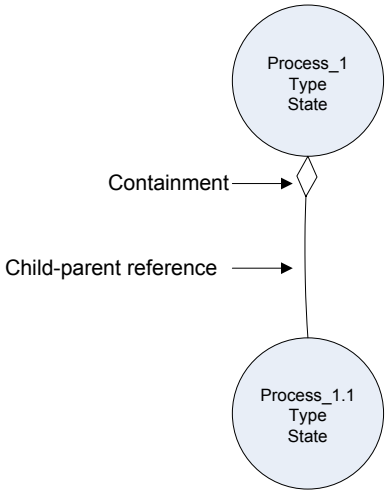

Figure 4.6: Notation used for processes.

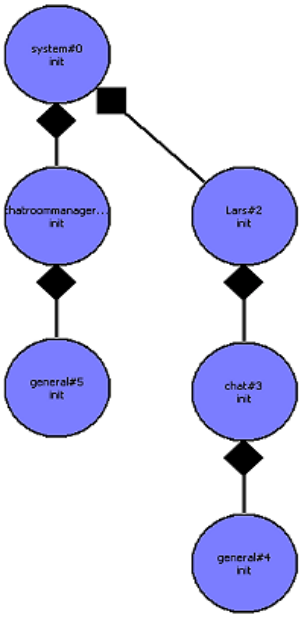

Figure 4.7: The system structure visualized as a tree in previous prototype.

4.5.2 Visualizing collaborations

Elementary collaborations are visualized using the notation in [Kar05]. In addition, role information is included in the figures. The collaboration is represented by a dashed ellipse, The roles are represent by dashed connectors between the elementary collaboration and the participants. The role name is represented by a label on the connector. At the end of the connectors, a circle represent the port that is used in the elementary collaboration (see Fig. 4.8). The notation is similar to the one used in [Gro04] (see Fig. 9.12 p. 166).

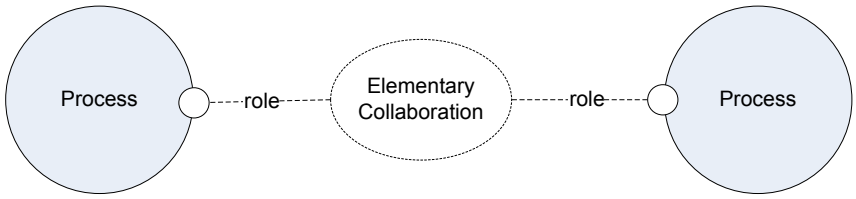

Figure 4.8: Illustration of the notation used for elementary collaborations.

In addition to the elementary collaboration, composite collaborations are visualized. For the same reasons as for system structure (see section 4.5.1) a composite collaboration is visualized as a tree. The root of the tree represent the composite collaboration that contains the other composite and elementary collaborations. The containment relationship is visualized the same way as the parent-child relationship for processes. A dashed connector between a composite collaboration (parent) and the contained composite and elementary collaboration (children) represent the relationship. At the parent end of the connector, a figure similar to the UML containment symbol is visualized.

4.6 Trace and UML Data Processing

The visualization relies on information from the UML model of the observed system and trace data. The information combined will make the discovery of system structure, active collaborations and the visualization possible. Maintaining the system structure is relativly simple. To maintain active collaborations however additaional constraints on the UML model must be defined.

4.6.1 Maintaing the system structure

In the specifications of the trace format in [Nes05], all trace events received by the logging framework contains the actor address of the state machine that experienced the transition. By comparing this actor address with the

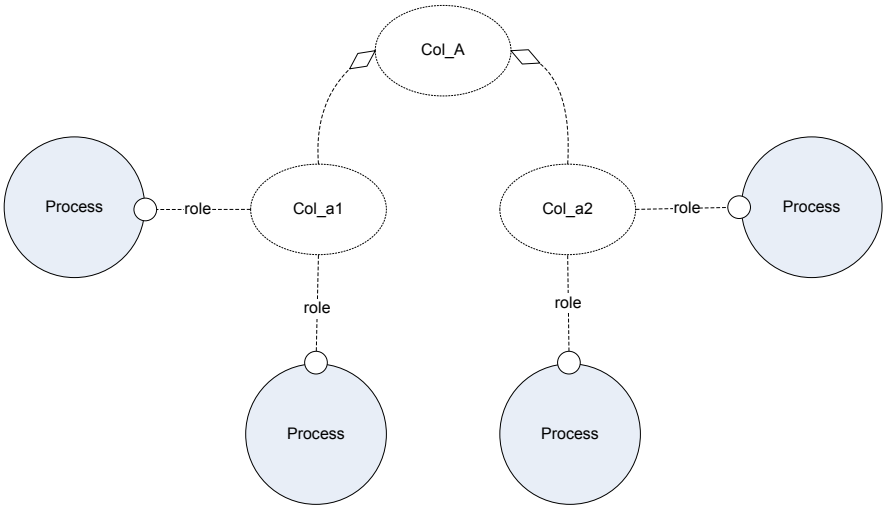

Figure 4.9: Illustration of the notation used for composite collaborations.

actor addresses in the set of existing processes, it is possible to trace the transistion to a process in the data model. If the data model does not contain a process with the address in the transition, it is assumed that this is a new process that just had its first transistion. From the actor address the type of the state machine can be extracted (see Fig. 4.5). When we know the type of the state machine, a new process can be added to the data model and initalized with the default values and ports it has as described in the UML model of the system. The format of the actor addresses (in Fig. 4.5) is hiearchical. By checking the address of the new process against the address of the existing processes, the parent process, if such a process exist, can be found and the new process can be added as a sub-process of its parent. An example of parent-child address relationship is shown below.

```
Finding parent example:

state machine: \a\b\c\d  is child of state machine: \a\b\c

state machine: \a\b\c\d  is NOT child of state machine: \a\b\d
```

If a process with the actor address in the transition in the logging event already exist in the data model, the process is updatet with the state information cointained in the transition. If a processes is terminated durring run-time, the transition in the logging event contains a terminated event. When a terminate event is received, the terminated state machine and all children, if any, are marked as terminated in the data model, and removed from the visualization. This way state and system structure is maintained during observation.

4.6.2 Detecting collaborations

The trace events received from the logging framework gives information about what is currently happening in the observed system. As seen in section 4.6.1, the system structure can be maintained by using trace data and UML data combined. This was possible because there was a relationship between the UML model and the trace information (actortype field in actor address), and a relationship between trace events and the process instances in the data model (actor address). In order to be able to visualize the collaborations that are taking place in the observed system, the same relationships between trace events and collaborations in the UML model, and trace events and collaboration use instances in the data model must exist. As with processes, we first have to locate the right type in order to create it, and later relocate it in the data model in order to update or remove it. With elementary collaborations this is in most cases trivial. With composite collaborations however, the task gets more complicated and additional constraints on the UML model must be defined.

Active collaborations

In the following sections collaborations will be described as active or in-active. Active and in-active collaborations are defined as followed:

- An elementary collaboration use in a class is considered active if a signal that is part of the collaboration the collaboration use represent is received by a process. It is considered in-active when the end-signal of the collaboration has been received.

- A composite collaboration/collaboration use is considered active as long as any of the contained elementary collaboration uses are active. When no elementary collaboration uses is active, the composite collaboration/collaboration use is considered in-active.

Elementary collaborations

(Må rydde opp i terminologien her, collaboration, collaboration use, collaboration use instance, process, state machine class..) The UML model specifices the following constraints on the UML model of as system [Kra06]:

- **System design constraint 1 -** A signal can only be part of a single elmentary collabortion.

- **System design constraint 2 -** At any given time only one collaboration can occupy a port.

These necessary constraints were added in [Kar05]. This means that there is a one-to-one relationship between signals and elementary collaborations defined in the UML model. Logging events provides the signals received durring transitions. Because of the one-to-one relationship in the UML model, finding the elementary collaboration the signal is part of is easy. A simple lookup in the data loaded from the UML model will suffice. Next the collaboration use representing this particular use of the collaboration in the UML model must be located. Processes are uniqly identified by their actor addresses. Collaboration uses have no such unique id. This means that the particular collaboration use that the received signal is part of has to be identified some other way. [Kra06] states that a collaboration is bound to a port of a process through a role-binding. The port represent the communication channel that all signals sent or received by the process will go through. In addition, [Kra06] specifies that only one collaboration use can occupy a port at any given time. This means that the collaboration use has to be terminated before another collaboration use can use the port. Together with the actor name, the port name in the actor address could be used to identify a unique assocation point at the process, which only one collaboration use could occupy at any moment in time. The information about ports in the trace data however, can not be used (see section 4.4) to identify the collaboration use. The ports only serve as a binding between a collaboration use in a class, and a APSM in a particular collaboration in the UML model.

The correct collaboration use in the UML model must be found by comparing all collaboration uses in the two participating classes with the collaboration that the signal is part of. Every class contains a collaboration use for each collaboration it plays a role. The particpants can be found by matching the actor address of the sender and receiver in a message with processes in the data model. When the correct collaboration use for both involved classes is found, the role the process plays and what port is to be used for communication can also be found. A class can, according to the UML model, contain several collaboration uses of a elementary collaboration (illustrated in Fig. 4.10). This causes problem for the detection of elementary collaboration uses. Since the port information in the logging data can not be used, there is no way of knowing which one of the collaboration uses of a certain collaboration that has been activated. As a result we have to add an additional constraint to the system description in order be able to visualize it.

- **System design constraint 3 -** Any system that want its collaborations to be visualized has to limit the number of collaboration uses of a certain type, that a process takes part in, to a single one.

With this additional constraint we can now find the correct port and role a process will play in all elementary collaborations.

```
|- class
    |- collaborationUse_1:Collaboration_1
        |- role_1 --> port_1
        |- role_2 -->
    |- collaborationUse_2:Collaboration_1
        |- role_1 --> port_2
        |- role_2 -->
```

Figure 4.10: An example of when elementary collaborations can not be detected.

There might be cases where two process of the same class play different roles in the same elementary collaboration (illustrated in Fig. 4.11). In these cases we have to chech the APSM that are part of the role binding in order to find the correct role for each of the participating processes. By comparing the received signal with the APSMs, the APSM that received the signal can be identified, and from the role binding the correct role for the different processes. According to the specification, there can only be one collaboration going on at the time at a specific port of process. If the ports

```
|- class
    |- collaborationUse_1:Collaboration_1
        |- role_1 --> port_1
        |- role_2 --> port_1
```

Figure 4.11: A process type playing two roles in the same collaboration.

that are bound to the roles are currently taking part in a collaboration, this collaboration is by definition the same type that the new signal is part of and the new signal received can be added to the current active collaboration on the ports. If there is no active collaboration on the ports, we add a new collaboration to the data model that uses the ports and has the correct roles.

A collaboration is terminated whenever a signal that is an end-signal for the specific collaboration is received. This signal or signals that terminates the collaboration can be extracted from the APSM that are part of the collaboration description. If such a signal is received, the collaboration is marked as terminated and removed from the visuliazation. The port are now free to take part in a new collaboration.

Composite collaborations

The UML model does not limit the collaboration uses of a certain elementary collaborations to be part of only one composite collaboration. Collaboration

uses of a elementary collaboration can be reused in many different composite collaborations. As a result the signals in a composite collaboration is not necessarily unique to that specific composite collaboration. Figure 4.12 illustrates the problem. The composite collaborations, collaboration_A and collaboration_B both, contain a collaboration use of the elementary collaboration Collaboration_1.

```
|- package
    |- collaboration_A
        |- collaboration use: Collaboration_1
    |- collaboration_B
        |- collaboration use: Collaboration_1
```

Figure 4.12: Two composite collaborations containg a collaboration use of the same elementary collaboration.

In addition an elementary collaboration can exist as a collaboration use of a elementary collaboration in a class, performing a function on its own, and as a collaboration use in a composite collaboration, performing a task as part of a greater more complex task. Figure 4.13 illustrates the problem. Two classes 'A' and 'B' contains a collaboration use of the elementary collaboration 'X'. This collaboration is also part of the composite collaboration 'Y'. 'A' could be using the elementary collaboration 'X' as 'stand alone', while 'B' could be using the elementary collaboration and be part of the composite collaboration 'Y'. If this is allowed in the model, there would be impossible to decide which of the two cases that are currently active, elementary collaboration 'X' alone, or 'X' as part of 'Y'.

```
|- package
     |- Class_A
          |- collaboration use: Collaboration_X
     |- Class_B
          |- collaboration use: Collaboration_X

|- package
     |- Collaboration_X
     |- Collaboration_Y
          |- collaboration use: Collaboration_X
```

Figure 4.13: A elementary collaboration as part of a composite collaboration through a collaboration use and as a elementary collaboration use independent of the composite collaboration.

The same relationships as described above (elementary collaborations

to composite collaboration) can be possible for composite-to-composite collaborations. A composite collaboration can occur as a use in several other comsposite collaboration uses. As with elementary collaboration, it is the signals that identify a composite collaboration. The composite collaboration contains all the signals of the collaboration it contains. With no restrictions on the UML model assuring that signals are unique for both composite and elementary collaborations detecting composite collaborations would be impossible. In order to make it possible to detect the composite collaboration we have to define some additional system design constraints.

- **System design constraint 4 -** Only one composite collaboration can contain a collaboration use of a certain elementary collaboration.
- **System design constraint 5 -** Only one composite collaboration can contain a collaboration use of a certain composite collaboration.

With this additional design constraints, the relationship between a signal and any type of collaboration should be a one-to-one relationship. It will be possible to decide what elementary and/or composite collaboration that is currently active based on a signal. The signal would decide the elementary collaboration and the elementary collaboration would decide the composite collaboration. All composite collaborations would be organized as trees with unique nodes.

When an elementary collaboration is detected and the collaboration use intance is created and addded to the data model, as described in section 4.6.2, we have to check if it is part of a composite collaboration in the UML model. If it is, the elementary collaboration use has to be added as a sub-collaboration of the composite collaboration it is a part of. In the same way as we add signals to existing elementary collaboration uses in the data model, we must add elementary collaborations to the right composite collaborations in the data model. This means that we somehow need to be able to identify this composite collaboration in order to check if it exist in order to retrieve it, or to decide if we must add a new one to the data model.

An instance of a elementary collaboration use in the data model could be indentified by looking up the sender or receiver of a signal and checking the ports that was associated with the specific collaboration use. The objects in the data model was identified by the actor address of the participants. Composite collaborations also has to be identified by their participants. If participants are allowed to join and leave composite collaborations of the same type, it might be impossible to decide which composite collaboration the elementary collaboration use the participant is a part of, belongs to. To avoid this problem another constraint on the UML model will be added.

- **System design constraint 6 -** A process that takes part in a elementary collaboration use that is part of composite collaboration can

not be part of any other composite collaborations of the same type while the composite collaboration is active.

Another problem occures if we have several active instances of a composite collaborations of the same type in the data model and want to add another elementary collaboration use as part of one of them. There has to exist a relationship between the elementary collaboration we should add, and one of the existing composite collaborations. The only relationship we can use are the actor addresses of the participants. In order for such a relationship to exist, one of the participants of the elementary collaboration we want to add, must allready be part of the composite collaboration. Figure 4.14 shows a composite collaboration were such a realtionship does not exist. If the elementary collaboration 'A' has been created first, comspoite collaboration 'Y' is also created and 'A' is added as a sub-collboration. If several instances of 'Y' exist, and a signal triggers elementary collaboration 'B', there is no way to decide which one of the instances of 'Y' we should add 'B' to. A composite collaboration were the necessary realtionship exist is

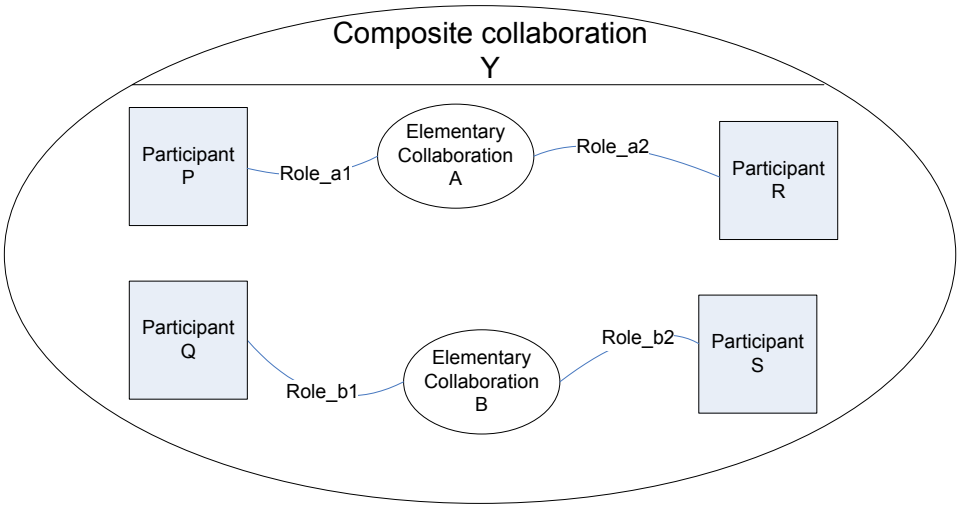

Figure 4.14: An illegal composite collaboration.

illustrated in Fig. 4.15. No matter which one of the collaboration are added first, one of the participants in the next elementary collaboration that should be added will already be a part of the composite collaboration. The order of the activation of the elementary collaboration can not be random. This is illustrated in Fig. 4.16. Legal orders of activation in Fig. 4.16 is ('A','C','B'), ('C','A','B'), ('C','B','A') and ('B','C','A'). If another order would occure, none of the particpants of the elmentary collaboration we want to add to the composite collaboration 'Y' would be a participant of 'Y', and we would not know were to add the elementary collaboration. To avoid cases like the one in 4.14 two additional constraint are added to the UML model.

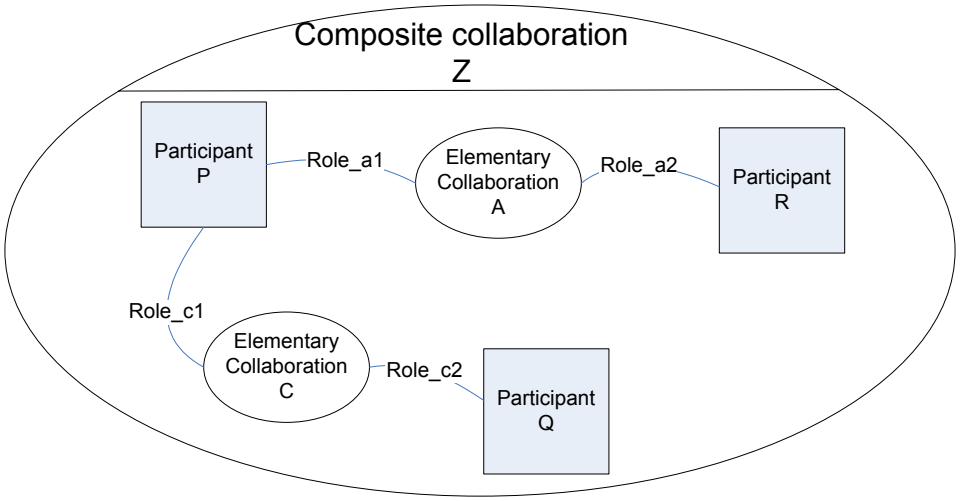

Figure 4.15: A legal composite collaboration.

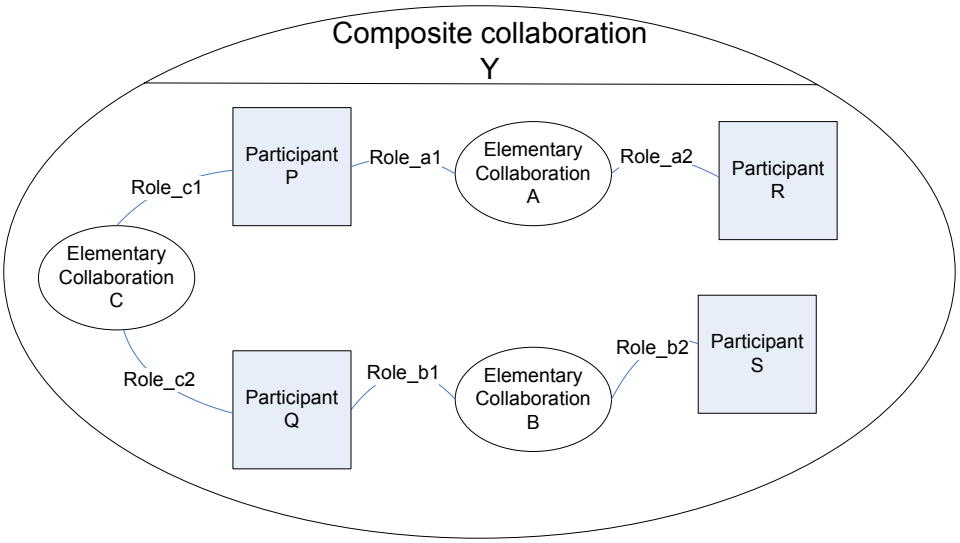

Figure 4.16: A legal composite collaboration if the elementary collaborations are added in the correct order.

- **System design constraint 7 -** All participants in a composite collaboration must be connected directly or indirectly through elementary collaboration uses.
- **System design constraint 8 -** The elementary collaborations cointained in a composite collaboration must be activeted in a order that guarantees that one of the participant is already a participant of the composite collaboration. An exception to this constraint is the activiation of the very first elementary collaboration in the composite collaboration.

With these constraints added to the UML model, we should be able to detect and visualize all active collaborations in the running system observed and remove them when they are in-active.

4.7 Playback control of Visualization

The buffered approach suggested in section 3.2.3 will buffer all trace information and add them at the user request. Figure 4.17 shows an overview of the design.

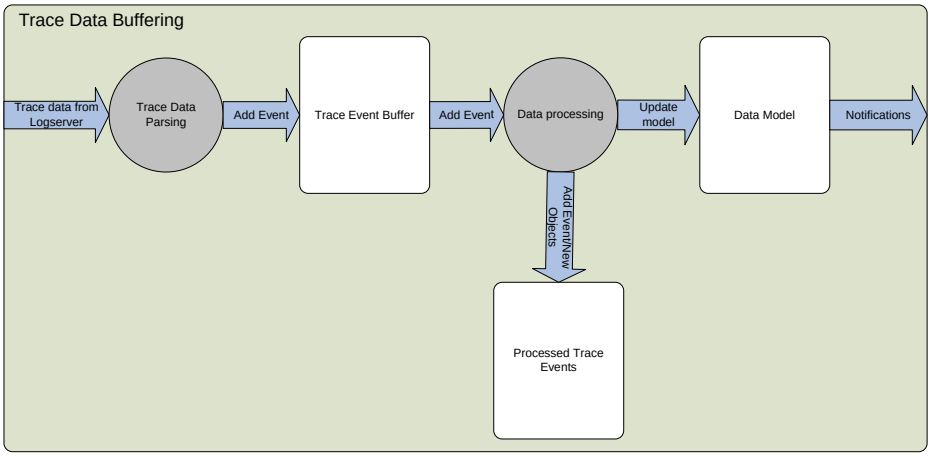

Figure 4.17: Overview of the trace event buffering.

The data received from the logging server is parsed by the monitor. The necessary information is extracted and stored in event objects. The event objects are placed in the trace event buffer. By clicking on the button for adding events, the trace event with the lowest Lamport stamp is taken out of the buffer and processed by the monitor. The resulting updates and new objects are added to the data model that notifies the rest of the system. In addition the trace event added to the monitor is stored in another buffer along with the resulting new data model objects. By clicking the step

backward button, the changes of the most recent added trace event can be reverted. The trace event subtracted is put back into the trace event buffer and can be added again.

4.7.1 Lamport stamp sorting

The Lamport logical clock in EJBActorFrame guarantees causal ordering of the logging events from the system obeserved. The EJBAactorFrame framework piggybacks the Lamport stamp on the event data received by the logging framework. The event data however is not received in the same order as the Lamport stamps. In previous work [Nes05] the Lamport stamps were used as reference point for drawing the the different elements of the sequence diagram. When event data are added to the diagram in a potensial wrong order, the diagram might be incorrect at a certain moment in time. When all event data are processed, the diagram would however be correct. When visualizing the system with collaborations, out of order event data can corrupt the visualization. Because the elementary collaborations are started and terminated based on the signals they consist of, the signals must be added to the visualization in the correct order to guarantee the correctness of the visualization. If i.e., a signal that is an ending signal for the collaboration would be received prior to the other signals of the collaboration, the resulting visulization would be incorrect. The collaboration would be created and removed at the same time, and the user would not be able to observe it. When the remaining signals of the collaboration is received by the trace editor, they would trigger the creation of a new collaboration. This collaboration would then not be properly removed because the ending signal has already been processed. To avoid this problem the trace events in the trace event buffer has to be sorted according to the lamport stamps. When a trace event is added to the buffer, it has to be inserted at the correct index. This should guarantee that the trace events are added to the visualization in the correct order and visualize the system observed correct. This is illustrated in Fig. 4.18.

Adjacent events

Adjacent events are events with same Lamport stamp. The order within these events are unknown, and it does not matter to the visualization. The visualization will not be corrupted by adding these events in a random order. The Lamport stamp guarantees that adjacent events will not effect each other because they can not occure between to processes that are communicating with each other.

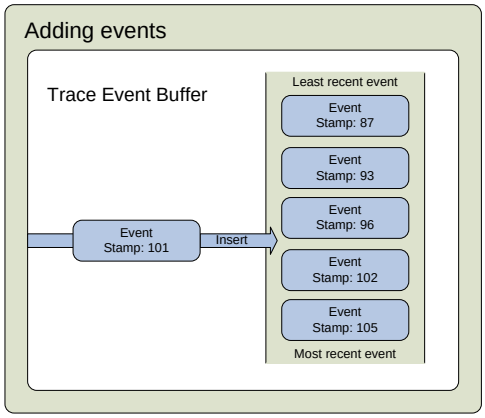

Figure 4.18: Inserting event into the trace event buffer.

4.7.2 Real-time or Post Mortem Visualization

When we add trace events to the buffer instead of adding them directly to the visualization, the visualization will not be real-time. The visualization is semi-post mortem. The user dont have to wait until the entire system run has finished. A trace event can be added as soon as it is placed in the buffer. Due to the problem of out of-order-events described in section 4.7.1, it is probably safest to let a certain number of events be added to the buffer, in order to resolve any out-of-order issues, before adding them to the visualization. Errors can however occure, if events are added when the system is highly active. The only way to guarantee that the system is visualized correctly is to do real post-mortem visualization.

4.7.3 Additional features

With the event-control mechanism you can skip back and forth in the visualization. When observing a very activ system, the number of events in the data buffer can be very large. Adding single events to the visualization N times might be time consuming and tedious. In order to improve the mechanism further, the ability to fast forward the visualization and to skip to 'interesting events' has been added. The user can add all events currently in the buffer or select a part of the system (a process), and skip to the next event that the process is a part of.

4.7.4 Elaborating the Design

An overview of the new design with the additions presented in section 4.3 through 4.7 is illustrated in Fig. 4.19. The additions of the UML loading and trace event buffering is included.

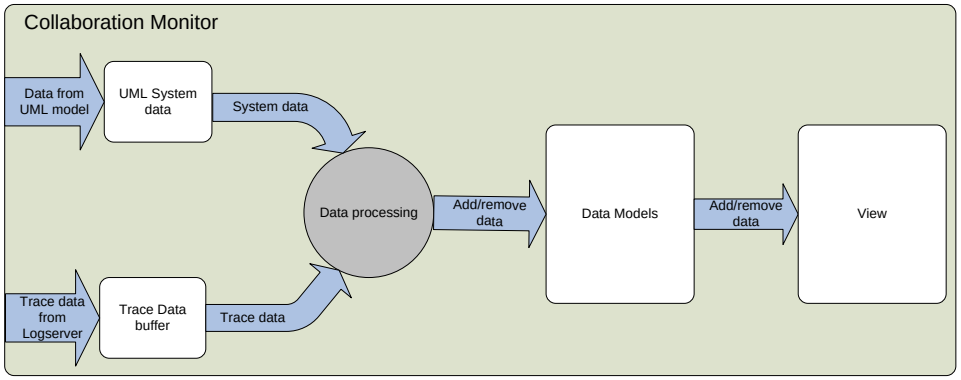

Figure 4.19: An overview of the extended structure of the collaboration monitor.

4.8 Improving the Overview

We previously stated the need and importance of improved overview in the monitor. The design and implementation of the different mechanism suggested in 3 and the existing mechanism that was not fully working in the previous prototype are explained.

4.8.1 Layout of Figures

From the UML model of a system (see section 2.5) it is clear that both the parts of the system and the collaborations they take part in, are organized in a hierarchy. Processes can contain processes and composite collaboration can contain other composite collaborations or elementary collaborations. Elementary collaborations contains signals. This means that we have a single system structure tree, possibly several collaborations trees of different size and in addition any signals not part of collaborations (i.e., framework signals if we choose to visualize them). This is illustrated in Fig. 4.20. The composite collaboration 'Col_A' contains the elementary collaborations 'Col_a1' and 'Col_a2' and represent the a collaboration tree. The proces hierarchy and the collaboration trees are interconnceted through roles. As a result the structure is no longer a single tree but a graph. This means that there are a number of possible ways to order the layout. There is no single optimal layout of such a graph. The optimal layout would be depended on what underlaying patterns the user want to focus on.

The draw2d package in GEF has algorithms for layout of directed graphs that layout graphs from top to bottom. This is currently the only supported layout algorithm in GEF and it is the one we will use. Implementing another layout algorithm from scratch would take a lot of time and is out of the scope of this thesis. The main system structure tree will be priori-

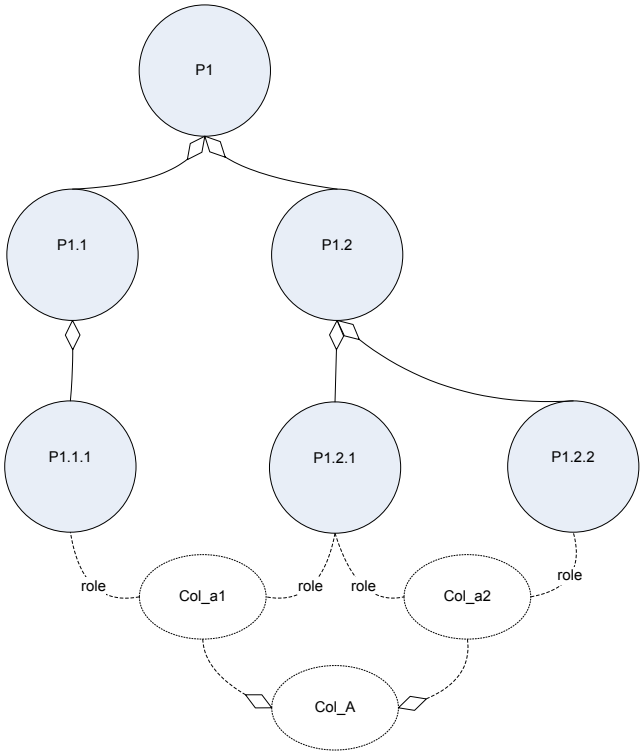

Figure 4.20: The system and collaboration trees connected by roles.

tized in the layout. Collaborations and signals positions will be calculated based on the position of the processess playing a role in the collaboration or sendind/receiving a signal. Collaborations and signals will be placed somewhere on the line between the involved processes. The layout will be performed the first time objects are added to the view and it can be triggered by a button in the menubar. In addition, the possibility to manually move objects in the view is still available.

4.8.2 Filtering Information

In order to improve the overview, mechanism to filter information is implemented. The filtering mechanisms will limit framework messages and provide functionality and system abstraction. Each of the mechanisms will separately filter and limit some part of the visualization.

Filtering Framework Messages

The trace format in [Nes05] specifies that all messages sent or received is tagged with a boolean value. True if a message is a framework message, false if not. This value could be used to filter out framework messages. At the time of writing, this information has not been extracted from the trace data. It is suspected that the trace information might not contain this information due to a bug. An alternativ way to remove the framework messages from the visualization is to ignore the signals in the EJBAactorFrame package when importing the UML model data. The monitor will ignore all signals received in the trace events that are not registered. Another way to filter the signals is to ignore all signals that are not part of any collaboration in the UML model. The latter approach will be used in this implementation.

System Structure Filtering

The main filtering mechanism for the system structure is based on the ideas from the previous prototype where it was partially implemented. The system structure can be filtered by collapsing or expanding parts of the system structure tree. This mechanism will remove all processes that are part of a collapsed process and it will reorganize the collaborations that the collapsed part of the system was participating in. If a process is particpating in a collaboration, the filtering mechansism considers any parent process of the involved processes, as long as it is not a common root for any of the particpants, indirectly involved in the collaboration. The 'common root' is the first process the both of the participating processes are a sub-processes of. This will be reflected in the filtering as illustrated in Fig. 4.22, 4.23 and 4.21.

Figure 4.21 illustrates a system with active collaborations. The system structure is fully expanded. If process 'P1.1.1' is collapsed, the children

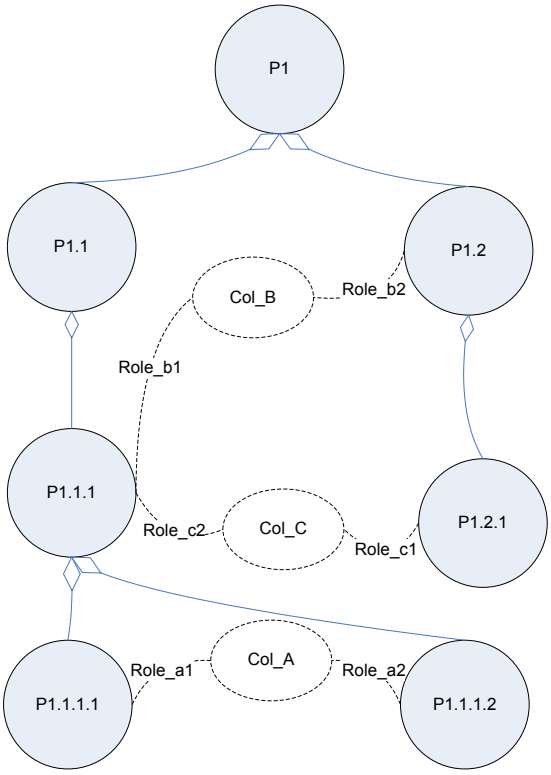

Figure 4.21: A fully expanded system structure with collaborations.

('P1.1.1.1' and 'P1.1.1.2') will be hidden. The collaboration 'Col_A' will also be removed from the view because non of the participants are visible. P1.1.1 is also the 'common root' for the two child processes, so no roles will be relocated. Figure 4.23 shows how roles can be relocated.

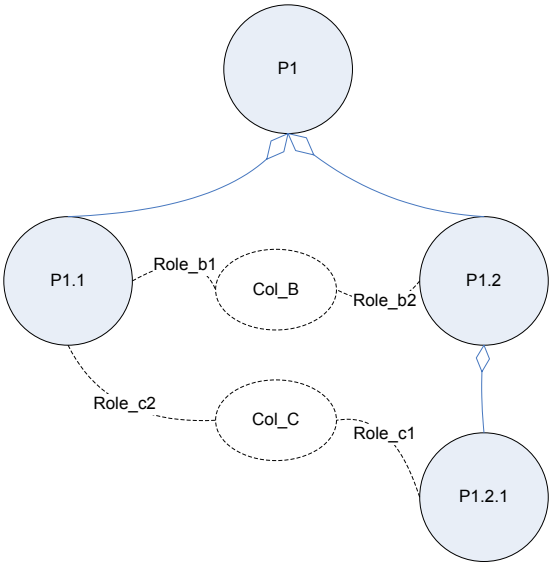

Figure 4.22: The system structure after collapsing process P1.1.1

In Fig. 4.23 the process P1.1 is collapsed. Some of collaboration 'Col_B' and 'Col_C' roles will be relocated. Role_b1 and Role_c2 will be attached to P1.1 to indicate that P1.1 indirectly is part of the collaborations through one or more of its inner parts (P1.1.1).

The collapse/expand filtering mechanism lets the user control the abstraction level of the system structure. Filtering processes at the same level in the hierarchy is however not possible with the mechanism. The grouping mechanism provides this function. Grouping lets the user group together processes of the same type. Processes of the same type will often reside at the same level in the system structure, illustrated in Fig. 4.24. With the collapse/expand approach you can only choose between observing everyone or none. With the grouping function it is possible to represent a group of processes as a single object, illustrated in Fig. 4.25. As a result you can filter processes at a specific level in the system structure.

Filtering Collaborations

The approach for collaboration filtering is based on the tree structure of the collaborations. The top level composite collaboration in the tree describes the top level functionality, while the contained collaborations describes the subtasks. By collapsing or expanding the collaboration or function tree,

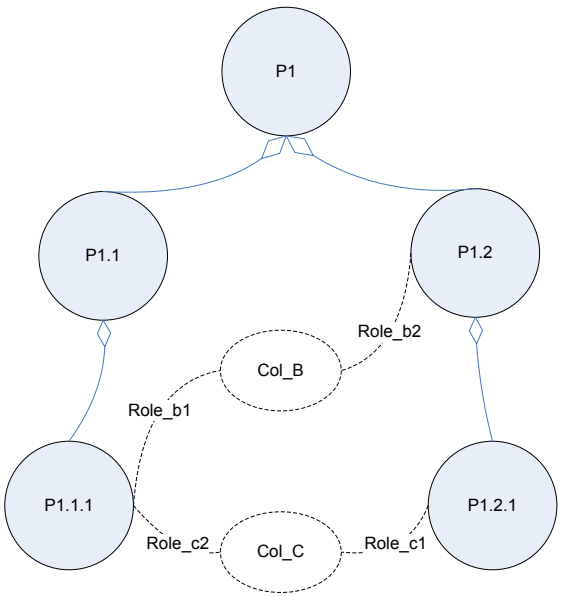

Figure 4.23: The system structure after collapsing process P1.1.

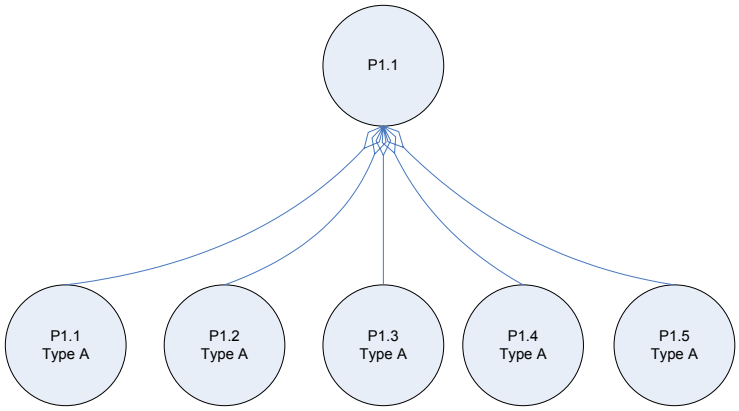

Figure 4.24: A number of processes of the same type at the same level in the hierarchy.

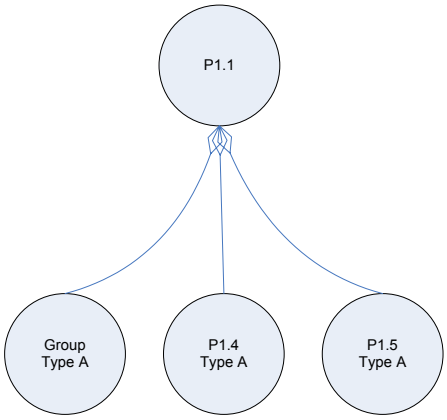

Figure 4.25: Grouping of processes.

the function could be obeserved at different levels. The behaviour of the mechanism is illustrated in Fig. 4.26, 4.27 and 4.28.

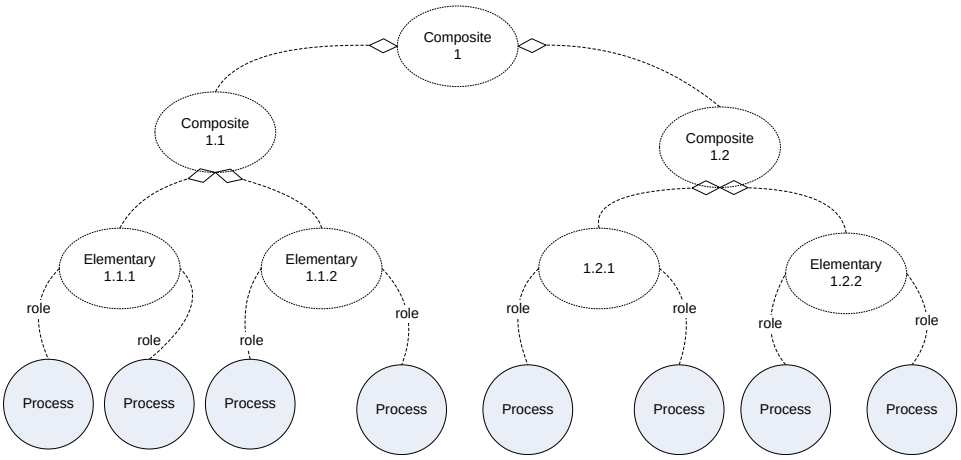

Figure 4.26: Illustration of the fully expanded collaboration tree.

Figure 4.26 shows a collaboration tree fully expanded. If coposite collaboration '1.1' and '1.2' were collapsed the result would be as shown in Fig. 4.27. The roles would now be moved from the elementary collaborations to '1.1' and '1.2' to indicate that the processes take part in a elementary collaboration that are contained in '1.1' and '1.2'.

By further collapsing composite collaboration '1', the function tree would be at the highest level of abstraction (see Fig. 4.28). Now all roles would point to composite collaboration '1' to indicate that the processes take part in an elementary collaboration contained in composite collaboration '1'.

It it not necessary to visualize the entire collaboration tree if it is fully or semi expanded. The parent collaborations in the tree, if not collapsed,

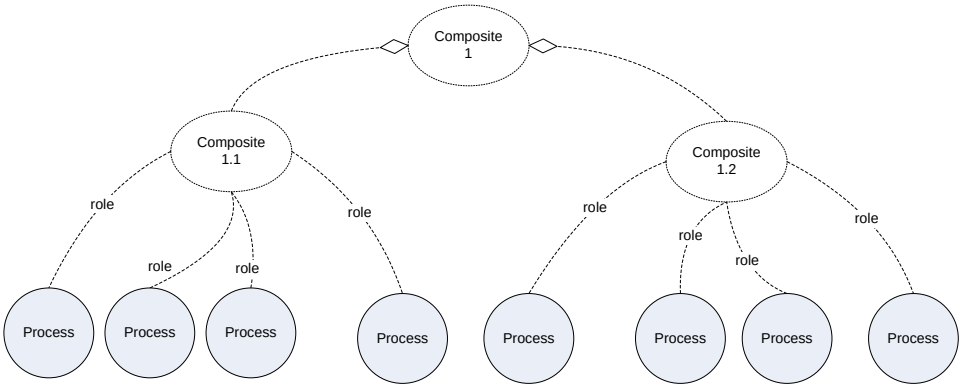

Figure 4.27: Illustration of the collaborations after a collapsing two composite collaborations.

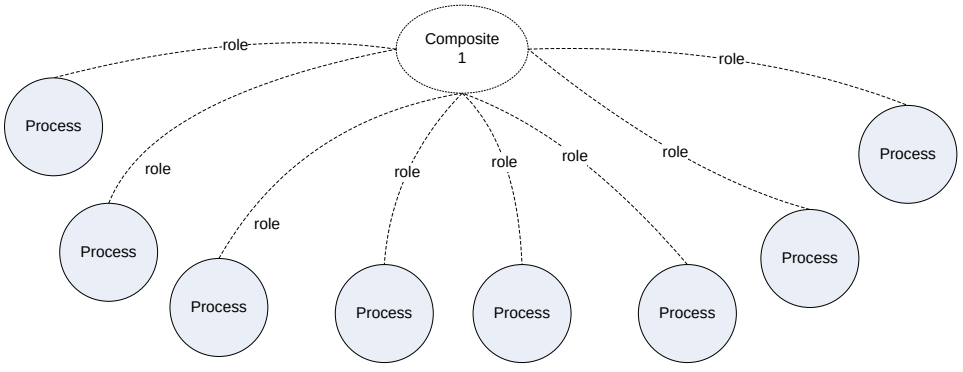

Figure 4.28: Illustration of the collaborations after a collapsing root composite collaborations.

could be omitted from the view. This way only the lowest level visible collaborations would be visualized in the view. This approach will be used in the implementation to further limit the information in the view.

Chapter 5

Testing

A test of the implemented functionality will be performed. A small sample application, that have been designed according to the constraint defined in chapter 4, will be used for testing.

5.1 The Sample Application

The test application consist of a small chat system. Clients can join or create chat rooms by contacting the a chatroom manager. When a client sends a message to the chat room, a composite collaboration will occure. The composite collaboration consist of two elementary collaborations. The composite collaboration is described below.

```
Composite collaboration: GroupChat
 |- elementary collaboration: ChatRoomChat
 |- elementary collaborations: BroadCastMsg
```

'ChatRoomChat' will exist between the 'ChatRoomSession' process and the 'ChatRoomSessionAgent' process in the 'ChatRoom' process. This collaboration will be used to demonstrate the collapsing of collaboration and how the process filtering and the collaboration filtering can be used together. For more details on the sample application, the reader is refferd to B.2.

5.2 Test Run

When the collaboration monitor is started, the first we must do is to load the data from the model of the observed system. A screenshot of the loaded data is shown in Fig. 5.1.

The next step is to start the sample application. The trace buffer will now receive data from the logging server and we can start adding trace events. Adding event is done by pushing the SFW button in the menu.

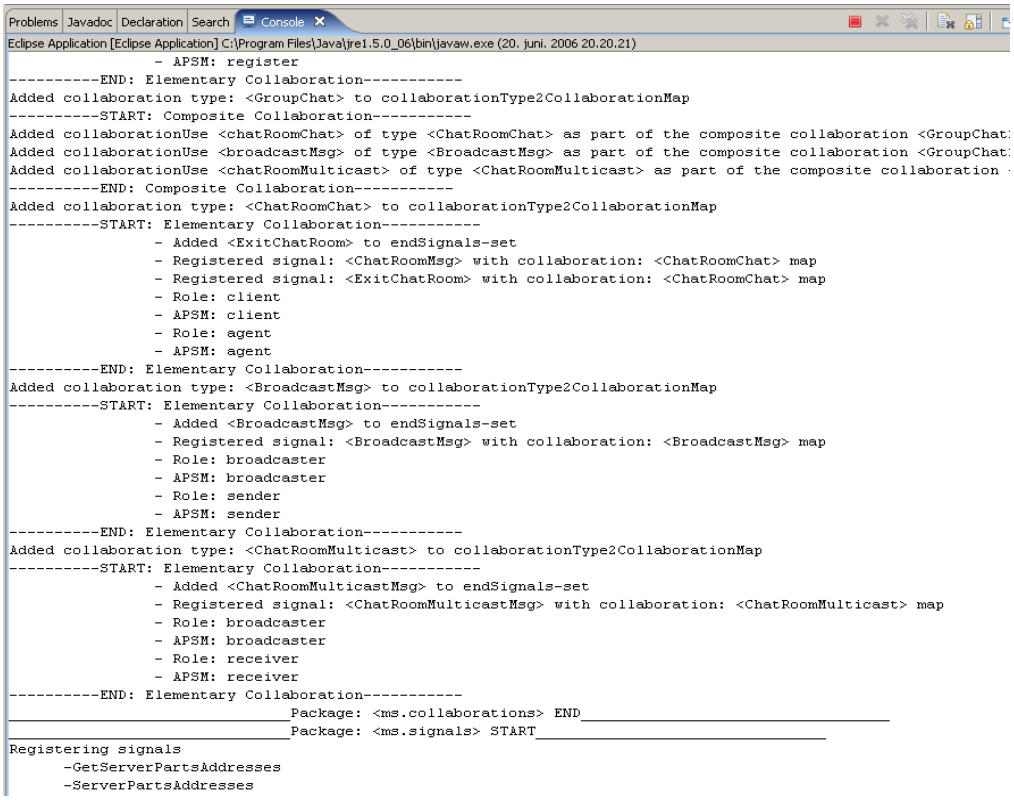

Figure 5.1: Loading of data is completed.

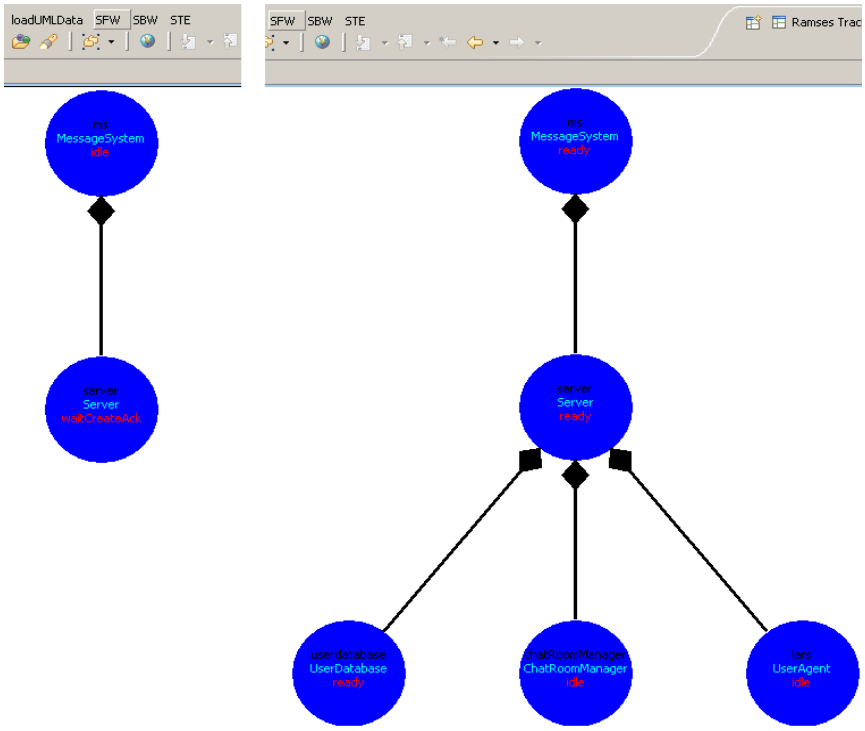

Figure 5.2: Using the stepping function.

After adding a couple of events the system looks like Fig. 5.3. As seen from the screenshot, the processes are layed out nicely. The implemented automatic layout works as intended. The figures also has their own tooltip.

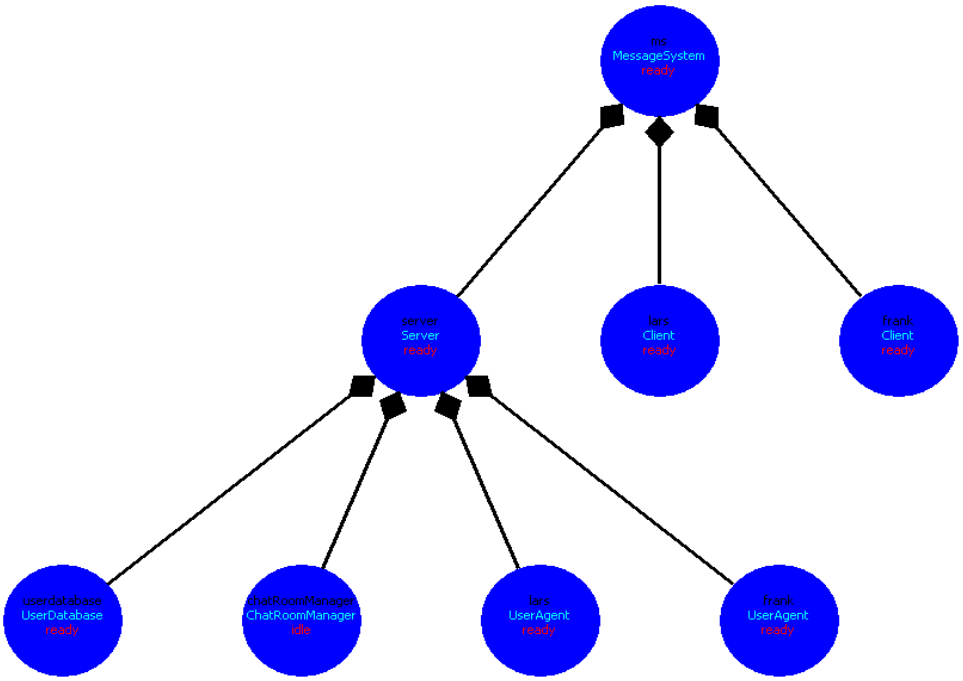

Figure 5.3: An example of the system hierarchy viusalized in the monitor.

These can contain additioal information about the figure, in addition to the information visible in the figure. The tooltip is helpfull if the view is zoomed out, and the information in the figures is hard to read.

5.2.1 Test of filtering

In this section the different implemented filtering mechanisms will be applied to the visualization.

Grouping

Figure 5.5 shows the process hierarchy before the grouping function is applied. The grouping mechanism will be applied to the user agents in the visualization.

After the grouping mechanism is applied, the user agent processes are represented by a single object in the view. This can be seen in Fig. **??**. The objects are viewed as one, and this is reflected in the collaborations they take part in.

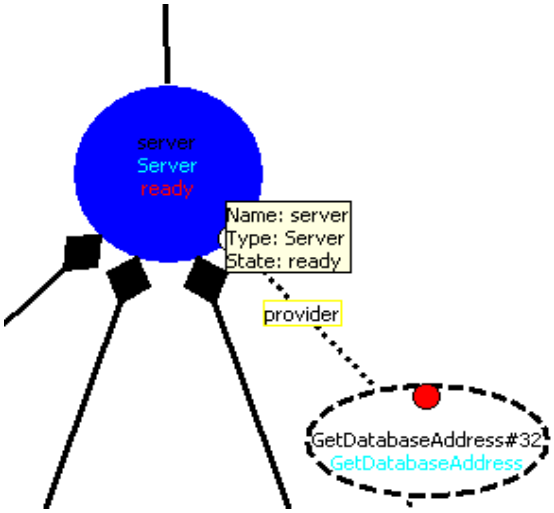

Figure 5.4: Additional information in tooltip figure.

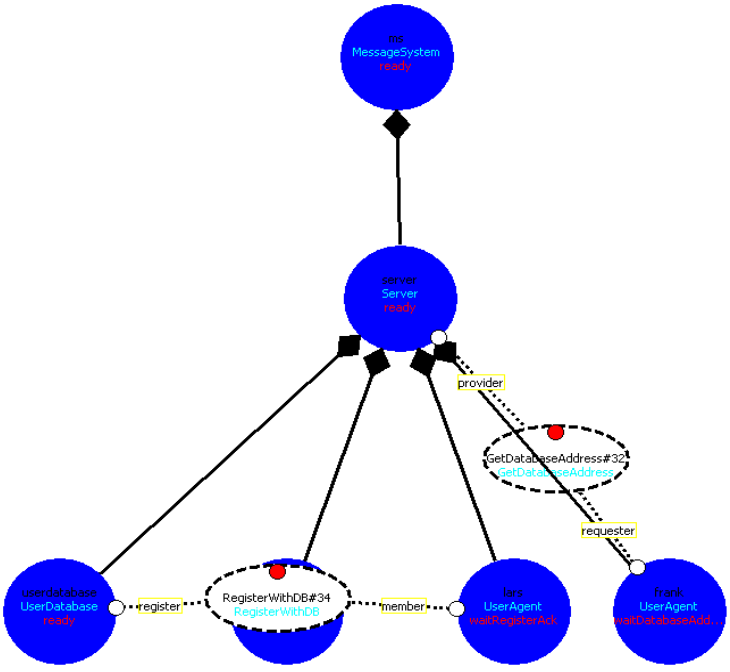

Figure 5.5: Before grouping.

Process filtering

With the process filtering a user can hide part of the system. In Fig. 5.6 the server process is collapsed. The result in shown in Fig. 5.7. The sub processes is hidden as well as the collaboration between the server and the user agent.

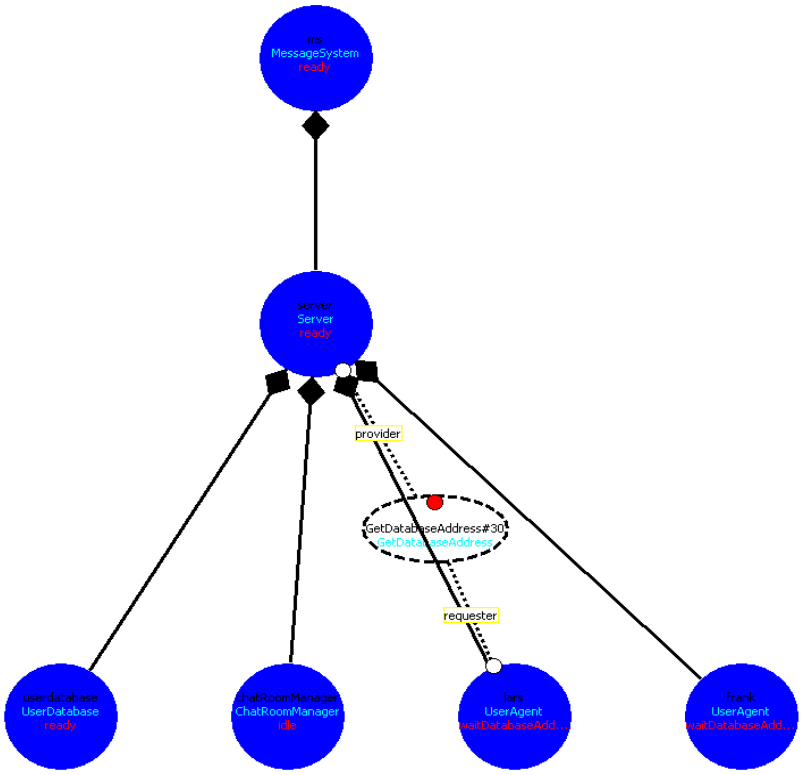

Figure 5.6: The systtem strcuture before collapsing.

Collaboration filtering

Fig. 5.8 shows a expanded composite collaboration. As explained in section 4.8.2, we do not show the composite collaboration if it is expanded, but only the collaborations it contains. When the composite collaboration that the two elementary collaboratios are a part of is collapsed, the visualization will look like Fig. 5.9. The role connections are now pointing to the composite collaboration, symbolizing that they take part in the contained elementary collaborations. The final screenshot in Fig. 5.10 shows both process and collaborations filtering applied to the system. If the 'ChatRoom' process in Fig. 5.9 is collapsed, the resulting visualization will look like the screen shot in Fig. 5.10.

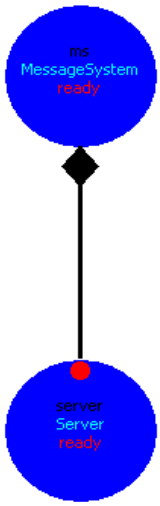

Figure 5.7: The system structure after collapsing.

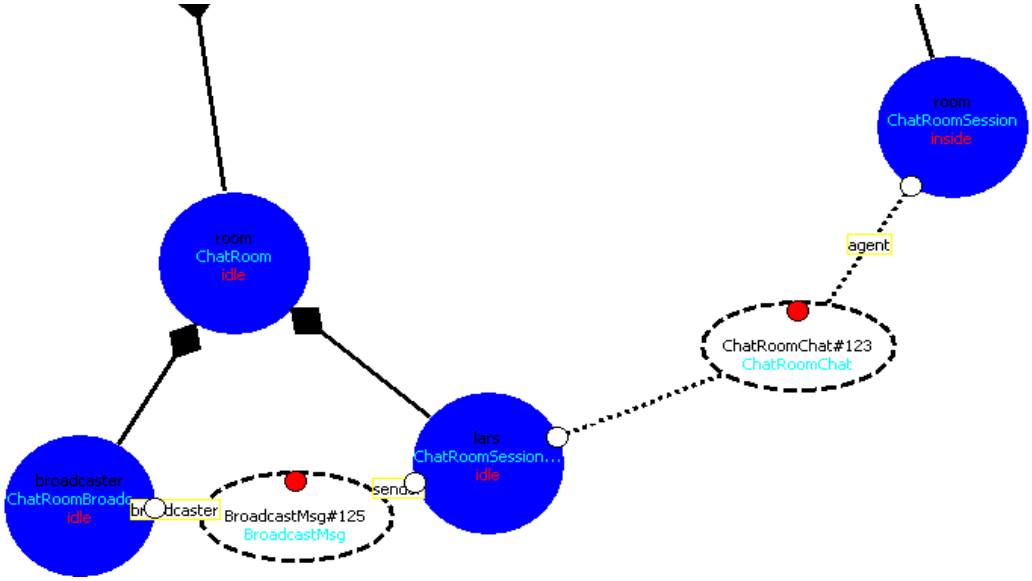

Figure 5.8: An expanded composite collaboration.

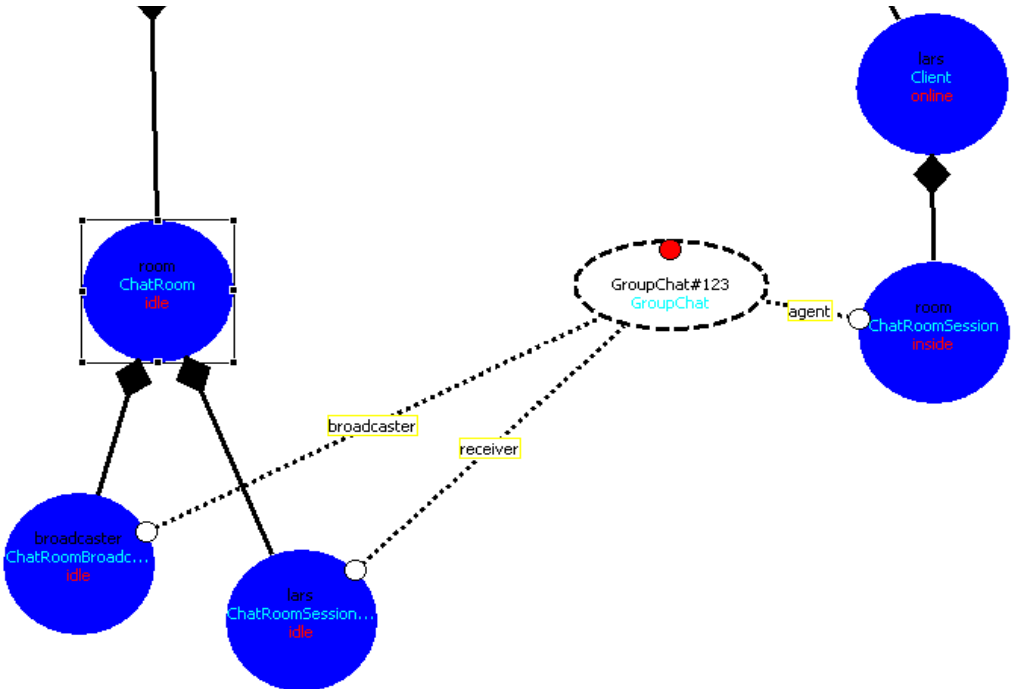

Figure 5.9: A collapsed composite collaboration.

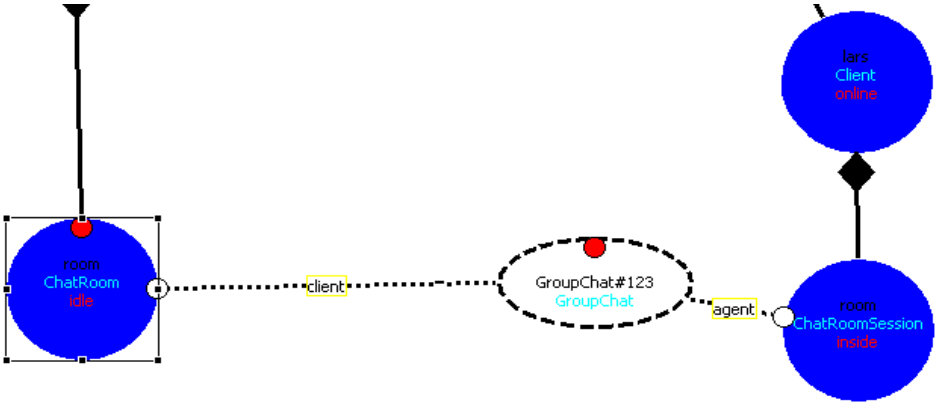

Figure 5.10: Use of both process and collaboration filter.

Chapter 6

Discussion

This chapter will offer a critical view on the work in this thesis. The discussion is divided in two parts. The first part will discuss and evaluate the mechanisms suggested in chapter 3 and implemented in chapter 4. The results of the test run in chapter 5 will be used as a basis for the evaluation. The next section will discuss the other solutions used in the realization of the monitor. Comparisons with alternative solutions will be performed and improvements will be suggested if needed.

6.1 Implemented functionality

The purpose of the mechanisms suggested in chapter 3 were to provide a visualiaztion that had a better scalability, overview and overall usability. To what extend these efforts have been successful will be presented.

6.1.1 Loading of data

As seen in Fig. 5.1, the loading of data worked as intended. The fully automatic approach was a good improvement compared to the manual adjustment needed in [Kar05]. It has increased the usability of the monitor considerably, as it is now possible to quickly change from the observation of one system to another, by simply loading another system model. The prototypical hard coded model specification in [Kar05] is not necessary anymore.

6.1.2 Event based control

The mechanism for the control of the visualization works as intended. It is possible to add a single event or several events in the buffer to the visualization. By stepping backwards you can reverse the changes of the recent added event. In this way, an execution can be studied in detail and repeatedly, while in [Kar05], the system could only be observed in the same speed as the actual execution of the program, which often was too fast. Moreover,

this approach prevents the inaccuracies indentified in 4.7.1, as the events are added to the visualization in the correct order. The sorting of events by Lamport stamps guarantees this.

6.1.3 Automatic layout of figures

The automatic layout of figures have been improved by using the directed graph layout algorithms provided by the draw2d package in GEF. The process tree is layed out in an orderly fashion, and the collaborations are positioned between the involved processes. However, objects in the screen might still be put on top of each other, i.e., a collaboration can be placed on top of a process. With the selection tool, a user can easily correct this. The number of necessary manual adjustments of the layout of figures has been reduced so the automatic layout has been improved. The layout algorithm in GEF has features like ranking of nodes and width and length calculation of sub-trees based on the containing nodes. This has not been used in the implemented prototype. Using these features could further improve the layout. Treating the system tree as several individual sub-trees, that can be indiviually organized may also provide better overview. The layout of figures is based on the structure of the process tree. There might be cases where one is less concerned with the structure of the system, and more concerned with the behaviour represented by the collaborations. In this case, a layout based on collaborations rather than the process tree might be usefull. By providing both collaboration focused layout, and process focused layout, a user could choose what should be the focus in the layout of the visualization. This additional layout option could improve the usability of the monitor.

6.1.4 Visualization of nested collaborations

From Fig. 5.8 and Fig. 5.9 it is clear, that nested collaborations are visualized correctly. With the algorithm and constraints defined in section 4.6.2, the visualization is possible. The nested collaborations can show the behaviour at different abstraction levels and can reduce the information visible to the user.

6.1.5 Filtering

Through the filtering mechanisms the visualization can be abstracted and customized so that the system behaviour is presented more clearly and is easier to understand.

System structure filtering

From the the test run (see Fig. 5.7) it can be seen that by collapsing and expanding the system structure we can effectivly reduce the amount of infor-

mation in the display. By collapsing nodes in the system tree, the observed system can be shown at different abstraction level by hiding internal processes. With the grouping mechanism (see Fig. **??**), it is possible to group processes of the same type at the same level in the system strcuture tree together, and observe them as a single object. Both mechanisms contribute to the reduction of information in the view and the scalability of the visualization.

Filtering of behaviour

The filtering of behaviour (see Fig. 5.9) makes it possible to observe the behaviour of the system at different levels of abstraction. By collapsing a nested collaboration the contained sub-structure are hidden. By expanding a nested collaboration the sub-structure can be viewed. The collapsing of nested collaborations reduce the information in the view and increases the scalability of the visualization.

Improving filtering

All the filtering mechanism in the monitor are in different ways locked to the structure of the system or the structure of the collaborations. For the system structure this is a weakness. It is not possible to observe a sub-system of the oberved system without also have to see every processes higher up in the hierarchy. An approach like the one used in the implementation of collaboration filtering, where only the lowest level visible node were visible, could be added. In addition, filtering mechanism that are independent of system structure should be considered. With the size of the figures used, the monitor can visualize, depending of the structure of the system, about 20 processes in a single display. If we reduced the size of the figures, it would be room for more figures, but the information contained in the figures would be barely readable. There are also limits to how many objects a user can observe in a display before the visualization become too big and complex. The implemented filtering mechanisms do provide improved scalability, but when large systems are observed they might not be able to reduce the amount of information sufficiently . The resulting visualization could be too big for an ordinary computer display. Alternative solutions must be considered in order to provide the necessary overview. This is discussed in section 6.2.7.

6.2 Monitor Design

This section will discuss the remaining parts of the monitor design and implementation not covered in section 6.1. The implemented solutions and the use of the existing frameworks will be discussed and alternatives or improvements suggested where needed.

6.2.1 Generality of the monitor

The different parts of the system architecture and the sample application used for testing is all written i Java. Since Java is a plattform independent language, the monitor should be able to run on most plattforms. The monitor is not limited to only visualize applications written in Java. I.e., if a program is created in Ramses, a different code generation tool for a different programming language or plattform could be used. A program has to meet the following requirements in order to be visualized:

- **Trace event reporting -** The system must be able to report trace events to the logging server in the trace format defined in [Nes05] (see appendix A.1). Lamport stamping of events must be supported in order to preserve the causal ordering of the events.

- **System description -** The system must offer a description of the system in a model in aggrement with the PAX Ramses III UML model. The model restrictions defined in section 4.6 must be followed.

6.2.2 Content of the Visualization

The visualization shows the processes and the collaborations in the observed system. Since collaborations consist of signals, only the behaviour of the system that is directly a part of the messaging between the different parts of a system. Internal processing in a part (i.e., algorithms and updating local variables) is not visualized. Some instructions in the program code and data structures are omitted in the visualization. Control and data flow should be visible in the visualization because they tend to follow the messages in the system, which we can visualize.

6.2.3 Invasive approach to tracing

The trace support mechanism used in the monitor is invasive. It requires modification of the program source code in order to be able to collect trace information. In a concurrent system, the additional code required in the observed program to support tracing, may affect the execution rates of processes which could produce a different result [PBS98]. There are alternatives to the invasive approach, but most systems that handles concurrency are invasive [PBS98]. It is also the usual approach to adding trace support in distributed systems [TSS95]. In [FC90] a bus monitor was used to avoid any invasive approach to tracing. In [Meh02] the JDI was used to avoid an invasive approach of tracing. Both the existing Ramses tool suite, the sample application and the monitor code is written in Java. In consequence, the monitor could have been using the JDI for trace collection. Both of the alternatives would however make the monitor plattform dependent. The

Ramses tool suite that the monitor is a part of, is suppose to be able generate code for for different plattforms. If the JDI had been used for trace collection, only program in Java code could have been visualized. By using the existing trace mechanism implemented in [Nes05], the monitor will remained platform independent.

To what degree the invasive approach affects the execution of the observed program is uncertain. The observations of out-of-order events received by the monitor from the logging server (see section 4.7.1) might suggests that the tracing could have an affect. The observation may also be caused by the behaviour of the the logging server. As this was not the main focus of this thesis, this should be studiet further elsewhere.

6.2.4 Real-time vs. Post-mortem visualization

The time of visualization affects how the user can interact with the visualization. Both real-time and post-mortem approaches has it's advantages and disadavantages. A post-mortem approach has the advantage of rich information. The information known pre-run-time combined with the information gathered durring run-rime gives a complete picture of the system. With all information known in advance of the visualization, it is easier to present the information in an optimal way. A real-time approach has the advantage that a user can interact with the visualization based on program output, thus have an immediate effect on the visualzation [PBS98]. In a post-mortem approach, this is not possible. A real-time approach provides an up to the moment view of the computation's progress and can reduce the overhead of of data storage [Kra98]. The approach taken in this thesis is described as semi-post-mortem (see section 4.7.2). The user is not restricted to wait to the very end of execution, but can add events as they happen. The risk of adding out-of-order events (see section 4.7.1) is possible, but has not been experienced durring testing. The test application was however controlled by the user, and events were added when the observed system was idle. In larger real-life applications where behaviour is less controllable, and the activity is high, the risk of adding out-of-order events is much higher. In consequence, if the system can not be controlled, the events should be added post-mortem. As a result, the user can not affect the visualization the same way as in a real-time approach. A post-mortem approach provides the opportunity for a more detailed display that can proceed at a user-specified pace, and will ususally interfear less with a program than a real-time approach [Kra98]. A real-time visualization can not be too detailed, as the viewer would fine it difficult to comprehend a rapidly updating, highly detailed display [Kra98]. Performance is also an issue if a real-time approach is taken [BS98]. The computational cost of layout and filtering of information in real-time might be to high if a large system is observed. In consequence, the visualization would not show the behaviour in real-time. This is avoided

by using post-mortem visualization. A disadvantage of the post-mortem approach is the amount of information that need to be maintained. In post-mortem all data all data must be saved which lead to a lot of overhead when observing larger system [Kra98]. How this may affect the monitor is discussed in section 6.2.7.

6.2.5 Presentation style

The graphical elements used to represent processes, collaborations and connections are very simple. They should be easy to understand for all types of users of the monitor. By using our own defined figures, we can easily change or add additional graphical representations. The existing figures can be expanded, new figures can be designed or pictures can be used in stead. The figures convey information through their position in the visualization (i.e., a process is visualized as a child) and through the textual information they contain (i.e., state information). Colour is used to indicate wheter or not an object in the screen in collapsed, and it separates the different elements from each other together with the shape of the elements. Colour could probably have been utilized better in the visualization. For instance, the monitor could indicate activity in a process or collaboration with a flash of a certain colour. The use of sound could also be considered.

The monitor shows the visualization frame by frame. There is no continous animation when objects in the view are changed or moved. Such animations have proven effective in other SV tools [Sta98] and there are existing frameworks that demonstrates how this can be done [Sta88]. Such a feature could be added without doing any re-design of the current monitor, because if should only affect the figures in the visualization.

6.2.6 Interacting with the monitor

The user can interact with the visualization through menu buttons, clicking on objects in the view (processes and collaborations) and by using the selection tool with the mouse. The different ways of interacting with the monitor should not be affected by larger programs. They should scale well to any type of observed program.

6.2.7 Scalability

Monolithic view approach

The monitor design uses the monolithic view approach. The advantage of such a view is that it compacts all information into a single view, and the user only has to focus on that view. The experience of using a single view in this thesis (see section 6.1.5) and in other SV tools [BH98], indicates that presenting all information in a single view is good for smaller programs,

but can be difficult for larger programs. Additional filtering mechanism could be added to improve the visualization, but adding additional filtering mechanisms is not trivial, as the mechanism has to be intergrated with the existing filters. If a single view is used, all filters work on the same view-model representing the view. The additional filters would probably need to add additional state information to the view-model. Adding filters is an increasingly complex task. The value of these filters would probably be limited in the context of scalability. A new approach to solving the issue of scalability should be taken. Instead of restricting the visualization to a single view, additional views should be added. This alternative should be better and easier to implement than adding additional filters. For example, a sub-system view and collaboration view could be added. The sub-system view would only contain a selected sub-system in the system structure. The collaboration view could show a collaboration and the participants. This approach is illustrated in Fig. 6.1. The existing view could show the entire system. If visualization in the main view becomes too big for the display, the user could use the other views to inspect certain parts of the system. With the structure of the information models in the monitor, this could be achieved by adding new view models on top of the data model (see section 4.1.1). The figures and probably the editparts could be used as is.

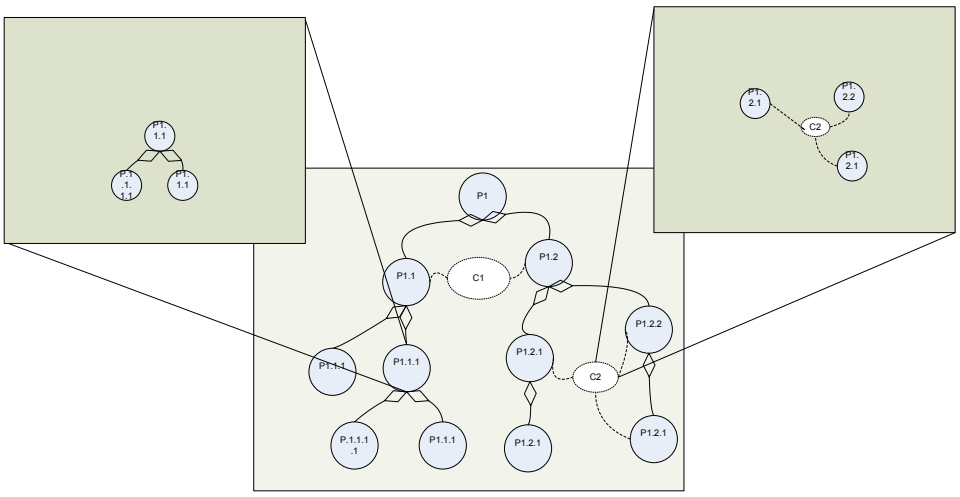

Figure 6.1: Illustration of the multi-view approach.

Performance Issues

With the sample application we can test the functionality of the prototype and fairly confidently say whether or not they add value. The sample appli-

cation however, is not suited for an performance evaluation of the prototype. For this purpose it is too small in size and too simple in behaviour. The behaviour of the test system is also highly controllable, does not have any random behaviour and the activity level is low compared to real-world examples. Despite the lack of large test systems, some of the current solutions would probably not scale a very large active systems.

Buffering of trace events In it's current form the there are no enforced limit on the number of events that will be buffered in the monitor. The monitor will stop when the Java virtual machine runs out of memory. In order to support the event stepping function during the entire system run, all events must be buffered. Observing large systems with a high level of activity would generate a lot of overhead in the form of trace events. Maintaining all events in the memory during observation might not be possible. In order to support larger and more activ systems, the buffering mechanism may have to be re-designed. Two approaches can be used. The trace events can be written to disk, and the system could be observed post-mortem by loading the trace data from a log. An alterative solutions is to limit the number of event that can be buffered, forcing them to be added to the visualization, and not provide unlimited support in stepping backwards in the visualization.

Number of graphical objects The monitor creates and maintains all elements in the viusalization that is not terminated. If certain elements are filtered out of the visualization through one of the provided mechanisms, they still occupy memory. They are not removed from the visualization, they are simply not visible. This might cause problems if very large systems are observed.

6.2.8 Constraints on the model

The constraints defined in section 4.6.2 might be too strict. Not allowing to reuse signals in different elementary collaborations or the reuse of elementary collaborations in different composite collaborations violates the the UML specification in [Gro04]. The core of the problem is to identify collaborations uses from the available trace data. Alternative solutions might be possible if the trace data could contain more information that could help identfy a collaboration. Manipulation of the existing plattform was not an option (see section 4.2), so this approach was not studied. This alternative apprioach should be studied further elsewhere, in an attempt to ease up on the restrictions defined in this thesis.

Chapter 7

Conclusions

This chapter summarizes the achievements in this thesis. Future works regarding the monitor for visualization of collaborations are proposed.

7.1 Achievements

In the context of visualizing distributed, reaktive systems, several achievements have been made. The main achievements are listed below.

- **Automatic loading of model data -** An automatic approach to loading necessary data from the model of an observed system has been implemented. The implemented solution worked as intended, and removed the need for manual adjustment of data.
- **Post mortem visualization -** A buffered approach to visualization was taken in order to control the speed of the visualization and to correct inaccurracies in the visualization. Trace event were buffered and sorted by Lamport stamp in order to produce a correct visualization. The mechanism gave the user complete control of the speed of the visualization, and the possibility to reverse the visualization.
- **Visualization of system behaviour at different level of abstraction -** Necessary constrains on the model of a system were defined and an algorithm for the detection of nested collaborations in a running system was suggested and implemented. Testing with an example application showed that the collaborations were visualized correct. The visualization could show the behaviour of the observed system at different level of abstraction through the visualized nested collaborations.
- **Improved scalability -** Mechanisms to improve the scalability of the visualization were suggested and implemented. Through testing they showed that they added addtional scalability to the visualization tool.

We argued that in order to further improve scalability, more views should be added.

7.2 Future works

From the discussion of the work it is evident that some areas require further work in order to improve the visualization tool. The following is proposed:

- **Alternative solutions for collaboration detetction -** The solutions proposed in this thesis to the detection of collaborations in a running program forces strict contraints on the design of the system. An investigation of alternative ways with less strict design constraints i.e., adding additional trace information in order to detect collaborations, could be investigated.
- **Additional views -** The posibility of adding additional views to the monitor, as suggested in section 6.2.7, in order to improve scalability and overview in the visualization could be investigated.
- **Performance evaluation -** The monitor needs to be tested with larger real-life applications in order to get an accurate estimation on how well the implemented solutions scale to observation of larger systems.
- **Empirical Evaluation -** To get a better understaning of how well the monitor explains observed system behaviour, an empirical evaluation needs to be performed. The evalutaion should use real-life application in order to be accurate.

Bibliography

[BH98] Marc H. Brown and John Hershberger. *Fundamental Techniques for Algorithm Animation Displays*, chapter 7. The MIT Press, 1998.

[BS98] Marc H. Brown and Robert Sedgewick. *User Interface Issues For Algorithm Animation*, chapter 11. The MIT Press, 1998.

[DFK+05] Jim D´Anjou, Scott Fairbrother, Dan Kehn, John Kellerman, and Pat McCarthy. *The Java Developer's Guide to Eclipse*, chapter 7. Addison-Wesley, 2005.

[FC90] S. Flinn and W. Cowan. Visualizing the execution of multiprocessor real-time programs. In *Proceedings on Graphics interface '90*, pages 293–300, Toronto, Ont., Canada, Canada, 1990. Canadian Information Processing Society.

[Fou06] Apache Software Foundation. Logging services. *http://logging.apache.org/*, 2006.

[Gro04] Object Management Group. *Unified Modeling Language: Superstructure, version 2.0.* OMG, 2004.

[Gro06a] EMF Group. Eclispe modeling framework. *http://www.eclipse.org/emf/*, 2006.

[Gro06b] GEF Group. Graphical editing framework. *http://www.eclipse.org/gef/*, 2006.

[Hud03] Randy Hudson. How to get started with gef. *http://www-128.ibm.com/developerworks/opensource/library/os-gef/*, 2003.

[Kar05] Lars Erik Karlsen. Collaboration-Oriented Visualization of Communicating State Machines. Project Thesis, 2005.

[Kra98] Eileen Kraemer. *Visualizing Concurrent Programs*, chapter 17. The MIT Press, 1998.

[Kra06] Frank Alexander Kraemer. *Pax Ramses – Constraints on UML 2.0 Models for the Use with Ramses (Internal Note)*. Department of Telematics, NTNU, Trondheim, Norway, March 2006.

[Lam78] Leslie Lamport. Time, clocks and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558–565, July 1978.

[Meh02] Katharina Mehner. JaVis: A UML-Based Visualization and Debugging Environment for Concurrent Java programs. *Software Visualization*, 2002.

[MH05] Geir Melby and Knut Eilif Husa. *ActorFrame Developers Guide*. Ericsson NorARC, Asker, Norway, September 2005.

[Mic06] Sun Microsystems. Java Platform Debugger Architecture (JPDA). *http://java.sun.com/products/jpda/*, 2006.

[Nes05] Ronnie Nessa. Trace Visualisation for distributed State Machines. Master's thesis, NTNU, 2005.

[Nor98] Stephen North. *Visualizing Graph Models of Software*, chapter 5. The MIT Press, 1998.

[PBS98] Blaine Price, Ronald Baecker, and Ian Small. *An Introduction to Software Visualization*, chapter 1. The MIT Press, 1998.

[Ree79] Trygve Reenskaug. Thing-model-view-editor an example from planningsystem. 1979.

[RJB04] James Rumbaugh, Ivar Jacobsen, and Grady Booch. *The Unified Modeling Language Reference Manual Second Edition*. Addison-Wesley, 2004.

[Sta88] John Stasko. The TANGO Algorithm Animation System. Technical report, Providence, RI, USA, 1988.

[Sta98] John Stasko. *Smooth, Continuous Animation for Portraying Algorithms and Processes*, chapter 8. The MIT Press, 1998.

[TSS95] B. Topol, J. T. Stasko, and V. Sunderam. Integrating visualization support into distributed computing systems. In *ICDCS '95: Proceedings of the 15th International Conference on Distributed Computing Systems*, page 19, Washington, DC, USA, 1995. IEEE Computer Society.

[Vli00] Hans Van Vliet. *Software Engineering - Principles and Practice*. Wiley, 2 edition, 2000.

Appendix A

Specifications

A.1 Trace Format

This table shows the trace format of the events received by the monitor from the logging server. This table is taken from [Nes05], pages 57 to 59.

Tag name	**Part of**	**Occurence**	**Legal values**	**fields**	**Description**
<transition>	-	1			The main tag that describes a transition
<actorid>	<transition>	1	text		The id of the actor which executed the transition
<currentstate>	<transition>	1	text		The state of the process before it executed the transition
<nextstate>	<transition>	1	text		The state the process entered after executing the transition
<receive>	<transition>	0..1		lamport-Stamp	Indicates that a receive event has occurred
<send>	<transition>	0..*		lamport-Stamp	Indicates that a send event has occurred

<startTimer>	<transition>	0..1		lamport-Stamp	Indicates that a start timer event has occurred
<stopTimer>	<transition>	0..1		lamport-Stamp	Indicates that a stop timer event has occurred
<timerExpired>	<transition>	0..1		lamport-Stamp	Indicates that a timer expired event has occurred
<warning>	<transition>	0..*	text		Warning generated during the transition
<message>	<send> or <receive>	1			The message being sent or received
<sender>	<message>	1	text		The sender address contained in the message.
<receiver>	<message>	1	text		The receiver address contained in the message.
<frameworkMsg>	<message>	1	true | false		Wether or not the message is a framework message. Not used at the moment
<content>	<message>	1	text		The content of the message
<create>	<transition>	0..1			Indicates that a new process has been created by this process.
<created>	<create>	1	text		The name of the process that was created.
<terminate>	<transition>	0.1			This process has terminated.

Table A.1: XML elements in trace object accepted by the Log Server

Appendix B

Source Code

B.1 Source Code for Collaboration Monitor

The source code for the collaboration monitor can be obtained by accessing the Ramses CVS server.

B.2 Source Code for Sample Application

The source code for the collaboration monitor can be obtained by accessing the Ramses CVS server.