Master thesis 551:

Scenario-based scheduling for real-time systems

Author

Bart Kerkhoff

Supervisors

Jozef Hooman	Radboud University Nijmegen	Computer science
Pepijn Vos	Radboud University Nijmegen	Management and Technology
Johan van de Hee	Océ-Technologies	
Ton Janssen	Océ-Technologies	

Radboud Universiteit Nijmegen

Abstract

In this research we assess the practical usefulness of scenario-based scheduling. This scheduling technique provides an alternative way of handling the scheduling of automatically generated programs from UML-RT models. The technique is relatively new and has not been used in practice yet. It performs well on small test models, however it is unknown whether it works for large projects. We address two issues in this research; can scenario-based scheduling handle large projects and if so, does it actually increase the performance of those projects?

At the start of the research, there was only an experimental implementation of scenario-based scheduling. This research starts with an analysis of that implementation. Since it appeared not to be suited for large projects, it had to be modified and upgraded. Lots of bugs needed to be removed and the functionality and flexibility had to be increased. Next, we have applied this new implementation of scenario-based scheduling to a large and complex project, Océ's VarioPrint 6250. A model was developed to analyze the impact of scenario-based scheduling on the real-time software development process and its resulting end product. According to this model, the application of scenario-based scheduling to the VarioPrint 6250 has been evaluated.

This research has shown that scenario-based scheduling can handle large and complex projects. For the test case with the VarioPrint 6250 it appeared that the scheduling technique was easier to apply than the conventional approach. Hence, it improves the development process. However, the product itself actually performed worse; response time increased and there was quite at lot of overhead. Overall, scenario-based scheduling was shown not to be useful for that project. We do expect better results for projects that have less computation power available. The results might also differ for projects with different kinds of scenarios. It is therefore recommended to further investigate the effects of scenario-based scheduling on different projects and scenarios.

Contents

1 Introduction

This report describes the research project into *scenario-based scheduling for real-time systems.* Scenario-based scheduling provides an alternative way of handling the scheduling of automatically generated code from UML-RT models. The theory sounds promising and claims to yield good results, however it currently has not been tested in practice. That is what this research is about. First, we will determine whether or not the technique is actually suited for large and complex projects. And second, we will analyze the effects of scenario-based scheduling on the real-time software development process and the produced product. In short, the aim is to answer the question whether scenario-based scheduling is useful in practice.

This research project will be the master thesis of Bart Kerkhoff, studying at the Radboud University Nijmegen. Bart studies computer science, with the graduation theme 'Embedded Systems' and the track 'Management and Technology'. The project is an assignment of, and will be done at, Océ Venlo.

For Océ, and other companies that generate real-time code from UML-RT models, this research is interesting because it will provide them with a new, and maybe better, way of scheduling their automatically generated code. With regard to the university and the academic world, this research is of interest because it further explores the new area of scenario-based scheduling and because it will provide a framework to assess real-time software and its development process.

We will start off with explaining the details of this research in chapter 2; what we are going to do. After that, in chapter 3, we will provide some background information about the scheduling of generated code from UML-RT models. How is this code scheduled by default and what different scheduling techniques are available? In chapter 4, the implementation of scenario-based scheduling will be considered. What is implemented prior to this research project and what still needs to be done in order to make it work for large and complex projects? Next, chapter 5 provides us with a model that can be used to assess the quality of the scheduling techniques used in this research. How do we determine if the resulting product is 'better'? And what are the consequences for the development process and has it improved? In chapter 6 we apply our adapted scenario-based scheduling implementation to a project that satisfies the demands of being large and complex. We introduce the project, describe the scenario on which we have applied the technique and our experiences in doing so. Chapter 7 assesses that project according to the model we have developed. Did scenario-based scheduling really improve the product and did it make the development process easier? Based on our experiences and the impact that was measured, we will advice Océ in chapter 8 about the use of scenario-based scheduling. Should they use the technique, for which projects will it be suited and what still needs to be done? Finally, chapter 9 provides the conclusions of this research project.

2 Research description

In this chapter the details of the research into s*cenario-based scheduling for real-time systems* will be discussed. We start with describing the objectives of this research. After that, in section 2.2, the research questions will be posed that need to be answered in order to achieve the objectives. In section 2.3 we will discuss why this research is relevant for Océ. The relevance with regard to the university will be discussed in 2.4.

2.1 Objectives

This research consists of two phases. First, we are going to integrate scenario-based scheduling, a scheduling technique developed in Korea [15, 19], in the software development process at Océ. In the second phase we will evaluate whether or not this is an improvement to the current real-time software development process and its produced product.
The first phase focuses on the technical aspects. We need to adopt and upgrade the existing implementation to fit within the software development process at Océ. The deliverables of this phase, are the *adopted tool*, an *example project, accompanying documentation* and a *report of the integration*. The example project will be a previous Océ product that has been re-created using the newly developed tools. That should show the technical feasibility.
Phase two will evaluate the new scheduling technique. The original software development process and its product will be compared to the new ones that are using the scheduling technique. For this comparison, we will derive an evaluation model based on literature about software development and real-time systems. This model will be made specific for the situation at Océ. Based on that model, a comparison of both the software development processes and the products will be made. Based on this comparison, we will give an advice for Océ whether or not they should use the tool developed in the first phase of this research.
In the second phase there are the following deliverables: a model that includes the important aspects of the software development process and its produced product, to what extend the original and the new situation meet these demands and an advice about whether or not to use the new tool.

2.2 Research questions

In order to reach the objectives stated above, two sets of questions have to be answered. The first one with regard to the technical feasibility and the second one with regard to the usefulness.

- Is it technically feasible to integrate the new scheduling technique into Océ's software development process?
  - ❑ What are the limitations/restrictions of the current implementation?
  - ❑ To what extent is the technique scalable? Can it be used for large projects?
  - ❑ Which modifications can be made to the current implementation to reduce the limitations/restrictions?

- To which extend does scenario-based scheduling improves the real-time software development process and its produced end product?
  - Which criteria should be used to evaluate the software development process and the resulting real-time product; how can we determine the quality of a real-time system?
  - To what extent does the original situation meets these criteria?
  - To what extent does the new situation meets these criteria?
  - What are the differences between both situations with regard to the criteria?

2.3 Relevance for Océ

This research will be conducted at Océ Venlo. In this section we give a general description of Océ, its software development process, and the relevance of predictable scheduling for Océ.

2.3.1 Océ

Let's starting by looking at Oce's mission statement:

"Océ enables its customers to manage their documents efficiently and effectively by offering print and document management products and services for professional environments"

Océ is involved in the document business. Not only printing, copying and scanning, but also the services surrounding document management like the document workflow. They are aiming at the high-end professional market. They offer a wide range of high-quality products; printers, scanners, copiers, color/no-color, wide format, continuos-feed, cut-sheet, and so on.
Océ is a large multi-national. They have direct sales and service organizations in 30 countries and are active in approximately 80 countries. Globally, they employ 24.000 people. There are lots of investments in research and development. More than 6 percent of the turnover is invested in innovation. Furthermore, 2.000 of their employees are involved in research and development, which is over 8 percent. In 2005, Océ's had a turnover of € 2.677.300.000,- and a net profit of € 78.800.000,- [21, 20].
As can be seen, research and development is important to Océ. They are continuously developing new, high-end products. The printers, copiers and scanners Océ produces, consist of two things: the actual hardware and the software that controls them. This research focuses on the embedded software.

2.3.2 The software development process

For the design of their embedded software, Océ uses UML-RT [14]. To create these UML-RT models and its diagrams, Océ uses 'IBM's Rational Rose RealTime' [4]. While creating these models, Océ already has to do some initial reasoning about the scheduling parameters. Next, Rose RealTime can automatically generate the code, the actual software program, from the UML-RT model. This leads to a working piece of software. However, it is difficult to control the scheduling of this code; controlling in which order messages should be processed. Components of the system that are involved in two tasks of different importance can, for example, not be executed on two different priorities. One of the tasks is therefore handled at the wrong priority. Also the response times of the system are not optimal.

Lacking control over the order in which messages are being processed makes the system unpredictable, it is unclear when the systems starts processing a message. The initial scheduling parameters therefore rarely suffice. To make the software more predictable, Océ has to take additional steps. The system has to be closely analyzed, insight is required in which parts of the system are active, what messages are being send by these active parts, how much time is required to process these message, etc… A lot of complex work with a high risk of errors. Using this information the system can be tuned, capsules can be mapped to different threads and threads can be assigned different priorities. After that, the real-time requirements will be met most of the time, however there is still no proof. Extensive testing is used after that, to detect whether or not there are still flaws. If Océ is still not convinced, they simply use more powerful processor. But this still does not offer hard guarantees.
Figure 1 graphically depicts the current software development process. It shows that first an initial UML-RT model has to be developed with initial scheduling parameters. Next, code can be generated that still needs to be tested for its real-time properties. Someone needs to devise and run these tests. To see whether or not an error can occur, these test should often run for quite a while. When an error occurs, the model and the error need to be analyzed; what was the exact cause of the error and how can the model be changed to prevent this error. Doing this analysis requires extensive user interaction. This test/analyze/fix loop normally has to be repeated several times.

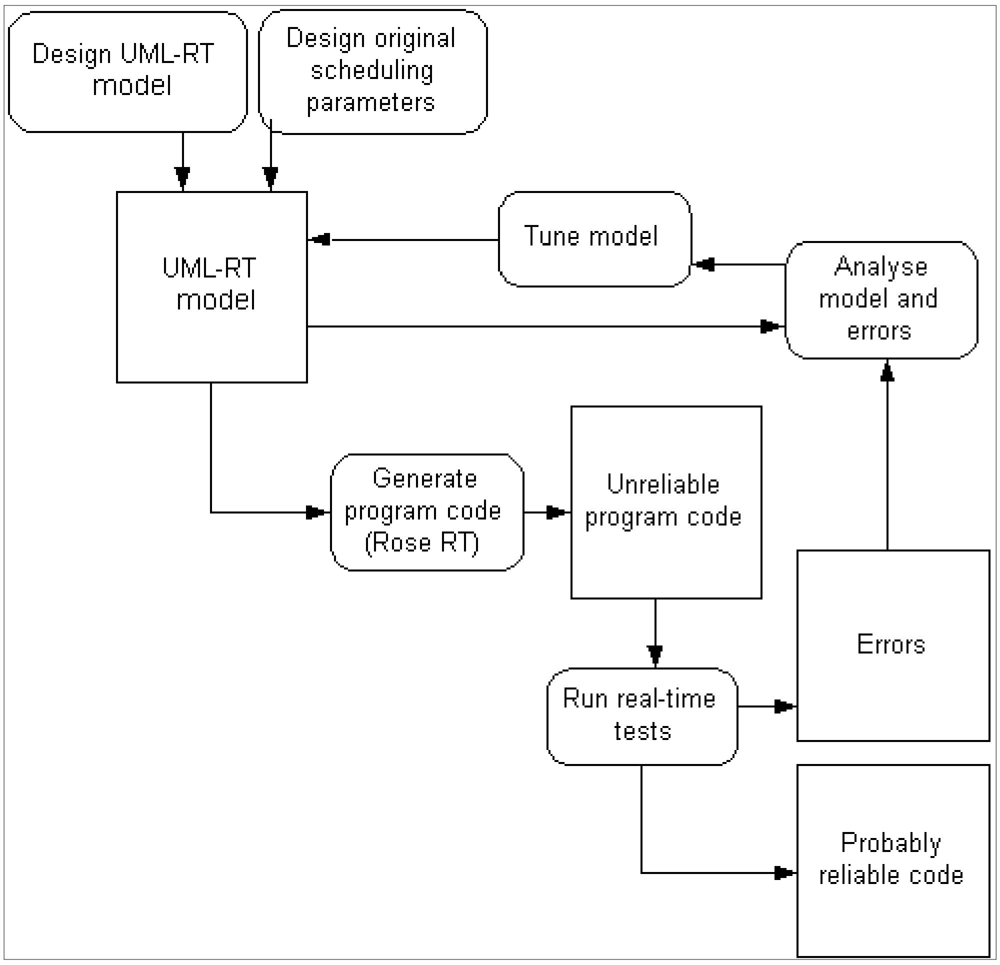


Figure 1: Original software development process

2.3.3 Océ and predictable scenario-based scheduling

Océ is looking for a way to improve the software development process described in section 2.3.2. At the end, they want to have predictable and reliable software. Furthermore, the process is not optimal; the test/fix loop is time and resource consuming.
Scenario-based-scheduling can potentially provide the desired improvements. The first research question will address whether or not using scenario-based-scheduling can be used in Océ's development process. Research question two answers whether scenario-based scheduling results in more predictable and better software, actually improves the development process, does not create too much overhead and thus whether or not scenario-based-scheduling is useful for Océ.

2.4 Relevance for the university

This section describes the relevance of this research and thesis towards the *Radboud University*. We will mention the contributions to academic research.

There has already been done some research into improving and predicting the scheduling of automatically generated programs from UML-RT diagrams. Attempts have been made to apply standard scheduling techniques [11], guidelines have been constructed [28, 29, 36] and some are venturing in the direction of scenario-based scheduling [13, 14, 15, 19]. Despite all this research, there is still no complete solution to the problem, they all have their specific disadvantages. Therefore this research subject is still open. The scenario-based scheduling attempts are relatively new. Currently, it is solely based on the research of Kim [13, 14, 15, 19]. We are not aware of any publications that either support or disprove the claims made. By putting this scheduling technique into practice, we will show if it is suitable for large-scale projects. Furthermore possible limitations and errors in the theory shall be identified and if possible, be solved. These things are addressed by the first research question and its sub-questions.

Next, there exists literature generally describing the software development process and also literature describing the important aspects of real-time systems. However we need to assess the complete picture, we need a model to assess the real-time software development process and the resulting real-time system. That model has to be composed. It will be a composition of theory about software development processes and theories about the assessment of the end product, the real-time system. Furthermore it has to encompass the specific demands at Océ. Developing this model will provide an answer to the first sub-question of the second research question.

3 Scheduling generated code

Having a predictable schedule can be important for ensuring that real-time requirements are met. In the literature, several options for achieving predictable scheduling are available. These can be divided by the level at which they are applied. At the lowest level there are the basic scheduling techniques that are being applied by the operating system. Those are not our prime interest, but they do provide useful background information. In appendix A, the most important basic scheduling techniques, their limitations and their guarantees are discussed.
Next, we go to a higher level. What can be done at the program level to achieve predictable scheduling? There are roughly two directions here. The first one, discussed in 3.3.1, uses 'implementation guidelines'. It presents guidelines that were composed to improve code performance and allow reasoning about scheduling theory. This is the main direction being used. After that, section 3.3.2 discusses the second direction; scenario-based scheduling. We will present the theory behind it. To get a better idea of what we are trying to do, we start with a short introduction to UML-RT and Rose RT. We will also show how they currently handle the scheduling of the generated code.
The goal of this chapter is not directly related to one of the research question. Its intention is to provide the reader with some background information into the scheduling of automatically generated code from UML-RT model.

3.1 UML-RT

'UML Real-Time (UML-RT)' is a modeling language. It can be used to develop real-time systems. UML-RT [14] is an extension of UML [6], based on ROOM [31]. UML is a set of standards and diagrams that can be used to create a model of a system. It shows what a system should do, which components are in, how these components interact, how the software is deployed upon the hardware, etc. In UML-RT, some of the timing aspects of real-time are added, the architecture can be better expressed and it facilitates easier re-use of system components.
UML-RT has two major changes with regard to UML [6]; capsules, ports and connectors have been introduced and a 'time service' is now available. Both of them will be discussed, since they will be used throughout this thesis.

3.1.1 Capsules, ports and connectors

Capsules, ports and connectors have been introduced to better express the architecture of the system and to allow for easier re-use of components. The central constructs in UML-RT are the capsules. These capsules represent the major architectural elements of a system [32]. Capsules can only communicate with other elements through the use of ports. A capsule can have several of these ports which should be linked with connectors to other, similar, ports. By similar we mean that they communicate using the same protocol and that the ports are each other's conjugate. Because of their ports, capsules have a clear interface to the outside. Therefore it is easy to re-use these capsules in other systems. Figure 2 shows a diagram composed of these elements. The big box is a capsule. Inside are four smaller boxes, these are sub-capsules. There are several small boxes on the edges of the capsules, these are the ports.

The lines connecting them are the connectors. In the bottom, you see five ports on the border of the big capsule, these ports can be seen and used from outside the capsule.

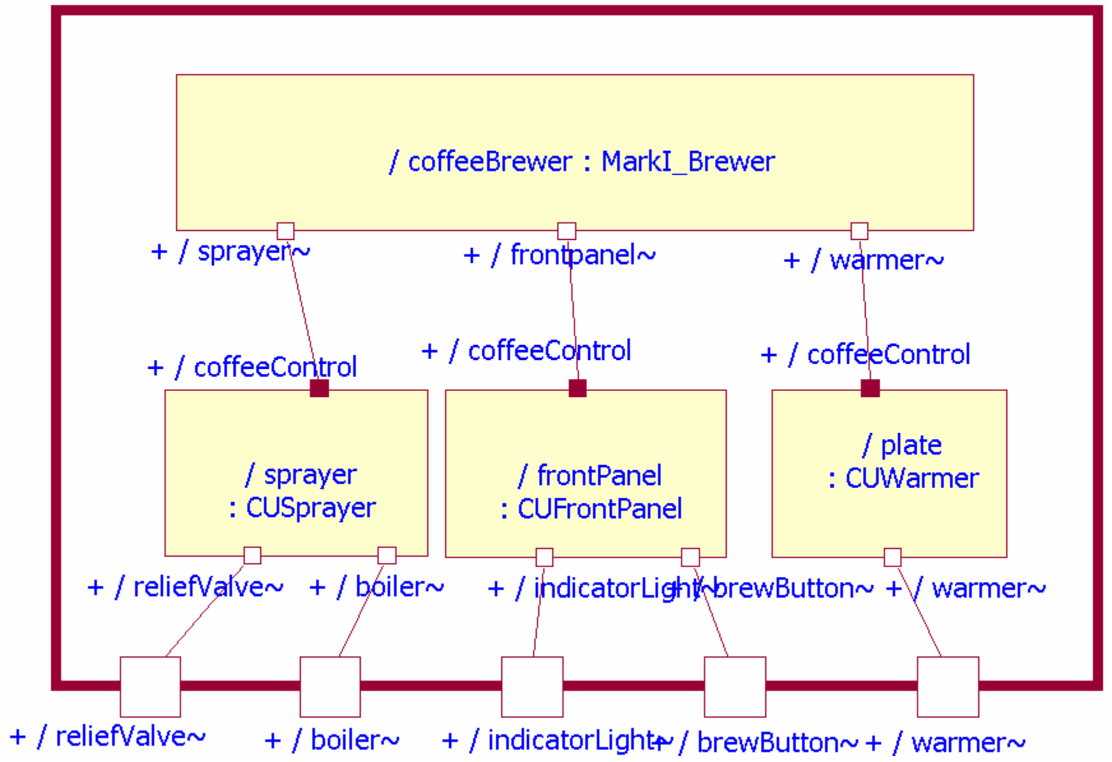


Figure 2: Structure diagram of a coffee machine [4]

Internally, these capsules can contain two things: sub-capsules and/or a state machine. Sub-capsules are instances of other capsules. These instances are assigned a name, the capsulerole, to identify them within the containing capsule. In Figure 2 you see an instance of ‘coffeeBrewer’ within the larger capsule. This instance of ‘coffeeBrewer’ has the capsule role ‘Mark1_Brewer’. By using sub-capsules, the workload is split and therefore the complexity is reduced. The state machine performs the real computations. Capsules are always in a certain state. When a message arrives on one of its ports, a transition from this state can be triggered. This transition executes some actions, possibly send out one or more signals on one of the ports, and then places the capsule in a new state. These transitions are subjected to run-to-completion. That means that whenever a transition is triggered, all actions should be executed; no other transitions will start executing before the first one is done. Figure 3 shows such a state machine. The ‘dot’ in the upper left corner is the starting state, that is the state the capsule is initially in. There are three states, the boxes, and several transitions, the arrows. In the lower right corner of the ‘ReceivingCards’ state, there is an icon that indicates were dealing with a complex state that has sub-states.

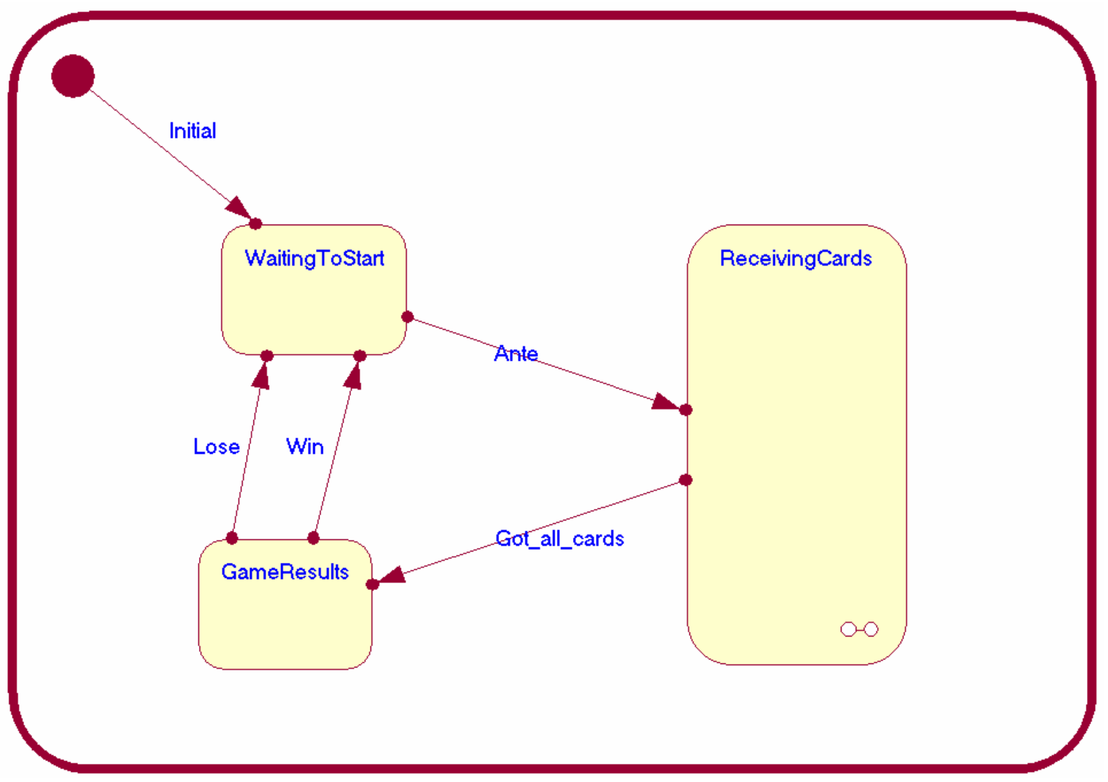


Figure 3: State machine of a card-game player [4]

3.1.2 Time service

A timing service was added to UML to support periodic tasks, using timers or schedule tasks. This timing service can be accessed using an external port, a 'Service Access Point (SAP)'[32]. Capsules can set it to generate multiple timeouts with a certain interval or to generate a single timeout after a specific time.
The timing service has nothing to do with guaranteeing response time or the scheduling of tasks.

3.2 Rose RT

IBM's Rational Rose RealTime [4], henceforth called Rose RT, is a software tool used by Océ in the development of their real-time products. It will also used for our scenario-based scheduling implementation, discussed later on. We will start with giving a general explanation of what Rose RT is and how it works. After that, we will focus on the target run-time system, RTS, and how threads are handled, which is of particular importance with regard to scheduling. And last, we take a look at RRTEI, an interface through which a programmer can access models and functions in Rose RT.
Rose RT is a tool to assist in the software development for real-time systems. It is based on the modeling language UML-RT [32]. Software developers use Rose RT to create such a model of the software system. From this model, the tool is able to automatically generate the corresponding program code. This code should be linked together with one of Rose's 'target

run-time systems' to create a working application. There exist target run-time systems for most of the commonly used real-time operating systems and thus, Rose RT can be used to develop programs for multiple platforms. Figure 4 graphically represent how Rose RT works.

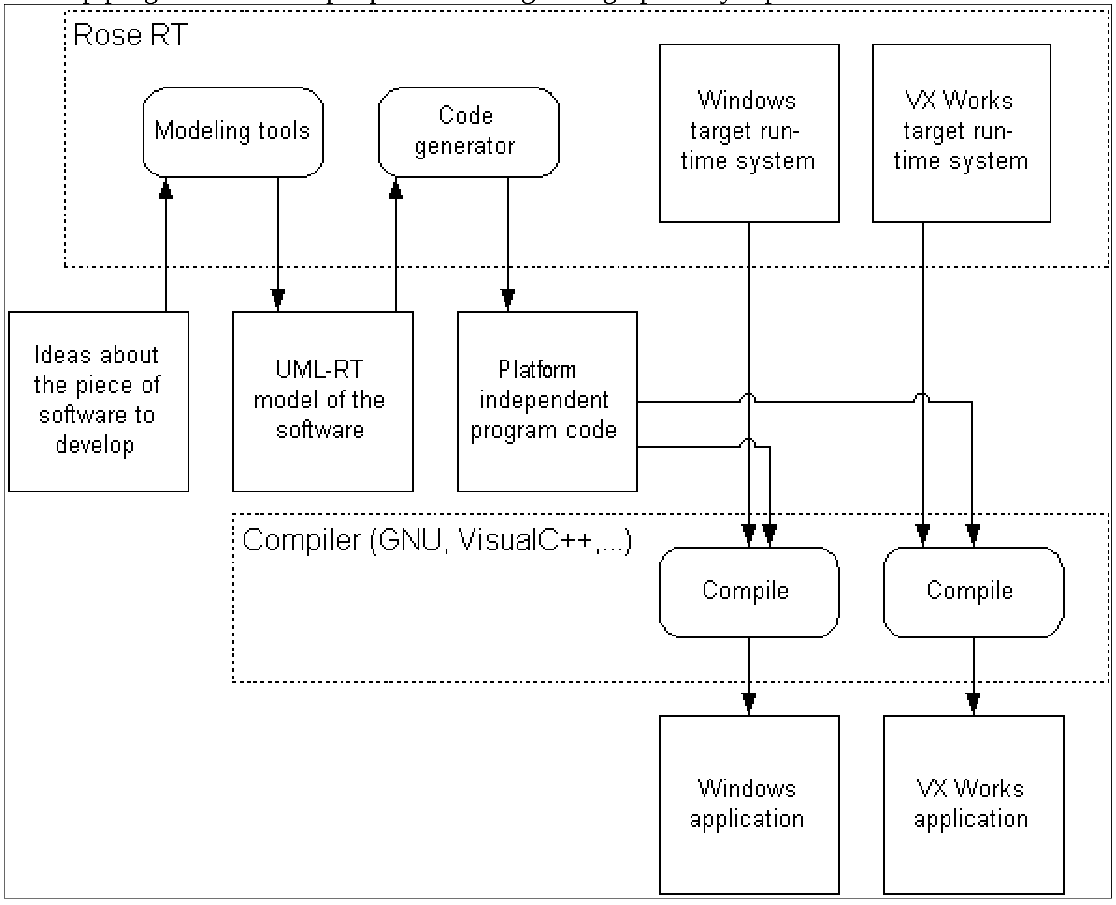

Rose RT
Modeling tools
Code generator
Windows target run-time system
VX Works target run-time system
Ideas about the piece of software to develop
UML-RT model of the software
Platform independent program code
Compiler (GNU, VisualC++,...)
Compile
Compile
Windows application
VX Works application

Figure 4: Schematic representation of Rose RT

3.2.1 The target run-time system and threading

A graphical representation of a threaded Rose RT application, Figure 5, can be found in the documentation accompanying Rose RT [4]. The program will consist of one process that includes one or more physical threads. Each thread is responsible for executing the behavior of one or more capsules. Run-to-completion is guaranteed for all transitions within a thread, but the OS can pre-empt an entire thread, and thus violate run-to-completion in favor of another thread. Each thread is assigned its own message queue and its own controller. This controller handles the incoming messages and executes the behavior of the capsules. The main process creates these threads and controller and does all the initialization and destruction. It also contains some shared structures and services. Threads can be assigned priorities to influence in what order things are executed [4]. These assigned priorities are static, they do not change when the program is executing.
By default, a program created with Rose RT contains two threads. One to handle timing and one that does the rest; execute the behavior of all capsules. To control the scheduling of the generated code, developers can create additional threads, modify their priorities, and map

capsules to these threads. Furthermore developers can assign priorities to the messages on a certain thread, this influences the scheduling within that specific thread.

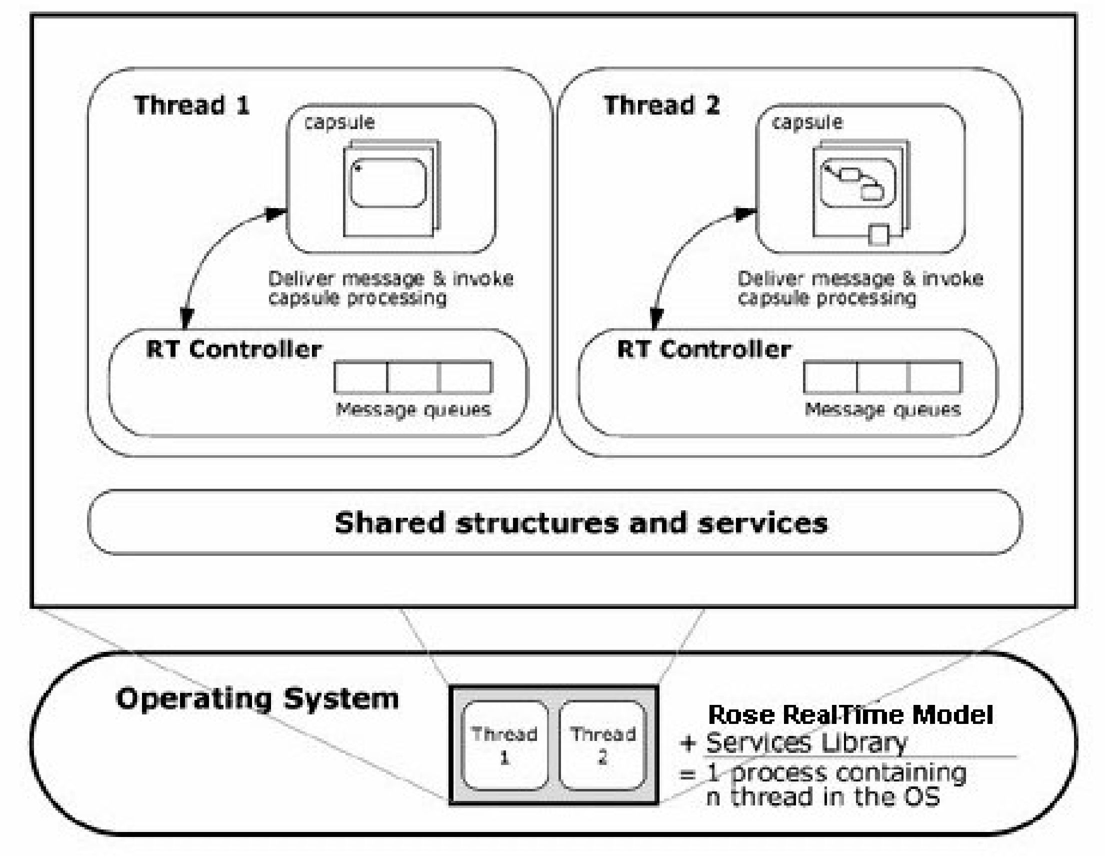

Thread 1
capsule
Deliver message & invoke capsule processing
RT Controller
Message queues
Thread 2
capsule
Deliver message & invoke capsule processing
RT Controller
Message queues
Shared structures and services
Operating System
Thread 1
Thread 2
Rose RealTime Model
+ Services Library
= 1 process containing n thread in the OS

Figure 5: Multi-threaded run-time system [4]

3.2.2 The RRTEI

The *Rational Rose RealTime Extensibility Interface*, or RRTEI, is an interface through which a programmer can access, modify or use models or functions in Rose RT. Programmers can extend or customize Rose RT with it. This RRTEI can be accessed directly using Visual Basic scripts or through the use of OLE automation, which is supported by most programming languages [4].
During this thesis, Visual Basic scripts will be used that use the RRTEI to extract information from Rose's UML-RT models.

3.3 Scheduling at the program level

With regard to real-time systems, the scheduling techniques discussed in appendix A are only a starting point. The programs and tasks that are being scheduled by these techniques can also be 'tuned' for better scheduling performance. We focus on two approaches discussed in literature. First we look at the most common approach, the use of implementation guidelines. After that, we look at a relatively new approach; scenario-based scheduling.

3.3.1 Implementation guidelines

To achieve better results with regard to timing and to make the real-time scheduling more predictable, implementation guidelines have been constructed [28, 29, 36]. These guidelines apply to real-time systems developed with Rose RT. If programs have been developed in compliance with these guidelines, they become more predictable.

The main goal of these guidelines is to reduce the possible blocking time, the time a task has to wait before it is allowed to execute. Given an upper bound on the blocking time for a certain computation, its worst case response time can be determined. Three ways of blocking are distinguished. Each is tackled by the guidelines.

The article [28] describes the following guidelines:

- With regard to message passing between objects, the 'Immediate Priority Ceiling Inheritance protocol' should be used to protect the message queues. This is a derivation of the 'Priority Inheritance' and 'Priority Ceiling' protocols [12, 33]. A mutex gets assigned a ceiling. This ceiling is the priority of the highest priority task that can lock the mutex. When a task acquires the mutex, its priority is immediately raised to the mutexes' ceiling. After the mutex has been released, the priority of the task is restored to normal. In short, this protocol ensures that sending and receiving messages is done at a high priority. This way, these actions are less likely to be pre-empted and the message queue, a critical resource, cannot cause a priority inversion.
- In addition to the use of that protocol, the execution time for these Send and Receive operations must be kept within certain bounds. They should not remain at the boosted priority for too long since that blocks other tasks and interferes with normal scheduling.
- An upper bound on tolerable blocking time should be established for all actors. The execution time of all transitions should be kept below this bound. If it exceeds the bound, the transition has to be divided in two separate transitions. This way, the run-to-completion mechanisms can't cause excessive blocking.
- It's advantageous to have as few threads as possible. However when grouping two actors to the same thread, the previous guideline should be satisfied by both actors for the least upper bound.

The underlying scheduling technique usually being used is RMS. When the guidelines are satisfied, the following formula can be used to determine the response times of the tasks [28]:

$$R_i = \sum_{j \in hp(i)} \left(\left\lceil \frac{R_i}{T_j} \right\rceil * C_j \right) + C_i + B_i$$

Where *Ri* is the response time for task *i*, *Ci* is the execution time of task *i*, *Bi* the blocking time for *i*, *Ti* the period and *hp(i)* is the set of tasks with a higher priority than *i*. So when using the guidelines, we can predict several properties of the produced programs. It can be calculated in advance what the maximum response times of the tasks are. Examples of the usage of this formula on real-time systems can found in [28, 29, 36].

Next to the guidelines, Saksena [28] does some suggestions about assigning priorities and thread allocation. Transactions, sequences of transitions triggered by certain events, will be assigned priorities. The shorter the deadline for an entire transaction, the higher the priority. All actors and messages get the priority of the highest-priority-transaction they are in. Furthermore it is preferable to group actors belonging to the same transaction onto one thread, as long as that complies with the last guideline. This grouping strategy should reduce the number of expensive context switches.

We now have a way to determine the maximum response times of the tasks, so we should be able to produce predictable real-time systems. In practice however, these guidelines are rarely being used. Kim [15] states that this is because of the fact that it is too hard for developers to apply the guidelines in practice. In most cases it is difficult to determine the required ‘upper bound on tolerable blocking time’ for all the actors. And it is also a lot of work to determine the execution times for all the transitions. Furthermore, a transaction can be initiated by different external events, requiring different response times. The guidelines do not support that and require the use of the worst-case response time, which leads to a waste of resources in the other cases. For the simple examples given in [28, 29, 36] it could be done, but in complex real-world projects, the guidelines will quickly become very hard to use. Next to these difficulties, Kim [15] also shows with an example that the task sets generated are not optimal.

3.3.2 Scenario-based scheduling

Saksena et al. [28, 29] already showed us some of the basic ideas of scenario-based scheduling. Only he used the word ‘transaction’ for it. This scheduling method was further developed by Kim et al. [13, 14, 15, 19]. It is claimed to derive a task set that guarantees optimal response times for each event, with the smallest possible number of threads.
The scenarios on which this method is based are defined as *“a sequence of messages and activities that progress once triggered by an external event (message)”* [15]. These scenarios are executed on additional threads. When a scenario is triggered, the priority of the associated thread is set to the priority specified by the scenario. The thread starts executing the transition triggered. All subsequently triggered transitions remain on that thread. Figure 6 illustrates such a scenario by means of a sequence diagram. The marked areas are part of the scenario. So contradictory to the guidelines [28], activities and computations belonging to a certain scenario are *always* executed on the same thread. Because of that, the number of context switches and their associated blocking risks are reduced.
External messages can trigger multiple scenario’s, depending on the state of the system. All possible scenarios triggered by a specific event are grouped into a so called ‘scenario group’. Initially, each scenario group is assigned its own thread. As we have seen with the implementation guidelines, it is best to have as little threads possible. Therefore developers can combine several scenario groups on one thread if these groups do not require executing concurrently.

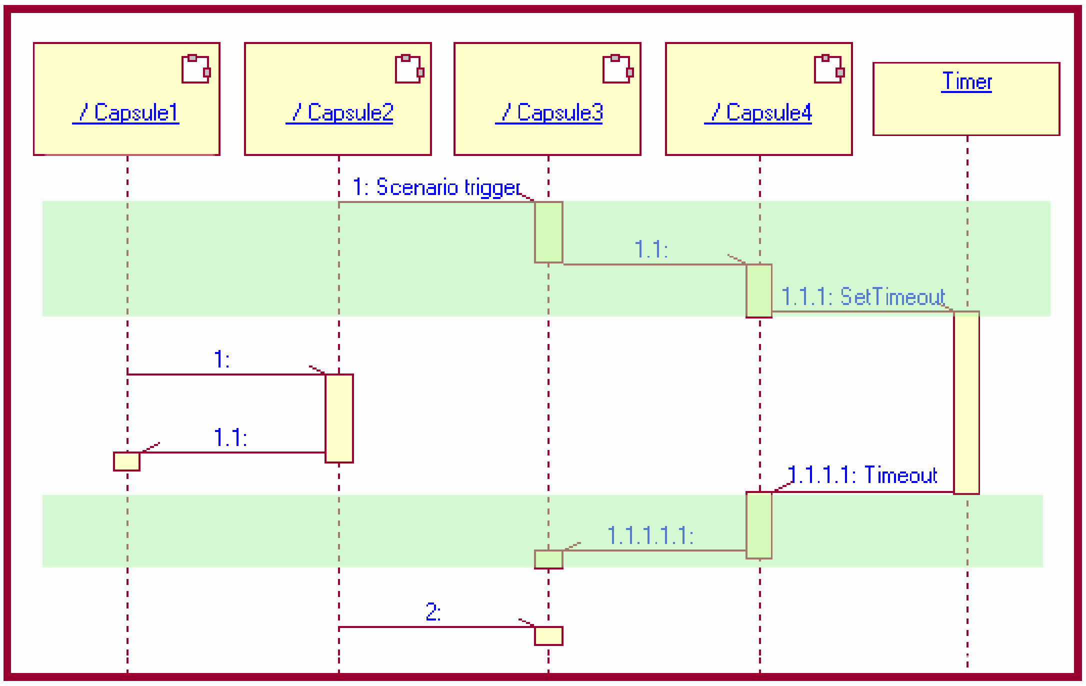


Figure 6: Sequence diagram of scenario

When the scenarios are constructed, the developer has to assign priorities to them. For the capsules involved in a scenario, developers do not have to assign threads, thread priorities or messages priorities anymore. In general, the shorter a scenarios deadline, the higher its priority. When a scenario group is triggered, it does not know in advance what specific scenario has been started and thus does not know what priority it should have. Therefore it will be assigned the maximum priority of all scenarios it could belong to. During execution, it will slowly become clear to what scenario it belongs and its priority is adjusted according to that new knowledge. Objects and threads therefore do not have a fixed priority, it is dynamically adjusted to the scenario it is processing. The underlying scheduling technique uses pre-emption thresholds [36]. Apart from the priorities, a pre-emption threshold can be set. If not, standard RMS [11, 16, 17] will be used.
It is important to note that a certain capsule can belong to multiple scenarios and therefore be executed on multiple threads. To ensure that the run-to-completion demands are not violated, we should protect access to these 'shared objects'. The *Immediate Priority Ceiling Inheritance protocol* [28], discussed previously, will be used for that purpose. When a capsule is executing its behavior, it is locked by a mutex.

3.4 Conclusion

This chapter has provided us with some background information into the scheduling of automatically generated code. We have seen how the generated code can be scheduled by default and we have seen two alternative approaches of handling this scheduling.
Scenario-based scheduling is one of the alternatives. It is a relatively new approach, has been used rarely and, apart from Kim's articles [13, 14, 15, 19], we are not aware of any publications that either disprove or support the claims made. Therefore this thesis continues

on that subject. It will be evaluated whether scenario-based scheduling can be used in a large-scale ‘real-world’ project and more specific, Océ projects.

4 Implementing scenario-based scheduling

Scenario-based scheduling is not being used in practice widely. As far as we know, it is not implemented in any software development tool. The only implementation that exist is SISA, an experimental implementation developed by Kim [15]. The implementation that will be used in this research will be an adapted version of SISA.
SISA is an extension to Rose RT [4], the software development tool already being used at Océ. This chapter begins with an analysis of SISA; what is it, how should it be used, how does it work, which changes are required? Next, we describe what has been done in order to get SISA to work at Océ, what were Océ's specific requirements, which upgrades were made and which bugs have been fixed. In short, how was SISA modified during this research?
This chapter will provide answers to two sub-questions of the first research question. The limitations, restrictions and required changes will be identified.

4.1 SISA, the original implementation

The implementation of scenario-based scheduling that will be used at Océ, will be based on SISA [15]. This implementation is experimental and has not been used commercially. In its current form, it is not ready to be used in companies like Océ. In this section, SISA will be analyzed; how should it be used, how does it work, what is implemented and what not, are there limitations/restrictions, are there bugs, what needs improvement?
There is limited documentation available about SISA, the only documentation that prove useful were the articles describing it [15, 19]. Because of the lack of documentation, most of the information in this section has been based on our own analysis. Test models have been constructed that address most of the relevant aspect and features available in Rose RT. SISA has been tested and analyzed based on these models.

4.1.1 Usage of SISA

In this section the usage of SISA, the original implementation, will be discussed. What actions should the developer take to make use of scenario-based scheduling?
SISA is an extension to Rose RT. Developers should still use Rose RT to develop the UML-RT model. When creating this model however, the developers are less bothered with threads and their priorities. Threads only need to be used for code that contains blocking calls. However, for backward compatibility, developers can still use common threads and priorities if desired.
When the model is finished, SISA comes in. The first RRTEI script searches for possible starting points of scenarios in the model. It looks for transitions triggered by time-outs or external messages coming from a SPP (Service Provision Port [4]). Next, the developer has to prune these starting points, remove the ones that should not become a scenario.
Based on these starting points, a second script should be executed on the model. This pass constructs the scenarios and scenario groups. For all starting points, it is determined which transitions are triggered by that event and all transitions and code that is triggered subsequently. This cascade of messages is a scenario group. By default, all scenario groups are assigned to one thread and have default priorities. The developer was supposed to edit these scenarios using a convenient 'Graphical User Interface (GUI)', however the GUI did

not work because of prior implementation changes. The XML files, in which the scenarios are stored, should currently be edited directly by hand. In contrast to what was said in Kim's article [15], priorities are assigned to scenario groups instead of individual scenarios. Next to the assignment of priorities, pre-emption thresholds [36] can be assigned if desired. Finally, threads have to be allocated. Scenario groups that do not need to run concurrently should be placed on the same thread.
After all scenarios are configured, a third pass of the model is required. In this phase, the UML-RT model will be altered. All actions that trigger one of the scenarios will be modified; new ones replace them. These new functions are defined in a modified run-time system. When a scenario is triggered, these functions in the modified RTS are called. They will setup the thread priorities and priority-inversion-prevention protocol after which the scenario can be started.
Next, Rose RT should generate the program code. After the code has been generated, SISA needs to include the scenario configuration into the code. Information about the threads, their priorities and the *Immediate Priority Ceiling Inheritance protocol* has to be included. When that is done, the code can be compiled and linked with one of the modified run-time systems. This leads to an application that uses scenario-based scheduling.

4.1.2 Workings of SISA

In this section we explain how SISA works internally. The internal structure of SISA is represented by Figure 7. Above the horizontal dotted line is the original run-time system. In there, several threads with statically mapped capsules are shown. There are also messages going back and forth between these threads, such inter-thread messages occur when two capsules on different threads try to communicate. It is important to observe that such messages involve an expensive context switch and thus the number of such messages should be limited. Below the horizontal dotted line there are the scenario-based scheduling additions. Extra threads are created next to the ones already present in the run-time system, note that these threads do not have capsules assigned to them.
When no scenario is initiated, the extra threads created by SISA remain idle. The program is executed by the threads in the RTS as would normally be done when SISA is not present. So it is not required that every action belongs to a scenario. Developers can create threads and code as they were used to; those will work in conjunction with SISA.
As mentioned in section 4.1.1, the code that initiates a scenario has been modified to call special functions in the RTS. These functions activate the specified additional thread. The message that triggered the scenario will be redirected to this thread. If the priority of the thread is less than the priority specified by the scenario, it will be increased. What happens next depends on the priorities and the operating system. The operating system determines which threads are ready to run and executes the thread with the highest priority. That can be one of the standard threads in the RTS, another SISA thread, or the thread that was just activated.
When the operating system allows a SISA thread to execute, its controller takes the message with the highest priority from its queue. Thread priority is increased to the scenarios' threshold value; that is a simple implementation of *pre-emption threshold scheduling* [36]. After the priority has been increased, the controller allows the correct capsule to process the message. If the action sends out signals that trigger other actions, these signals and corresponding actions remain on the thread. The controller will execute those new actions on the same thread, with the same priority and threshold. During the executing of a scenario, there will be no inter-thread message passing. Therefore no expensive thread switches have to

occur. When the scenario has finished, the thread inherits the priority of the highest priority message waiting in its queue. If the queue is empty, the thread becomes idle again.
In the original RTS of Rose RT, each capsule is assigned to a thread. There can be only one thread that executes the actions of a capsule instance. When using SISA, that property does not hold anymore. A capsule can be executed by the thread it is mapped to or by a thread that executes a scenario the capsule is in. See Figure 8 for a common example; two threads are doing computations and share a mechanism to store or process their results. The fact that multiple threads can execute the behavior of a capsule could violate the run-to-completion mechanism and lead to unpredictable results. Therefore all capsule are protected with a mutex. Only one thread a time can execute the behavior of a capsule. To prevent priority inversions [12, 33] and long blocking times, the mutexes uses by the RTS are replaced by *Immediate Priority Ceiling Inheritance* mutexes [28].

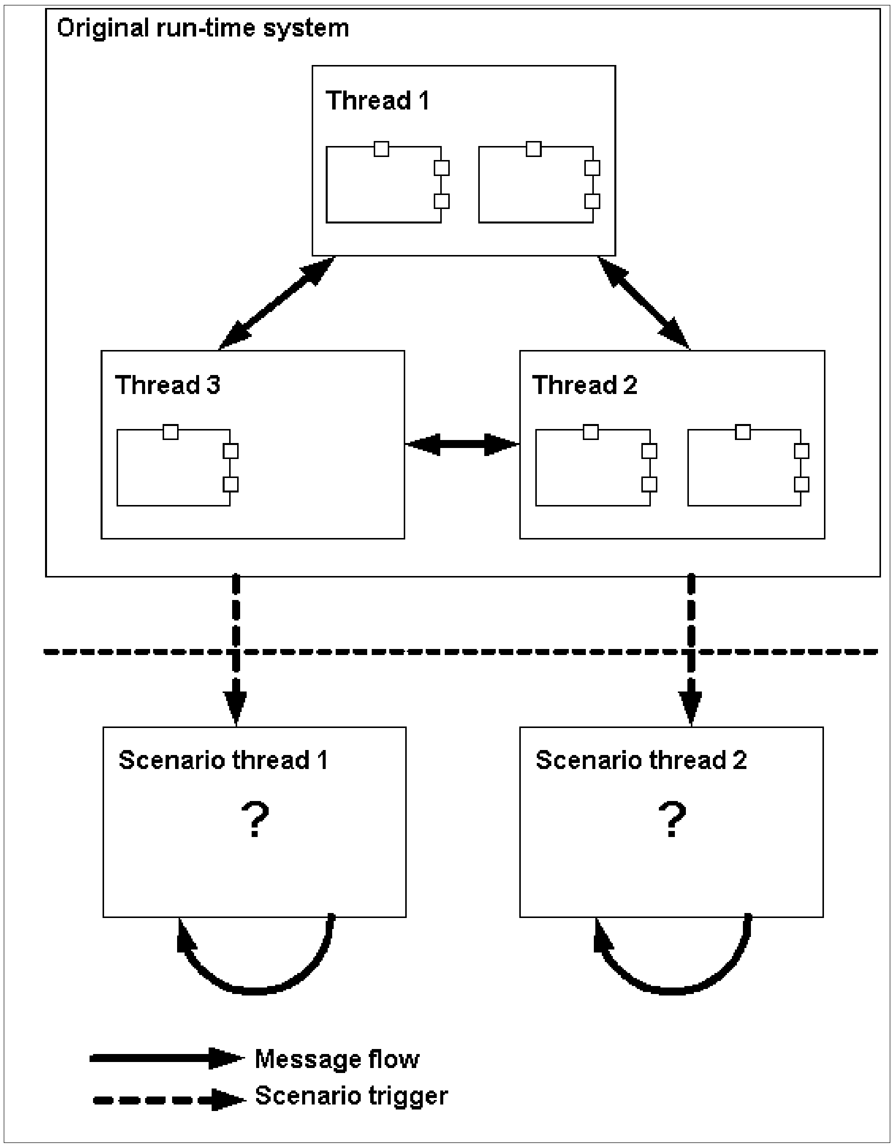


Figure 7: SISA's internal structure

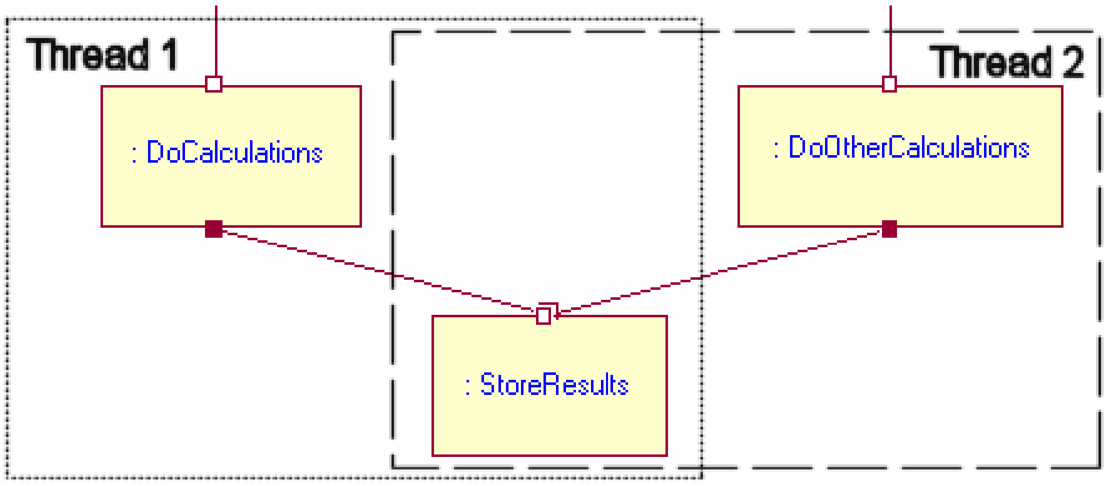


Figure 8: Capsule in multiple threads

4.1.3 Changes required

The original implementation of SISA can be considered a ‘proof of concept’. It illustrates that the idea of scenario-based scheduling works for simple Rose RT models. For large-scale practical use, this implementation is not suited. There is no documentation, only basic Rose RT functionality is implemented, etc…
There are mainly four areas at which the original implementation has to be improved in order to be useful to this research:

- SISA should not crash and should produce the correct output. In other words, there should not be any bugs.
- More functionality of Rose RT should be supported. Currently many features cannot be used because they are not implemented in SISA.
- User friendliness should be improved. Though it is not the objective of this research to provide Océ with a ‘neat and flashy’ implementation, a good user interfaces should be present to illustrate the ease of use.
- Originally, SISA was developed for an old version of the Rose RT ‘Run-Time-System’ and for a different platform. It has to be ported to the new RTS and the platform being used at Océ.

These four points were derived with the goal of this research in mind; to answer the question whether or not scenario-based scheduling can be useful for commercial purposes and large projects. The goal is not to create a neat, flashy, well-documented implementation of scenario-based scheduling that can directly be used by Océ or other commercial parties. Therefore, aspects like providing good documentation and user-manuals are not part of this research. Since the implementation provided by this research is meant only for showing the usefulness of the technique, we will not spend too much time on the maintainability of our implementation.

In other words, our goal is to investigate whether it works. If it does, the results of this research can be considered a starting point for companies to create their own implementation that suits all their needs and meets all their specific demands.

To find the bugs and missing features in the original implementation of SISA, three techniques were used. First, a review of the existing code was done. Just by analyzing the code it could be seen what it can or cannot do and some of the bugs could be found. Second, a simple Rose RT model was constructed that utilizes a wide range of concepts used by Rose RT. That model, discussed in detail in appendix C, also clearly shows the effects of different scheduling techniques and parameters. By applying the original implementation of SISA to this model and by analyzing the crashes and unexpected results that followed, missing features and bugs were identified. And last, we scaled up. Instead of using the smaller test model of phase two, we applied SISA to a real and large Océ project, which will be discussed in chapter 6.
In the following sections we will provide a detailed list of improvements required in the four areas selected. We start by identifying the bugs currently present, next the missing features will be discussed, followed by the user-friendliness and last the platform and environment for SISA

4.1.3.1 Bugs

It is important that the tools used for scenario-based scheduling do not crash or produce incorrect output. There should not be any bugs. In the original implementation of SISA there were. Changing the operating system and Rose RT version caused some of these bugs. And because of the fact that SISA was intended as a ‘proof-of-concept’ instead of commercially usable application, creating robust software was not the main priority. Due to these bugs, SISA either crashes or produces incorrect results. Below we describe the bugs we have found and their consequences.

- SISA produces incorrect scenarios when composite states are involved. The state machines in the capsules include states of course. When these states are complex, they can be divided into sub-states. In the models using these composite states, the scenario could terminate prematurely, missing some of the transitions. SISA stops recognizing the sequence of transitions halfway and terminates the scenario. Experiments showed that the exact problem was with the *junction points*. These points lie at the borders of these sub-states. When transitions go into these junction points, an outgoing transition is triggered immediately after that. This outgoing transition can also include code that sends out messages. SISA stops scanning at these junction points, the code on the outgoing transitions is not taken into account.
- For some models, executing any of the model passes results in the script crashing with a vague error message. The reason has been defined as a failure to extract the top-capsule from the model. Analyses showed that SISA’s scripts parse a certain text-string, extracted from the model, to locate the top-capsule. However in some undefined cases, the syntax of this string slightly differs, which makes the parse fail.
- While running the code modifying Python scripts on Windows 2000, an error with regard to the file-system occurs. This error has been traced to a file renaming operation in the scripts. A modified file is renamed to the original files but these files were not properly closed. On the operating system SISA was designed, that probably went well, however some operating systems require the files to be closed first.

- Errors occur when elements involved in scenario-based scheduling are placed directly in the top capsule. The RRTEI scripts that process the UML-RT model requires 'capsule roles' to differentiate between multiple instances of capsules. Since there can be only one instance of the top capsule, it does not have such a role. Therefore the processing of this capsule fails.
- If there are multiple timers in one capsule, SISA produces incorrect results. The scenario information does not end up with the corresponding timer. When modifying the model, phase three of the RRTEI scripts, it is not verified what timer they are dealing with.
- When multiple transitions in a capsule can be triggered by an incoming message (depending on the capsule's state), not all of them show up in the XML-file describing the scenario. This only happens when more than one transition triggers subsequent other transitions.
- The second pass of the model, where the scenarios are build, sometimes produces incorrect results when a certain port has multiple occurrences in the model.
- Phase two of the RRTEI scripts fails if the C++ lines that trigger a scenario span multiple lines. The script considers only one line.
- Another bug in the second phase occurs when there exists recursion within capsule operations. The script does not check for loops at that point and therefore it will loop indefinitely itself.
- When an active scenario traverses over an unwired port, it gets cut of. The RRTEI scripts cannot pass over these unwired ports.
- The third pass of the model does not modify all the required code. Only the code belonging directly to a transition is checked. However, there also is code associated with *choicepoints* and *capsule operations*. These code fragments should also be considered.
- Some compilers, like 'Visual C++ 6.0' we used, do not accept the code modifications made by SISA. They fail on the commands that initiate the additional threads.
- When ports have a cardinality of more than one, it is an array of ports, priorities are not assigned correctly when using the 'send' command. SISA is supposed to allow assigning different priorities to each of these port instances. However when the 'send' command is used, all ports are signaled at once, all ports receive the same priority information.
- Windows NT uses so-called 'Priority Boosts' [2] to prevent starvation and enhance responsiveness of interactive tasks. Though this is not a bug and in general it is a good thing, it also makes the tasks and scenarios less predictable. The mechanism lets a low priority task pre-empt a task of higher priority if it has been blocked for a long period of time. This 'blocking-factor' is not covered by the RMA theorem [11, 16, 17] and therefore renders this theorem not usable.
- The *Immediate Priority Ceiling Inheritance protocol*, used by the mutexes, results in undesired behavior in two ways. First, the capsule roles taking part in scenarios should be assigned a ceiling equal to at least the priority of its containing scenario. In the current implementation however, this ceiling is assigned to the capsule in general, not to only the specific role taking part in the scenario. Therefore all other similar capsules operate at the wrong priority. The second problem is somewhat more complex. When a mutex is being entered, priority is immediately raised to the highest priority task that could potentially access the mutex [28]. This means that if there is a high priority scenario that could use a capsule, that capsule is always executed at that high priority. In Figure 9 this is illustrated. Assume that 'ImportantTask' executes a scenario, running at a high priority, which uses 'DoCalculations' for some calculations. 'DoCalculations' becomes part of that scenario

and will be protected by a mutex having a ceiling equal to the scenario's priority. When the 'ImportantTask' is idle and the 'NormalTask' is active, the calculations are still executed at high priority. At this point, this priority boost isn't required.

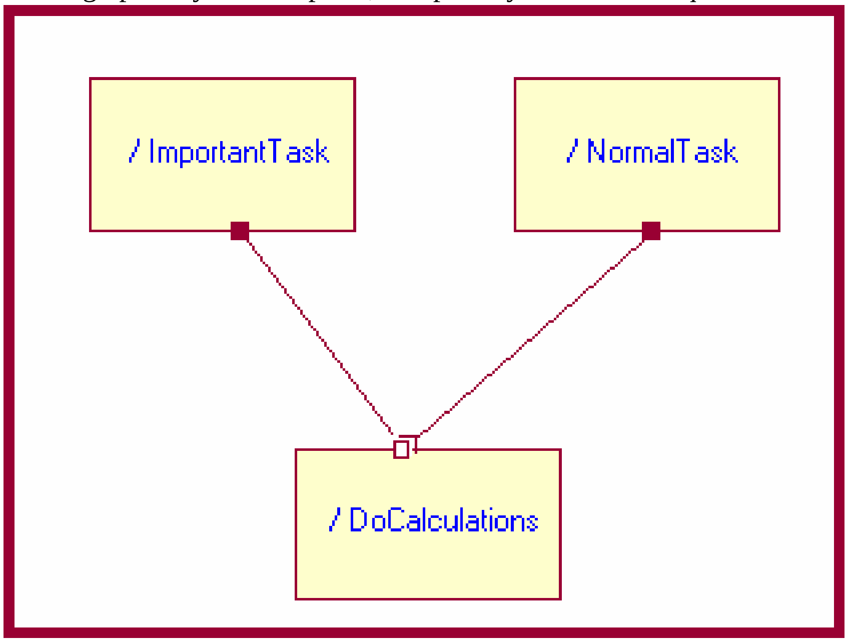


Figure 9: Problem situation for mutex

4.1.3.2 Missing features and functionality

SISA implements the basic things required to use scenario-based scheduling. Some more advanced aspects of Rose RT are currently not implemented. New features introduced in Rose RT version 2003 aren't included either; it was designed for version 2001. Furthermore Océ has some specific demands, things that SISA should be able to do for them. Missing functionality and features are discussed below.

- Only asynchronous sending of messages is implemented. Models that want to use SISA are therefore restricted to the use of these asynchronous messages. Although this is the most frequently used type of message, Rose RT also supports synchronous messages. Because of this, SISA limits the capabilities of Rose RT.
- The timing functions are not completely implemented either. There are relative and absolute timers, timers that expire after a certain interval or timers that expire at a certain, absolute, time. SISA currently not supports the absolute timers. Furthermore, it only accepts timers set with a 'RTTimespec' specifying the desired interval. Rose RT supports using a number (in centiseconds) to specify the interval, but SISA does not.
- A newly added feature to Rose RT is the 'external-port' [4]. External programs can trigger an event on this port. Since it can be triggered by external events, it's another potential starting point for scenarios. It therefore has to be recognized as such by SISA.
- Another potential starting point for scenarios, is the 'RTCustomController'. Through this controller, external programs can initiate communications with the real-time system. SISA does not recognize this controller as a scenario trigger.

- For some of their products, Océ has problems with regard to the initialization. Using correct priorities can potentially solve these problems. Scenario-based scheduling might be a solution. SISA does not recognize the initialization as a scenario, so in its current state SISA can't be used for that.
- In fact, all incoming signals can be potential triggers for a scenario. Even internal actions can initiate a scenario. Allowing all signals as scenario trigger is desired by Océ.
- When there are multiple instances of a capsule in a model and this capsule contains a scenario starting point, this starting point and its associated thread and priority will apply to all instances of this capsule. This isn't desirable.
- In [15], Kim tells us priorities should be assigned to scenarios, more specific, to the 'final node' of a scenario. The implementation, however, does not allow that, priorities are assigned to entire scenario groups. That leads to the same problems for which the 'implementation guidelines' [28] were criticizes in [15]. The entire scenario group should be assigned the highest (worst-case) priority of all underlying scenarios, which is a waste of resources in all other cases except the worst-case.
- While using SISA, capsules can only be incarnated on different threads during the initiation of the model. After that initiation phase, capsules can only be incarnate on their parent's thread. This is a serious limitation, since it is frequently being used.
- Problems can arise when a scenario contains blocking calls. A blocking call waits for some kind of event, this could potentially take quite a while to occur. In the mean time, the thread that does the call gets suspended. In order to prevent other parts of the system from being suspended, the capsules executing these calls should be executed on a separate thread. The problem with scenario-based scheduling is that when a scenario is being executed, control is never passed to another thread (to prevent slow context switches). This means that the blocking call therefore is executed by the scenario thread, and thus, the scenario thread will be suspended. Blocking calls should therefore never be part of a scenario, because that could suspend parts of the system.

4.1.3.3 User friendliness

As we have already expressed in section 4.1.3, our main goal is not to provide a 'neat and flashy' implementation. However for our own convenience and to illustrate the ease of use, some sort of user interfaces is required and it should not be difficult for developers to apply scenario-based scheduling. The original implementation lacks user friendliness on several fields. These are discussed below.

- After you have run SISA on a Rose RT model, you can't easily make modifications anymore. First, you have to do a complete rebuild and second, the prior run has made modifications to your model that you might not need to be there after the modifications. If it appears after testing that a scenario should be removed or reconfigured, developers have to manually check the entire model for code modifications made by SISA and undo them.
- Kim's article [15] promised a GUI to modify the scenarios; adjust priorities, set pre-emption thresholds and allocate scenarios to threads. Due to modifications made to SISA and the XML-file that describes the scenarios, the GUI does not work anymore. Configuring the scenarios now needs to be done manually in the XML-file.
- SISA is not user-friendly. The script that runs the three passes on the model is not neatly integrated in Rose RT; you have to execute it using the script editor. After that, an

incomprehensible pop-up is shown in which you have to select the desired pass. Moreover, it is not robust. If something is wrong (i.e. no active component is set), the script crashed without a good error message.

- Continuing on the bad user-friendliness, SISA consist of several components. These components were made in three different program languages. The scripts that should be run on the model are Visual Basic scripts using RRTEI [4], the scenario editor was made using Java [1] and Python scripts [3] were used for modifying the produced code. They all need to be executed in a different way. So SISA is certainly not easy to use by the developers. Furthermore, maintenance is difficult because knowledge of all three programming languages is required.
- What could be a problem for developers, is that SISA can be used on only one project a time. You cannot configure the files to which the starting points and scenario information is written. Each time SISA generates one of these files, the previous one is discarded and all information is lost.

4.1.3.4 Environment

SISA has originally been developed in an environment different than the one at Océ. Océ uses x86 processors that are not compatible with the Sparc processors SISA was developed for. Also the operating system is different. Originally a SUN or POSIX-compliant system had to be used. Océ uses Windows NT and Vx-Works. The latter is POSIX-compliant, however using native functions will probably give better performance. These platform dependent issues mainly reside in the modified run-time system. This RTS must therefore be ported to its new environment. Next to these RTS issues, some other small issues will arise, like the previously mention file rename bug.
SISA's target run-time system is based on the 2001 version of Rose RT. The newest version is from 2003. We need to upgrade to this new RTS.
Something that will also require our attention are the priority levels. Different operating systems use different priority schemes. The POSIX standard, for which SISA was developed, normally supports 256 different priority levels. Windows only supports 6 (thread) priority levels [2]. Scenarios developed for POSIX systems are therefore not interchangeable with Windows.

4.2 A new scenario-based scheduling implementation

To overcome the limitations, restrictions and bugs present in the original implementation of scenario-based scheduling and to make it more suitable for Océ to use, SISA has been altered. This chapter will discuss what has been altered and why. We will also discuss what has not been changed, which limitations are left and why.

4.2.1 Modifications made

The following modifications were made to the original implementation of SISA:

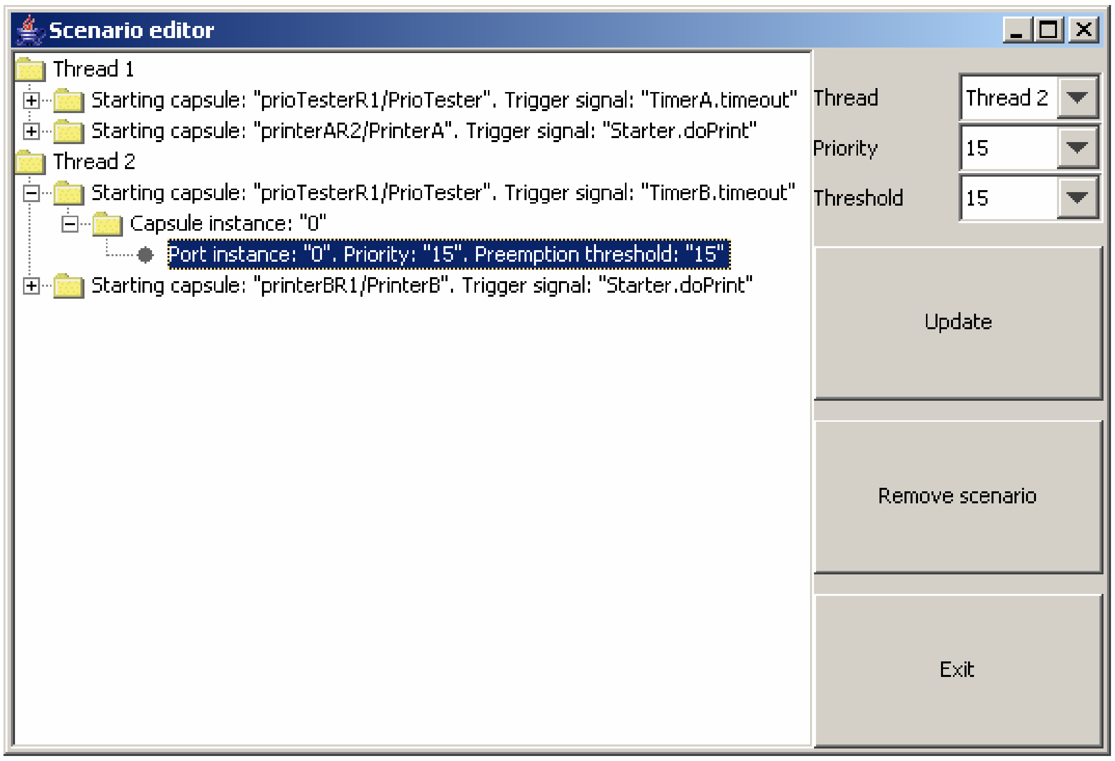
- Initially, checking when to start a scenario was done statically on the model. The model was modified to call special RTS functions to initialize the scenario. We moved to dynamic, run-time, checking of when to initiate a scenario. During start-up, all scenario triggers and the corresponding scenario information is stored in a hash table. When a

‘send’, ‘enable’ or ‘inform’ operation is called, the RTS verifies if it is a scenario trigger. This decision is based on the receiving capsule, the port and the incoming message. If it actually is a scenario, all parameters will be set.
This new approach has advantages but also a major disadvantage. Let’s first discuss that disadvantage. The main problem of run-time checking is performance degradation. ‘Send’ operations occur frequently, a small delay there can have a huge impact. When doing run-time checking, such a delay is inevitable. However we have tried to minimize that delay. A hash table is used to search the triggers. Using a hash table is one of the fastest ways to see whether or not a scenario should be initiated and what the parameters of that scenario are [35]. This can all be done in constant time, O(1) [35]. Next to the extra computation time required, some extra memory is needed to store the hash table and the scenario information. But that isn’t much, at most a few kilobytes. Only for systems very low on memory that could be a problem.
What are the advantages of dynamic run-time checking? Most important is that we don’t need to make modifications to the model anymore. The model does not need to call the ‘special’ RTS functions to start a scenario. Developers can easily modify the model and develop it further, they can also ‘play’ with the scenarios and its parameters, all without wondering if it conflicts with a previous attempt to apply scenario-based scheduling. It is also easier to run intermediate tests on the model. Dynamically checking the signals is therefore friendlier towards the developers.
Another advantage is that the new approach allows for all incoming signals to be potential scenario starting points, which is a requirement of Océ. For the original implementation of SISA it appeared to be very difficult, if not impossible, to trace what model element triggered the signal and had to be modified. Implementing the ‘external port’ triggers and the missing timing functions was shown trivial when using this approach.
Last, a lot of the complexity has been removed from the third model pass by the RRTEI script which originally modified the model. All bugs and problems associated with that pass are eliminated. The fact that not all the code was processed does not matter anymore; we don’t need to check any code at all. Furthermore the limitation of not having more than one timer per capsule has been lifted.
Moving from static to dynamic checking for scenario starting points, makes life easier for the developers, opens up several new possibilities and creates a more flexible system. Whether the performance degradation poses a significant problem remains to be seen and will depend on the situation.

- All the scripts and tools have been made more robust and now neatly report the errors encountered.
  Several of the bugs were fixed by making the scripts more robust. Errors with regard to extracting the top capsule have been fixed by improving the string parser used. It is now more flexible and can cope with the minor variations that occur. File-system operations in the Python script have been improved, files are closed after they have been used and files that are not needed anymore are removed. That fixed the file-system bug that occurred when using the script on a different platform. The Python script now also checks whether or not a file has been previously modified. In its original implementation, it just applied the modifications twice. That corrupted the code.
  If no active component is set, no top capsule is defined in that component or one of the required files is missing, the developer is notified with a pop-up showing the nature of the error. Previously, the scripts just crashed.

Improving the robustness and the error handling is required to be able to use SISA on real-world projects. The original implementation was only suitable for the ‘clean’ models created specially for testing SISA. The original error messages and scripts crashes could only be interpreted by someone who thoroughly knew the code of SISA. Developers that just ‘use’ SISA are not expected to have that knowledge.

- Several improvements with regard to user friendliness have been made. The previously discussed improved error reporting is one of them. Another one that we have already discussed is the fact that there are no modifications made directly to the model anymore.
  Furthermore, a GUI (Figure 10) has been constructed to modify the scenario information files. Developers do not have to manually edit the XML-files anymore to adjust scenario parameters. The GUI gives the developers a clear view of the scenario starting points, the thread allocation, the priorities and the threshold values assigned. Developers are allowed to change scenario parameters and to remove scenarios from being scenario-based scheduled.


Thread 1
Starting capsule: "prioTesterR1/PrioTester", Trigger signal: "TimerA.timeout"
Starting capsule: "printerAR2/PrinterA", Trigger signal: "Starter.doPrint"
Thread 2
Starting capsule: "prioTesterR1/PrioTester", Trigger signal: "TimerB.timeout"
Capsule instance: "0"
Port instance: "0", Priority: "15", Preemption threshold: "15"
Starting capsule: "printerBR1/PrinterB", Trigger signal: "Starter.doPrint"

Thread: Thread 2
Priority: 15
Threshold: 15

Figure 10: The graphical scenario file editor

All scripts have been integrated into Rose RT. The three RRTEI scripts can be individually accessed through the menu, Figure 11. Developers do not have to run these scripts using the script editor anymore and the incomprehensible pop-up to select the pass they want to run is removed. After running the first or second pass of the RRTEI scripts, the produced output files probably need to be modified since the default values that were assigned by the scripts have to be customized. The developer is asked, by means of a pop-up, if he wants to edit the file using the appropriate editor. For example, after the second pass the Java-based scenario editor will be loaded automatically and opens the correct file. Also the code-modifying Python scripts are integrated. These are inserted in the so-called 'make-files'. This means that they are automatically executed after the code has been generated.

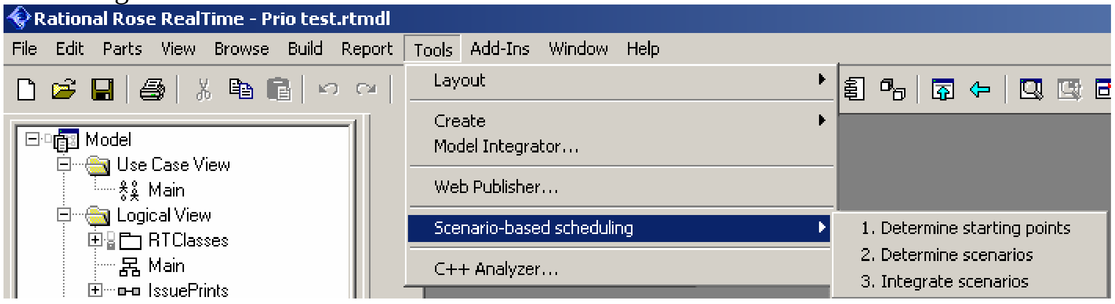


Figure 11: Menu integration of RRTEI scripts

As can be seen, the problems for the developers with regard to the use of multiple programming languages have been eased. The different languages still exist, but the ways of executing them are handled by Rose RT, they are hidden towards the developer.
Furthermore there was the limitation that scenario-based scheduling could only be used on one model a time. That was due to the naming conventions of the intermediate files produced by the RRTEI scripts. These naming conventions have been changed; the component name is included in the filename, which makes the files unique for each component. Developer can work on multiple scenario-based projects without the risks of overwriting another project's files.
User friendliness is important for scenario-based scheduling. If it is too hard for developers to use, they will not use it; they are not obliged to do so The extra burden scenario-based scheduling places on the developers should not outweigh the advantages is has.

- With regard to the use of different operating systems, different compilers and a new version of Rose RT, there also have been several improvements. We have already discussed the modifications made to the Python scripts to fix the file-system problems some operating systems have.
  Another step that has been taken is the upgrade of SISA's RTS, which was based on Rose RT 2001, to the most recent version, based on Rose RT. All modifications with regard to scenario-based scheduling that were made to the 2001 version were extracted. These changes were somewhat adjusted and applied to the RTS of 2003.
  A small part of the RTS contains platform dependent code. In the original version of SISA, there only was platform dependent code for POSIX-compliant systems. Océ uses the operating systems Windows NT and Vx-Works. Windows NT does not use the POSIX standard. Existing Windows NT code was taken from the standard RTS'03 and the new functions with regard to mutexes and thread-priority-adjustments were added. The 'priority boosts' functionality [2], employed by Windows NT, was disabled to improve predictability. Vx-Works on the other hand is POSIX compliant, however we have chosen to re-write the code to make use of Vx-Works' native thread and mutex handling. An advantage of these native mutexes is that they have build-in priority-inversion-prevention functionality. Those native functions are likely to be more efficient than our custom 'Immediate Priority Ceiling Inheritance protocol'.
  The thread priorities posed another problem. Different operating systems use different priority schemes. To be able to use scenario-based scheduling on multiple operating systems and to be able to exchange scenario information, we have to agree on a common

priority scheme. The Vx-Works priority system was chosen, since it allows us to use a wide range of priority levels and thus gives us most flexibility. Priorities range from 0 (highest) to 255 (lowest), with 75 as ‘default’. Important to note is that the lower the number, the higher (more important) the priority. Platform dependent code in the RTS maps these priority levels to native priorities for other operating systems.

- Finally, the first ‘Immediate Priority Ceiling Inheritance protocol’-related problem was solved. For Vx-Works, its native mutexes and priority-inversion-prevention mechanism solved it. Using them does not require us to set a ceiling in advance. For Windows NT we have also eliminated the need to set the correct ceiling in advance. The ceiling is initially set to the lowest possible priority. When a thread tries to enter the mutex, the thread priority is compared to the ceiling. If required, the ceiling is increased and the thread currently in the mutex is boosted. Performance losses should be minimal, these ceiling adjustments occur rarely, so in most cases only a simple conditional has to be evaluated before entering a mutex.
  This approach grants us another huge advantage; we don’t need to track down the scenarios anymore. These scenario traces were only used to determine the ceilings for the capsules. The second and most complex phase of the RRTEI scripts can therefore be left out. That solves all problems related to that phase.

Of all these changes, the following three are most important:
1. No modifications are made to the UML-RT model anymore
2. All messages are allowed to trigger a scenario
3. Easy to use; everything is either automatically handled by make-files or can be accessed through a graphical user interface.

The modifications discussed are all supposed to improve the quality, functionality, flexibility and user friendliness of SISA. These properties are required for scenario-based scheduling to be an option for real-world projects and for it to become a success. Most of the limitations, restriction and bugs discussed in 4.1.3 are eliminated by these improvements.

4.2.2 Known limitations, restrictions and bugs

Not all limitations, restrictions and bugs were solved. In most cases the reason is that Océ does not require these things to be solved, they can easily do without. The list below shows what is missing in the current implementation and why.

- Synchronous messages are not supported by our implementation of scenario-based scheduling. The policy at Océ with regard to synchronous messages is not to use them, since they bring along risks. The sender’s capsule blocks while waiting for an answer from the receiver. That could lead to deadlocks [35]. Because of that risk and the fact that synchronous messages are not a necessity, they are not used by Océ. Hence they do not need to be implemented in the new version of SISA.
- External messages arriving at a ‘RTCustomController’ cannot trigger a scenario in our implementation. This is not implemented, though it originally was one of Océ’s wishes. Programming guidelines at Océ indicate that an event arriving at a ‘RTCustomController’ will immediately be converted into a signal and will be send out on a port. This is done by a simple ‘wrapper’ around the controller. If a scenario needs to be assigned to an event on the controller, developers should assign it to the signal produced by the controller’s

‘wrapper’. Since the effects are the same, there is no need to implement the possibility to directly start scenarios at the controller.

- Despite the fact that Océ indicated that they had problems with regard to the initialization and that those problems could potentially be solved by scenario-based scheduling, they did not require it to be implemented. Océ is currently looking for other ways to solve the problems and the results are promising, so there was no need to implement it here.
- Like in the original implementation of scenario-based scheduling, priorities are still being assigned to scenario groups instead of individual scenarios. As we have explained before, that could lead to inefficiencies in some cases. On the other hand, constantly calculating the priorities and changing them is not efficient either. It is questionable if the performance gains outweigh the additional overhead. That will largely depend on the situation in which scenario-based scheduling is being used.
  Océ does not think that they will benefit from the fact that priorities can be assigned to individual scenarios. Because of that, and the fact that it is difficult to implement, it was decided not to implement this feature in the new version of SISA either.
- The problems with regard to model elements in the top capsule still exist. The RRTEI scripts and the RTS rely on capsule roles, and the top capsule simply does not have one. There is no simple workaround for this problem.
  However we do not think that this will be a problem. Océ uses the top capsules merely as containers, they only contain sub-capsules. Because of the absence of timers or other ports (except for the ‘frame’-port which is not relevant), there is no way in which a scenario can be initiated in the top capsule. Therefore this limitation should not pose any problems.
  If, despite the programming guidelines at Océ, someone does want to initiate a scenario from the top capsule, there is a workaround. The developer can create a new capsule that only contains the original top capsule and assign this new capsule as the new top capsule. This way the original top capsule is not the top capsule anymore in the new situation and therefore it is not inflicted by the limitation anymore.
- As we have explained before, dynamic run-time checking for scenario starting points needs to be done as efficiently possible. Whether we are required to start a scenario is verified based on the capsule role, the port role, the signal and the cardinality of the capsule. Normally, that will be good enough, but there are cases in which it fails.
  Figure 12 shows an example where is goes wrong. There are two instances of ‘CapsuleA’, both having a capsule and port cardinality of one. Since we are dealing with similar capsules, we need to use the capsule roles to distinguish them. However, since they do not share the same parent, this role can be the same for both capsules. When using the criteria given before, these capsules cannot be distinguished. That makes it impossible to define a scenario starting point that applies to only one of them.

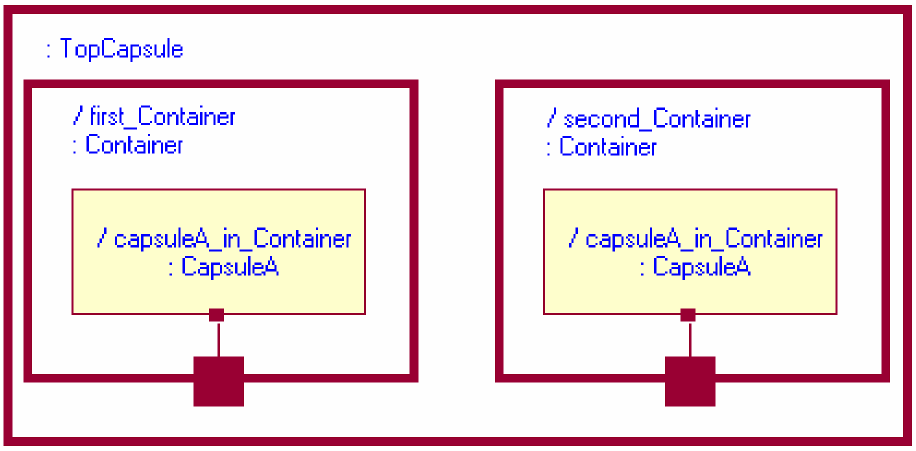


Figure 12: Two indistinguishable capsules

There is a way to overcome this problem. We have to add another criteria to determine the equality. The path of roles till the top capsule is chosen for that purpose. For the left instance of 'CapsuleA', that would become "TopCapsule/first_Container/capsuleA_in_Container". This extra criterion makes the capsules distinguishable and therefore allows the starting of a scenario in only one of the capsules. However, constructing the path-string is a complex operation and it is not preferred to do this each time a message is being sent. By default, this extensive checking is therefore disabled; it can be enabled with an extra parameter to the Python scripts.

- The second problem with regard to the 'Immediate Priority Ceiling Inheritance protocol', where threads often run at the wrong priority, is still present for Windows NT. For Vx-Works using the native mutexes solved it. Depending on the situation, this can be quite a problem. A possible solution would be to use a different protocol. The 'Basic Priority Inheritance Protocol' [33] for example, is not subjected to the problem described. However that protocol introduces new problems like possible deadlocks and blocking chains [33]. All approaches have their pros and cons; it depends on the situation which is best.
- The problem with regard to scenarios not being able to include blocking calls has not been solved. A possible solution could be to allow the user to manually terminate a scenario. However this is currently not implemented. It therefore remains a restriction not to use blocking calls in scenarios.

There are still some limitations left, however none of them appears to be a show-stopper. Important to note is that the implementation differs from the theory on two points. We have not used the described protocol to prevent priority-inversions; some inefficiencies in that protocol were identified. Furthermore, priorities are assigned to what the theory calls scenario-groups. That requires the developer to set less priorities and, when using sub-scenarios, does not offer less functionality. It also reduces processor overhead.

4.3 Conclusion

In this chapter we have answered two sub-research-questions. The limitations of the original scenario-based scheduling were identified in 4.1.3; the list provided there answers the first

sub-question. In 4.2, we have discussed the improvements made to the original implementation and which limitations and restrictions are still left in the new implementation. That answers sub-question number three of the first research question. This new implementation we have developed will be used during the remainder of this research.
We still need to address the scalability issues. Theoretically, scalability appears not to be a problem. In chapter 6 our new implementation will be applied to a large project. Practice will show if there are scalability issues and whether the quality of our implementation is sufficient to handle large project.

5 Model to assess performance

In chapter 4 we have created an implementation of scenario-based scheduling. We now have to determine whether using this implementation is an improvement to the original development process. To make this comparison, we first have to determine where we should look at. Which aspects of the real-time software development process and the produced product are important? This chapter will provide an answer to that question, the first sub-question of the second research question.
First, we will take a look at the background of software measuring; this also leads to the selection of a technique for developing the measurement model. After that, the measurement plan for our specific case, derived using Park's guidebook [23], will be discussed. The detailed steps of this derivation can be found in appendix D.

5.1 Software measurements

There does not exist a universal way of measuring software and its development process. Much literature exists about how software and its accompanying development process can be measured and assessed. We need to narrow down this literature and select an approach that best serves our specific needs. Several variables influence the set of measurements and the measurement plan to choose. You need to have a clear idea of what you want to learn, for what purpose you are measuring and at which level you are doing the measurements.
We will come to the 'what to learn'-aspect later. First we will start with the purpose of learning and measuring. Park et al. [23] identified four main reasons for measuring software processes and their products:

- to characterize
- to evaluate
- to predict
- to improve

Each purpose requires different information and therefore has a distinct way of doing the measurements. In our case we are measuring 'to evaluate'; we evaluate the performance in order to compare both approaches. That is the first step in narrowing down our search for a good measurement plan.
We can narrow down the set of possible measurements further by considering at which level we are measuring. There can be identified two categories with regard to this aspect [25, 26]:

> *"**Process** indicators enable a software engineering organization to gain insight into the efficiency of an existing process (i.e. the paradigm, software engineering tasks, work products, and milestones). They enable managers and practitioners to assess what works and what doesn't. Process metrics are collected across all projects and over long periods of time. Their intent is to provide indicators that lead to long term software process improvement."* [25]

And;

> *"**Project** indicators enable a software project manager to (1) assess the status of an on-going project; (2) track potential risks; (3) uncover problem areas before they go 'critical'; (4) adjust work flow or tasks, and (5) evaluate the project team's ability to control quality of software work products."* [25]

Applying scenario-based scheduling is a modification to the development process. So we need to assess whether the scenario-based scheduling approach works for the process, we are not trying to control an individual project. In this phase we can also see a problem emerging, namely the 'long periods of time' over which the measurements have to be taken. The duration of this thesis only allows us to assess the process based on one specific project. That will give a rough indication of whether or not scenario-based scheduling works. To clearly measure the impact, a further study will be required.
Furthermore we should only focus on the objects for measurement that are likely to change due to the use of scenario-based scheduling. For example, the literature commonly uses the 'lines of code (LOC)' as a measurement [23, 24, 25, 26]. The effect of scenario-based scheduling is not likely to have a significant impact on the 'lines of code' produced. Therefore we can leave measurements like this one out, they do not change and by this, they do not affect the choice of whether or not to use scenario-based scheduling.
Literature also gives us several guidelines to which good measurements should adhere. Roche and Pressman [25, 26] use the following list of demands with regard to the formulation of metrics:

- measurement objectives should be established prior to data collection
- metric definitions should be clear and unambiguous
- metrics should be constructed upon an underlying theory that has validity within the domain of application
- metrics to be useful ought to be tailored to specific products or processes

With regard to the analysis they provide the following guidelines:

- data collection and analysis should be automated
- appropriate statistical techniques are necessary to establish relationships between intermediate and final product characteristics
- interpretative guidelines and recommendations should be established for each metric

We will try to apply to these guidelines as far as possible with regard to the construction of our measurement plan.
Now let's consider the 'what to learn'-aspect. There are lots of reasons why software measurements should be taken. The exact reason why is probably the most important factor in determining what measurements to use. From this perspective, the 'goal-question-metric (GQM)' method has been developed [9, 10, 27]. This method starts with goals, what you want to learn. Since there is something to learn, there are questions related to these goals. From these questions, measurements can be derived to answer them. The GQM method explicitly links the measurements to the goals. Therefore, the purpose of each of the measurements should be clear. This GQM method is a well-known and popular approach and will probably become the de-facto standard for software measurement [34].

To derive our measurement plan, we will use a guidebook from Park: “Goal-Driven Software Measurement” [23]. This guidebook is based on the GQM method with some additions. It provides a detailed step-by-step approach in establishing a measurement plan starting from the business goals. When using this method, we should keep in mind all the things discussed above.

5.2 Measurement plan

In this section we are going to present our measurement plan. This plan addresses why, what and how we are going to measure. We have used Parks’ guidebook [23] to derive this plan. A detailed description of all the intermediate steps can be found in appendix A.
First, we start with describing the objectives of our measurement plan. After that, we give a detailed description of the plan, what should be done and what should be measured in order to achieve the objectives. Finally, we will discuss how the gathering of the measurements we have identified should be implemented.

5.2.1 Objective

In chapter 4 of this thesis we have described the creation of an implementation of scenario-based scheduling. This scheduling approach could potentially make it easier for the software developers at Océ to create real-time software that meets their real-time demands. The aim of the current section is to formulate a measurement plan that determines the impact of scenario-based scheduling.
In order to achieve that objective, several things have to be measured. First, we need to assess the quality of the produced real-time software by its response times, its ability to meet deadlines and its hardware requirements. Second, we need to assess the efficiency of the development process by the number of man-hours and other resources that are invested in it.

5.2.2 Description

This section will describe the details of the measurement plan. We will first take at look at the goals involved, what the general goal is and how it leads to more specific measurement goals. After that, we focus on these measurement goals and which measures are required to achieve them. Finally, the scope of this measurement plan will be discussed, that is, what is its range of application?

Goals
Our main goal is to “*assess the real-time software development process in order to determine the impact of scenario-based scheduling*”; we need to determine if it is a success. This goal consists mainly of two components: the real-time *software produced* and its *development process*. In order to achieve the main goal, both of the individual components will have to be assessed first. The produced software is the most important one. All measurements derived from that sub goal will get the highest priority.
With regard to the assessment of the real-time software quality we have identified three factors that we consider most important in determining the quality. Those are the response times of the critical tasks, whether or not there are other tasks that miss their deadline and the hardware requirements of the software. These factors are things we want to measure; they are our measurement goals.

The performance of the software process has two measurement goals of its own. Important are the amount of man-hours invested in it and the resources that are required for doing the scheduling related tests.

Measurements
The five measurement goals derived in the previous section require the certain information and measurements. For the product related goals those are:

- **Effects on critical tasks.** Important for the critical tasks are their response times. First we would like to have some information about these response times in practice, what is the maximum response time encountered, what are its minimal and average values and is this response time stable? We therefore need to implement a way to measure the response times of the critical tasks. Second, theoretical information with regard to the predictability is needed. Can we prove the existence of an upper bound on the response time of a certain task?
- **Effects on all tasks.** While achieving minimal response time for the critical tasks is nice, it is important for all tasks to meet their deadlines. In order to ensure everything else works as it should, the system should pass all the ordinary tests devised for it. No errors with regard to scheduling may occur. An example of such a scheduling related error is the buffer overflow. When such an error occurs during testing, the ‘consumer’ of this buffer requires a tighter deadline.
- **Effects on hardware usage.** With regard to the hardware, the two most important things are the CPU and the memory. The operating systems that are being used provide ways of measuring both of them. If the usage of both is low and the response times and deadlines are within safe limits, slower and cheaper hardware can be considered.

And for the process-related goal they are:

- **Effect on man-hours.** The amount of man-hours invested in a project is an indication of the productiveness of these people. If fewer hours are required for achieving the same goals, the performance of the team is better. First, we would like to know the amount of man-hours invested in scheduling related aspects. That is supposed to be less when using scenario-based scheduling, the additional tools are supposed to make it easier for the developers and allow them to do more in less time. Second we are also interested in the total amount of man-hours invested in the entire project. Using scenario-based scheduling might have an unpredicted influence on other phases of the development process. The cumulative effects should be visible when looking at the total amount of hours.
- **Effect on test resources.** In the original real-time software development process, there is a test/fix-loop that needs to be iterated several times. For doing these tests, the actual hardware or a simulator is required. These things are expensive and scarce. It is preferred to keep their use to a minimum. That leaves these resources available for others and does not require the developer to wait for them if they are being used. We need to measure the usage of both the simulator (BBO) and the real hardware (EPT).

Scope
The goal of this measurement plan is to evaluate the real-time software development process and its product to determine whether or not scenario-based scheduling is a good thing. Measurements should only be applied on a few real-time software development projects to

make a solid evaluation. These software development processes should use UML-RT as the modeling technique, use Rose RT as the development tool and should be representative for Océ projects.
During this thesis, we will conduct a preliminary evaluation. We take an existing Océ project and redo the scheduling related steps of that project. Both the original and the new process result in a final product; therefore the product measurements can be taken and compared. The process-related measurement however, will pose a problem. It is not feasible to redo the entire project during the thesis. The partial development trajectory we are going to follow differs significantly from the original process. Therefore we cannot do a solid comparison of both processes. We will try to provide the measurement data from our test project as far as possible. That will provide the reader with a rough indication of what scenario-based scheduling does to the process.

5.2.3 Implementation

In the previous section several measurements were proposed. This section describes the actions to be taken to implement them. For each of the measurements we will show if and where it can be found. If the information required is not yet available, we will show what steps are required to implement a way of gathering the information. We start with the product-related measurements and continue with the ones used to measure the performance of the process.
There are five measurements that are related to the product. The object of measurement is the *final* product. Therefore all of these measurements should only be taken once, when the product has been completed.

- **Response times.** The response times of the tasks, or in our case scenarios, are too small to measure manually. Logging should be applied by the code. At the beginning and the end of the interval, the time should be logged. For this logging and time stamping, we can use Océ's logging tool. A script can then be used to extract the required information from these logs.
- **Theoretical upper bounds on response times.** These values will have to be manually calculated. Theory about RMA [11, 16, 17] and pre-emption threshold scheduling [36] can be used for this. Useful information required for these calculations can be gathered from the UML-RT model.
- **Missed deadlines.** In order to verify whether or not the entire systems still works and thus, none of the deadlines are violated, we use the standard tests devised by Océ. If the machine passes all these tests without showing scheduling related errors, we assume the machine works fine.
- **CPU usage.** This can be measured automatically by tools in the operating system. For Windows there is the 'task manager' and for Vx-Works there is WindView [8]. There are however some problems with the configuration of most of the systems produced by Océ. The real-time software runs at a Vx-Works target. WindView however, requires another computer to extract, store and analyze the log-files from this target. Only one computer is connected to the target, the one that controls the user interface of the machine. Our only option is to use that machine for extracting and storing the log-files. To do that, WindView normally requires the Tornado environment to be installed and running on that computer, however it is not in our case. We have therefore extracted parts of Tornado that retrieve and store the log-files and have modified them so they do not require Tornado to

be installed anymore. Furthermore we have created a small program through which a Rose RT program can start and stop the logging of CPU-usage. This way it can accurately be controlled when to start and stop logging. Afterwards, these log-files can be copied from the machine running the user interface. They can then be analyzed on a computer that does have Tornado installed.

- **Memory usage.** With regard to memory usage, the same measurement tools can be used as with CPU usage. For Vx-Works there is also a tool available called ‘memShow’ [7]. This tool display memory statistics at a specific point in time.

Now we have come to the development process related measurements. These measurements require continuous recording during the entire development process. We cannot apply them to our test project, since the project does not simulate the entire process. For the test project, these measurements will have to be manually recorded.

- **Total amount of man-hours spend on the entire project.** The man-hours spend on a certain project are already being recorded at Océ. They use the so-called TOCK database for that purpose. Each employee has to specify the amount of hours worked on a project on a weekly basis. Required information can be extracted from this database.
- **Total amount of man-hours spend on scheduling related activities.** Again, the TOCK database can be used for this. However some small modifications have to be applied. For each project, it should also be recorded on what the hours are spent. A separate entry for scheduling related activities can be created. Now, the required data can be extracted from this database afterwards.
- **Testing hours on the simulator.** Currently, a reservation sheet is being used for the BBO. If someone wants to run some test, a reservation has to be made and recorded on this sheet. A new field should be added to this sheet which indicates the purpose of the tests (if it is scheduling related). At the end of each week, this data can be collected.
- **Testing hours on the actual hardware.** The same reservation sheets are used as for the simulator. We propose the same measurement approach.

Taking some of these measurements might influence the process or the product. Measuring and logging processor usage requires some processing capacity itself; the measurements will therefor slightly raise the processor load. This also goes for other measurements like determining the response time for example. However that should not be a problem. We are comparing two situations. Both situations will be influenced by the measurements in the same amount. If logging the response times extends the response time with 1 millisecond, that will be the case for both the original and the new situation. They can still be compared.

5.3 Conclusion

Figure 13 provides a graphical overview of the measurement plan discussed in 5.2. It visualizes the way it has been derived. It shows the linkage between main goals, measurement goals, the measurements and the sources of information. The branch on the left, derived from the product related sub goal, is the most important one and will get the highest priority.
This measurement plan provides an answer to the first sub-question of the second research question. It allows us to assess the real-time software development process and the produced product.

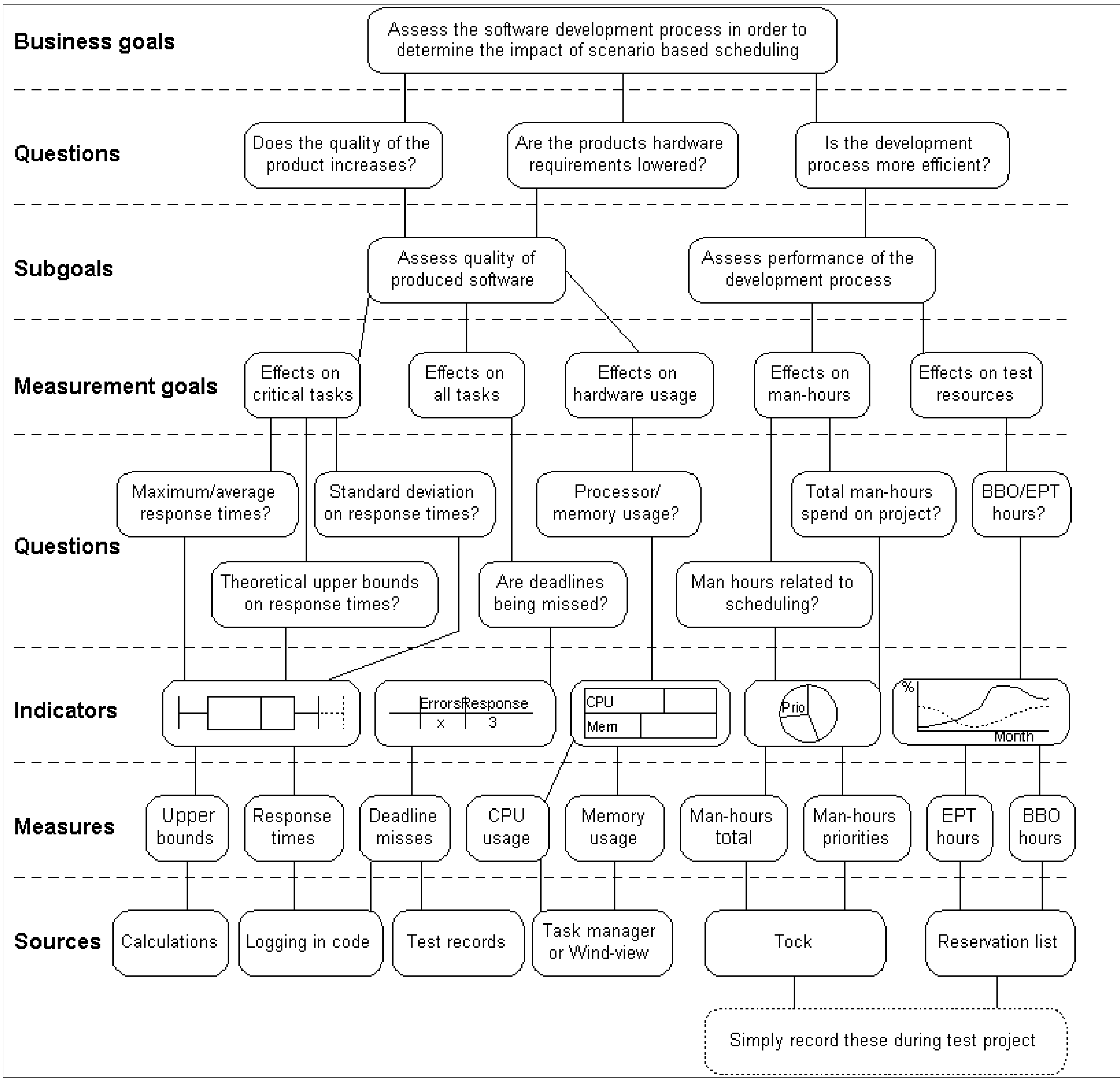

Business goals
Assess the software development process in order to determine the impact of scenario based scheduling
Questions
Does the quality of the product increases?
Are the products hardware requirements lowered?
Is the development process more efficient?
Subgoals
Assess quality of produced software
Assess performance of the development process
Measurement goals
Effects on critical tasks
Effects on all tasks
Effects on hardware usage
Effects on man-hours
Effects on test resources
Questions
Maximum/average response times?
Standard deviation on response times?
Processor/ memory usage?
Total man-hours spend on project?
BBO/EPT hours?
Theoretical upper bounds on response times?
Are deadlines being missed?
Man hours related to scheduling?
Indicators
Errors x
Response 3
CPU
Mem
Prio
%
Month
Measures
Upper bounds
Response times
Deadline misses
CPU usage
Memory usage
Man-hours total
Man-hours priorities
EPT hours
BBO hours
Sources
Calculations
Logging in code
Test records
Task manager or Wind-view
Tock
Reservation list
Simply record these during test project

Figure 13: Overview of the measurement plan derivation

6 VarioPrint 6250 project

To analyze the effects and impact of scenario-based scheduling and to assess whether or not it is suited for large commercial projects, our implementation was applied to such a project. This chapter will describe our experiences with that test-project, Océ's VarioPrint 6250. We start with a short motivation of why we have chosen our specific project. Then, in section 6.2, we will describe a scheduling related problem that we have tried to solve using scenario-based scheduling. The results of this attempt will be described afterwards; have we been successful, what changes were required to the original model, what problems were encountered, etc…
The remaining scalability issues of the first research question will be addressed. Can scenario-based scheduling handle this large-scale project and is it technically feasible to use it in the development process?

6.1 Why the VarioPrint 6250?

In the original articles describing the theory of scenario-based scheduling [15, 19] small example projects can be found. We ourselves used a small test model, described in appendix C for some of our tests. One of our goals however, is to show the effects on large industrial projects and for that purpose these small examples are not suited.
Therefore we need such a large industrial project. The recently launched VarioPrint 6250 [22] was chosen. It can surely be considered a large project, according to Océ's numbers 1.500 man-years have been invested in developing this product [22]. Furthermore, it is a very recent project that uses new and complex technologies; it was introduced as the world's fastest duplex printer. If scenario-based scheduling yields positive results for that project, it will likely be very impressive.

6.2 Direct stops

For this research, we restrict ourselves to only one of the scenarios in the VarioPrint 6250. We will consider the so-called 'direct-stop' scenario. During such a direct stop, a serious malfunction occurs that requires the system to immediately stop what it is doing. It is important that the system is stopped as soon as possible. When a direct-stop scenario is not handled fast enough, sheets of paper could pile up in the 'process-area', the part where the actual printing is performed. If these sheets of paper get stuck there, they cannot be removed by the operator of the machine anymore. A technician should come by to fix this problem and in the meanwhile the system cannot be used. For a high volume system like the VarioPrint 6250 that is very costly.
Handling direct-stops faster reduces the risks to the machine and the risks of downtime due to required maintenance. We are going to use scenario-based scheduling to assign a high priority to these direct stops to handle them faster. The remainder of this chapter explains how it was done and what our experiences are. In the next chapter, the actual effects will be analyzed. Did we actually reduce the direct-stop time, does the rest of the system still meets its performance requirements and what was the effect on the development process?

6.3 Applying scenario-based scheduling

We have successfully applied scenario-based scheduling to the direct stop scenario of the VarioPrint 6250. In order to make it work, we had to make some minor changes to the original model. These changes are described in 6.3.1. Apart from these changes, applying scenario-based scheduling was shown to be easy.
Two direct-stop starting points have been identified; a software- and hardware related one. At both points the priority is raised. Within the direct stop, the stopping of the process-area is even more critical than the rest, therefor a sub-scenario raises its priority even further.
A small problem emerged with regard to responses to the direct-stop arriving over the CAN-bus. They are not by-default part of the scenario that required the reply. This could be solved by either re-starting the scenario when the reply comes in or let the original scenario do a blocking wait for the CAN-reply. However in our case it does not really matter if these replies are not being handled at a high priority, they are just notifications of what has already been done by the parts of the machine at the other end of the CAN-bus.
Some quick tests showed that the machine appears to be working correctly when using our modified run-time system and the scenarios. The log-files indicate that the RTS recognizes direct stops when they occur and starts the appropriate scenario. So it appears scenario-based scheduling is working for the VarioPrint 6250, an example of a large projects. However we still don't know whether it is a real improvement. That will be determined in chapter 7.

6.3.1 Required changes to the VarioPrint 6250

In order to be able to use scenario-based scheduling with the VarioPrint 6250, we had to make some changes to the model; the product was not originally developed with scenario-based scheduling in mind. We are now going to discuss these changes.

- Of course, we need to use our customized run-time system. There are however some complications in changing the run-time system. Our RTS is backward compatible with the original RTS, but Océ had made some modifications of their own. Ideally, both modified RTS' should be merged, but that goes beyond the scope of this research.
  Océ's RTS contains two major additions: an improved timer and additional logging. We do not need the extra logging, the VarioPrint 6250 makes no use of it; its functions can simply be replaced with dummies. The improved timer consists of only a few simple functions; we have integrated them into our RTS.
- Another problem with regard to the run-time system is the fact that it uses 64-bit integers for its timers. The Vx-Works version being used does not support the associated 64-bit calculations, more specifically, it does not support modulo operations with 64-bit values. Additional object files have to be linked in order to support these operations. These different timers are also used by the model itself. Fortunately, these timers are being accessed through a wrapper. Only this wrapper-class needs some minor modifications to make the new timers work.
- The generated code needs to be modified so it contains all the details of the configured scenarios. We need to automatically add this scenario information each time that the files are generated. We therefore include an additional make-file into the original ones.

6.4 Conclusion

By successfully applying scenario-based scheduling to the VarioPrint 6250, which is an example of a large project, we have seen that the scheduling technique can handle that scale and supports all functionality involved. The remaining issues of the first research question are therefore solved, we can handle the large scale and it is technically feasible to integrate the scheduling technique into Océ's software development process.

7 Analysis of the modified VarioPrint 6250

In the previous chapter scenario-based scheduling was applied to handle the direct-stop scenario in Océ’s VarioPrint 6250. We have seen that it works well at the first glance. In this chapter we present a more detailed analysis of the impact of our changes on the VarioPrint 6250. The model described in chapter 5 will be used for this analysis. This chapter addresses the remaining questions of the second research question. The scope of this research does not allow us to completely follow the proposed measurement plan. We cannot come to scientifically sound conclusions with regard to the development process since we cannot replicate the same conditions under which the original scheduling parameters and settings were developed. Therefore we will mainly focus on the product measurements, which were also assigned the highest priority in the measurement plan.
In the first section, 7.1, we will discuss the product measurements. How we obtained them and what these values are for both the original and the adapted VarioPrint 6250. We will also try to explain the values we have found. Next, we will nevertheless attempt to say something about the development process. Finally we draw a conclusion; was the appliance of scenario-based scheduling to the VarioPrint 2650 a success?

7.1 Impact on the final product

There are mainly three product-related measurements. We start by analyzing the response time of the critical task, in our case the direct stop. After that, the other tasks will be considered. Finally, we will look at resource usage.

7.1.1 Response times

In order to determine whether scenario-based scheduling really speeds things up, we are now going to analyze the response time of the direct stop. First, we need to clearly define what this response time is and how we are going to measure it. After that, we will take the measurements for both our adapted product and the original one. Finally the predictability will be considered, whether scenario-based scheduling offers more guarantees with regard to the response time.

7.1.1.1 Details of the direct stops’ response time

For the direct stop, the stopping time of the ‘process’ is most critical. The response time of the paper-input modules for example is not that important. Therefore we have identified the response time as the time between the event where the direct stop is identified and the time when the ‘process’ reports to be finished handling the direct stop. We are aware of the fact that the ‘process’ has actually stopped a little sooner; its reply was delayed by the CAN-bus. That does not matter, the response times of the original and the new situation should be considered in relation to each other. Both suffer that delay.
To determine when we hit a starting- or endpoint, we use Océ’s logging tools. At the start and the end of the scenario, we let the model make an entry in the log-file. The logging tool automatically adds a microsecond accurate timestamp to this entry.

For both the original and the adapted version, we are going to measure the response times of six direct stops. We let these stops occur when the machine is handling a print-job. The initiation of the direct stop is automated, they always occur at the same time. From these samples, we determine the minimal, maximal and average response time. We also determine the standard deviation as an indication whether the response times can be considered stable.

7.1.1.2 Measured response times

During the test runs we have recorded the following direct stop response times (in milliseconds) for the original VarioPrint 6250 and the one that uses scenario-based scheduling:

	1	**2**	**3**	**4**	**5**	**6**
Adapted	2179,675	2178,214	2181,345	2177,811	2177,595	2177,971
Original	2171,196	2172,916	2171,609	2173,241	2172,559	2172,931

From this table we can derive the values that our model from chapter 5 requires.

	Adapted	**Original**
Minimum response	2177,595 ms	2171,196 ms
Average response	2178,768 ms	2172,409 ms
Maximum response	2181,345 ms	2173,241 ms
Response's standard deviation	1,462701 ms	0,819222 ms

Below, a box plot [18] graphically represents the measured response times:

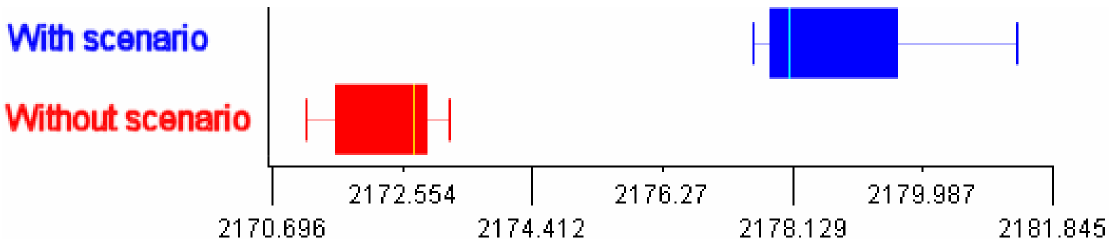


Figure 14: Box plot of the measured response times

We can see that there have not been any improvements with regard to the response time of the direct stop, things are even worse. Since reducing this response time was the main reason for using scenario-based scheduling, we can conclude that it failed for the direct stop of the VarioPrint 6250.
We need the CPU measurements of paragraph 7.1.3.2 to explain this failure, for now just assume that we have selected the wrong test-project and scenario. In this specific situation there is no scheduling problem.

7.1.1.3 Predictability of the response time

Calculating a theoretical upper bound on the response time of a direct stop is too complex for both the original and the adapted version. A huge amount of capsules are involved in the

direct stop and there are also other uncertain and uncontrollable factors; for example the CAN-bus used for communicating with other nodes and those other nodes themselves.
However applying scenario-based scheduling removes some of the uncertainties:

- In the original system all capsules that are notified of a direct stop (paper-input, engine-output, process, etc…) are executed in a random order. In the worst case, the critical ‘process’ is executed last. When using our sub-scenario for the ‘process’ we remove this random execution. We force the ‘process’ to execute its actions first.
- Furthermore, in the original implementation only the top-error-manager runs at a high priority thread. The previously mentioned capsules that are also notified are executed on the default thread. This thread also handles the normal behavior. There might be an unknown amount of messages still pending on this thread. Before the error handling capsules can execute their behavior, these messages need to be handled first. Executing the error-code on a different, high priority, thread requires that at most the transition currently being executed needs to be finished in order to satisfy run-to-completion.
- Another disadvantage of this default thread is that it runs at a relatively low priority. There are approximately 16 other threads that run on higher or equal priority and can therefore block the default thread and direct-stop handling.

All these uncertainties could have been removed without using scenario-based scheduling. However that involves a lot of complex and error-sensitive work like completely tracing down the scenario by hand and setting the priorities of all capsules involved. The amount and the complexity of the work are probably the reasons why it has not been done.
So, it is still not feasible to establish a useful upper bound on the response time and therefore create a fully predictable system. However scenario-based scheduling makes it easier, at least for the VarioPrint 6250, to build a product that has less uncertainties.

7.1.2 Overall performance

The overall performance concerns the entire system, not only the previously discussed critical tasks. To verify whether or not the overall performance is sufficient, we use Océ’s ‘Smoke-test’. This test is very broad and is therefore used by Océ to test the system as a whole. Our modified version should not fail any part of the test that the original product passes. If this is the case, we know that applying scenario-based scheduling to the direct stop does not cause other tasks to miss their deadlines.
Both the original and our modified version of the VarioPrint 6250 passed the ‘Smoke-test’ without any problems. We can therefore conclude that scenario-based scheduling did not interfere with the other tasks running on the main-node of the system.

7.1.3 Resource usage

We also need to consider the resource usage. Using scenario-based scheduling will give a certain overhead, but how much? There are two types of resources that are of interest: memory and CPU usage. For both we will analyze the effects of applying scenario-based scheduling.

7.1.3.1 Memory usage

All additional memory required for scenario-based scheduling is being allocated during start-up. This involves the memory required for the additional threads, the administration of these threads, the scenario triggering information, the hash-table used for detecting the triggers and some buffers. The amount of memory required depends on the number of scenarios being used and the settings used for the buffers and the hash-table.
In order to determine the difference in memory usage, we have used Vx-Works' 'memShow' functionality [7]. We have measured the memory usage for both systems in two states, when the system is 'ready' and during identical print-jobs. The results were not what we had expected. All tests showed that when using scenario-based scheduling, the system uses 2.679.440 bytes less than without scenario-based scheduling. The same model was used for both the modified and the original system, so the difference in memory usage should be related to the changes in the RTS. We have determined this unexpected result to be caused by the additional logging functionality Océ has included in its own customized RTS. This functionality is included, though not being used, in the original situation, but was not implemented in our modified run-time system. For these logging purposes, a buffer is constructed that contains 50.000 entries of structures containing 16 words of 4 bytes. That means that this buffer consumes 3.200.000 bytes of memory. Our scenario-based scheduling modifications therefore require an additional 520.560 bytes of memory on top of the standard run-time system.
So with regard to the memory usage, scenario-based scheduling performs quite well. It causes some memory overhead, but that was expected.

7.1.3.2 CPU usage

We have also measured the CPU usage of the VarioPrint 6250 at two points. First, we have measured it under normal circumstances. A 100-page print-job was send to both the original and the new system. WindView [8] was automatically triggered by this job to start logging CPU-usage. After the job had finished, WindView automatically stopped logging. The second point at which we have measured the CPU-usage was the direct stop. Again WindView was automatically triggered to start and stop logging. It should be noted that the system and its CPU usage will be influenced by this measurement. Resulting percentages will normally be slightly lower. While comparing the values this does not matter, both situations occur the same overhead due to the logging.

	Adapted VarioPrint 6250	**Original VarioPrint 6250**
During print-job	4,5931 %	4,2911 %
During direct stop	3,6335 %	3,2014 %

Two conclusions can be drawn from these numbers. For both scenarios it can be seen that our adapted version uses more CPU than the original one. This can be explained by the fact that the system still has to do the same calculations plus some additional checks whether or not scenarios are started. We suspect the overhead to be relative to processor usage, when more computations are being processed, there are generally more messages being send and thus the overhead increases. During the print-job, our scenario-based scheduling implementation uses $\frac{4,5931}{4,2911} = 107,0\%$ of CPU relative to the original system. During a direct stop

$\frac{3,6335}{3,2014} = 113,5\%$ of the original usage is required. These numbers are quite high. A more efficient implementation can potentially lower them, however there will always be some overhead. Possibly a simpler implementation of the hash-function can be used in future implementations. Currently it is quite expensive to compute a scenarios hash-value. Another idea is to integrate functions into the RTS that allow the user’s code to temporarily disable the expensive scenario checking.
Next, we can also see that the processor load is low in all situations. The previously discussed overheads were therefore not a problem for the VarioPrint 6250, there was quite enough of CPU still available. This low CPU usage also explains why our direct-stop response times have not improved. The direct-stop problem is not scheduling related. Most of the time the main-node is idle, waiting for messages to traverse the CAN-bus and being processed by the local nodes. Because of the low processor load, all tasks can do their desired computations quickly either way. Using a different scheduling algorithm therefore does not have much of an impact in this situation. Even under the most ideal conditions, where all required computations can be done instantly, we can reduce the direct stop response time with a mere 3,2014% at most. Scheduling cannot influence the other 96,7986% percent of the response time.

7.2 Impact on the development process

Most of the time spend on applying scenario-based scheduling to the VarioPrint 6250, was accounted to debugging our run-time system and getting to know how the existing software exactly works. Since the RTS now works fine and normally scenario-based scheduling will be applied by a developer that knows all the ins and outs of the system, this time can normally be omitted.
The following things needed to be done in order to make use of scenario-based scheduling:

1. Prepare the project for scenario-based scheduling. Select the modified RTS and adjust the makefiles to automatically execute the required scripts during a build.
2. Let the scripts generate all potential scenario starting points. For a complex system like the VarioPrint 6250, this roughly takes five seconds, so that will not be a problem at all.
3. Next, these starting points will have to be pruned. The starting points for the scenarios that you want to include need to be identified. All other scenario starting points can be removed.
4. With only the desired scenario starting points left, the second script needs to generate the scenarios. Again, this only takes a few seconds.
5. And then the difficult part. Scheduling parameters need to be assigned to the scenarios. The developer has to assign threads, priorities and pre-emption thresholds to the scenarios. A good rule of thumb, already being used at Océ, is assigning priorities according to ‘earliest deadline first’. The shorter a tasks (or scenarios) deadline, the higher its priority.
6. When all parameters are set, phase three of the RRTEI scripts can be run. This produces the additional code that needs to be included in the generated program. The time this takes is, again, only a few seconds.
7. After that, the project can be build as usual. Modifying scheduling parameters works fine with incremental builds, only one file needs to be recompiled.

8. Of course, testing is still required. It has to be verified if the timing requirements are met and whether the system functions properly. If there remain scheduling related problems, you should start over again from the fifth step.

Optionally, you can roughly calculate an upper bound on the response times of the highest priority tasks. This aids the developer in setting the scheduling parameters in the fifth step, and if done correctly and thoroughly, could reduce to number of test/adjust iterations required. Doing this for all tasks however, quickly becomes infeasible.
So if you have enough knowledge of your system and know what scheduling parameters to use, a scenario can be integrated into your system within an hour. Furthermore our implementation of scenario-based scheduling is fully backward compatible with conventional threads and priorities. If part of the system uses these conventional techniques, those remain to work fine.
Integrating scheduling scenarios into our project was easy, but it also provides other advantages over the conventional approach. In the VarioPrint 6250, there is one main capsule that is in control when a direct stop occurs. However other components also need to be notified of the direct stop and take some actions. With conventional means, you have to identify all capsules that have to do ‘something’ when a direct stop occurs. All these capsules should be incarnated on the error handling thread. This results in all these capsules always operating on this high-priority thread. Since direct-stops are supposed to be rare, this means that these capsules nearly always run at the wrong priority. Océ avoids this by not placing these capsules on the error thread. However that results in the direct stop not completely being handled at high priority. This is not a problem for scenario-based scheduling. It identifies all involved capsules by itself and, when using Vx-Works’ native priority-inversion prevention mutexes, these capsules are automatically executed at high priority only when in a direct stop situation. So with regard to the direct stops, it is much easier to apply scenario-based scheduling than conventional techniques.

7.3 Conclusion

We have seen that scenario-based scheduling did not lower the direct-stop response time as we had hoped. Océ assumed that CPU usage would be high during a direct stop and were therefore under the impression that different scheduling techniques could enhance performance. However as we have measured, processor usage is low even during a direct stop. Most of the time, the main-node is idle and waiting for the local nodes and the CAN-bus. Since the case we have tried to improve appeared to be not scheduling related, it makes sense that scenario-based scheduling did not improve the situation; we applied our solution to the wrong case.
Despite this setback, there were other things we did learn from these experiences. We have seen that scenario-based scheduling can handle large and complex projects like the VarioPrint 6250, the machine still passes all its tests. With regard to the memory usage, scenario-based scheduling is quite efficient; an additional 0,5 megabyte was required for our case. However the processor overhead is quite large and could potentially be a problem. Making the implementation more efficient can still reduce this overhead. Furthermore it was shown that scenario-based scheduling is easy to apply and that it removes some of the uncertainties with regard to the response time.

We can conclude that scenario-based scheduling did work for the VarioPrint 6250, a large and complex project. It was shown that it could handle the size and complexity involved. However we did not see any of the expected improvements. The reason for not seeing any improvements can be explained by the fact that we applied the technique to the wrong case; it appeared not to be scheduling related. It works, but the results are bad.

8 The future of scenario-based scheduling

During this thesis it was shown that scenario-based scheduling can be used for large and complex projects. The VarioPrint 6250 operated on our custom run-time system without any problems. However scenario-based scheduling failed in reducing the response time for the direct stop, what should have been its main advantage. So we have seen that scenario-based scheduling isn’t always the best solution. What should Océ and other companies do now? Should they back out of the new scheduling technique, can it be useful for different projects or should they first do some more research? This chapter is going to provide a suggestion about how to continue. We start with listing the disadvantages scenario-based scheduling has, followed by its advantages and we conclude with our advice.

8.1 Disadvantages of scenario-based scheduling

During this thesis and its test project it was shown that scenario-based scheduling still has some limitations and disadvantages. A detailed list of current limitations was already given in paragraph 4.2.2. Below we will discuss on the most problematic ones.

- The biggest problem is the CPU overhead caused by scenario-based scheduling. For our tests with the VarioPrint 6250, we have measured a relative increase in processor load of up to 13,5%. This overhead is mainly caused by the fact that for each ‘send’-action it has to be checked whether or not it initiates a scenario. Since these ‘send’-actions occur very frequently, a short delay has a huge impact. Overhead can by reduced by improving the implementation of scenario-based scheduling. The efficiency of the implementation can still be increased and other changes can be considered:
  - ❑ The hash-function can be simplified. Currently all strings used in identifying a scenario starting point have to be traversed completely in order to calculate the required hash-values. Since most of these strings (capsule name, capsule-role and port-role) are static, their hash-values can also be computed once, during their construction. This speeds things up during run-time. Furthermore, a less extensive hash-function can be considered; the computation time you save might outweigh the occasional false-hit in the hash-table.
  - ❑ Functions can be added to the RTS that allow the model to temporarily disable the checking for scenario starting points. This can be used during a direct stop for example. Overhead can be reduced almost to zero at the costs of temporarily not recognizing additional scenarios.
  - ❑ Different ways of initiating scenarios can also be considered. Instead of our flexible but expensive approach, in which each ‘send’-action has to be checked at run-time, the original static approach can be followed. In this static approach, the model is responsible for initiating the scenarios. The model needs to be modified so that it calls special RTS functions that initiate a scenario. Flexibility and ease of use will then be sacrificed for the sake of efficiency.
- As we have explained in more detail in paragraph 4.1.3.2, scenarios should not include blocking calls. These calls can block other parts of the software undesirably. It is up to the developers to ensure this condition is met. Blocking calls should traditionally be executed

on dedicated threads, but in the midst of a scenario, switching to this dedicated thread is not allowed.

8.2 Advantages of scenario-based scheduling

Next to the disadvantages we have mentioned, scenario-based scheduling also provides advantages over conventional scheduling means.

- An important thing is that scenario-based scheduling is fully backward compatible with conventional scheduling means. It does not have to be used for the entire system. If there are blocking calls or when the conventional approach is preferred, that is fine, it still works in conjunction with scenario-based scheduling. So developers are allowed to use it only when it suits them best.
- Creating and modifying scenarios is easy. Scenarios are often easier to apply than the conventional scheduling techniques. Consider for example a scenario that spreads out to several different capsules in the system, like the direct stop; there are a lot of capsules that need to be notified of its occurrence. If scenario-based scheduling is being used, only the starting point has to be identified, all subsequently triggered capsules are automatically executed on the same priority as the starting capsule. To achieve the same effect with conventional means, all subsequently triggered capsules have to be manually traced and identified. All of them need to be explicitly executed on the same thread (or at least one with same priority) as the thread on which the scenario is initiated.
- Scenario-based scheduling also allows capsules to run at multiple priorities. There are situations in which that has advantages. Consider a capsule that handles all the errors in a system. Errors can have different severity's (warnings, errors, fatal errors…). The more severe an error is, the higher the priority should be at which it is handled.

8.3 Advice

Using scenario-based scheduling has its advantages and disadvantages. Because it is backward compatible, developers can still use conventional techniques when those are better suited or when scenario-based scheduling does not work at all. Scenario-based scheduling can be considered an addition to the conventional scheduling techniques; it can be used only when it provides an advantage over normal scheduling. The only serious problem that remains is the processor overhead it causes. We therefore suggest companies to first investigate whether the overhead of scenario-based scheduling can be reduced. Some options for doing so have already been discussed. Reducing the overhead will automatically lead to reduced response times, since there is also less overhead within scenario execution.
During this thesis and the test project we have seen that scenario-based scheduling can handle large and complex systems. What we have also seen is that it does not provide advantages for products and scenarios that have more than enough processing capacity available, like the VarioPrint 6250 with its direct stop. However scenario-based scheduling does still have potential for systems that do not have an abundance of processing power available or for scenarios which are computation intensive; scenarios that do not spend most of their time waiting. It is therefore suggested for companies to do additional measurements on those

systems. They have to analyze the effect of scenario-based scheduling in those situations. In that field, scenario-based scheduling might still turn out to be useful.

So for the products and scenarios that have enough processing power available, we suggest companies not to use scenario-based scheduling; it will not improve performance. The improvements scenario-based scheduling is supposed to have are outweighed by an increased overhead. However it might prove useful for other systems. In order to determine whether that is actually so, we suggest companies to first optimize the implementation to reduce the overhead. When that succeeds, they should measure the effects of scenario-based scheduling on systems that do not have an abundance of resources available and the effects on scenarios which are computation intensive. We think that the results will be better for those systems. If they are, there is actually no reason why not to introduce scenario-based scheduling for those systems.

9 Conclusion

This thesis analyzed a new scheduling technique, based on scenarios, for automatically generated code from UML-RT models. Scenario-based scheduling only existed in theory until now and apart from some simple test models, it had not been used in practice yet. In this thesis two questions were addressed, whether or not the scheduling technique actually works for huge and complex projects and what impact it has on such projects. In this chapter we will discuss the answers we have found for both questions.
Apart from providing an implementation of scenario-based scheduling and analyzing its impact, there have also been some side effects to this research. These effects provided valuable knowledge not directly related to scenario-based scheduling. Section 9.3 briefly discusses these side effects.

9.1 Can it be used for large projects?

The first research question we have tried to answer was:

> *Is it technically feasible to integrate the new scheduling technique into Océ's software development process?*

It was comprised of sub-questions addressing scalability, limitations/restrictions to the current (experimental) implementation and what modifications could be made in order to remove some of these limitations and restrictions.
Quite a lot of limitations and restrictions to SISA, the original implementation of scenario-based scheduling, have been identified. They have been discussed in section 7.1. The remainder of chapter 7 discussed the modifications we have made to SISA. These modifications removed most of the limitations and restrictions. Furthermore the new implementation is fully backward compatible with the original scheduling approach, if there are still limitations or restriction in a specific situation, that situation can be resolved using conventional means. One major problem left is the CPU overhead caused by scenario-based scheduling.
With regard to the scalability, we did not see theoretical limitations. We therefore applied our scenario-based scheduling implementation to Océ's VarioPrint 6250, a huge and complex project in whose development 1.500 man-years have been spend. And it worked; the machine passed all its tests. Scalability therefore is not an issue anymore.
Finally the first main research question. Yes, scenario-based scheduling can be used in the development of large and complex products. Our success with the VarioPrint 6250 showed that the remaining limitations and the scalability were no problem. Furthermore, as section 6.3 showed, scenario-based scheduling was easy to apply in the development process.

9.2 Does it actually improve things?

Our second research question was:

What is the advice about using the scheduling technique in the software development process?

In other words, is scenario-based scheduling better than conventional scheduling means? We first needed to determine what we consider better. After that, both our scenario-based scheduling implementation and the original implementation had to be measured according to these criteria for 'better'. Finally we have to analyze and elaborate on these measurements in order to come to an advice.
But first the question about the definition of 'better'. In chapter 5 we have developed a model that addresses all the aspects relevant in our situation to determine what is 'better'. The model considers the efficiency of the development process used and also analyzes the product it has produced. This model can be used in the future to determine the effects of changes in the development process of real-time software.
Next, we discuss the actual assessment of both our new and the original development process and associated products. Because the original development process could not be exactly replicated, it was not possible to do a solid comparison of the development process. Our assessment is therefore mainly based on the product aspects. We applied the model we have developed to our test project with the VarioPrint 6250. It turned out that, in this case, scenario-based scheduling performed worse than the conventional approach. Response times have slightly increased and there was relatively quite a large overhead in CPU usage. On the other hand we think that there were improvements with regard to the development process, Though that was one of the things that we could not thoroughly measure. Applying our scenario was shown to be very easy, whereas doing it correctly in the original situation would have taken more effort. However, as a whole, the results of applying scenario-based scheduling to the VarioPrint 6250 are worse than they were in the original situation.
Finally we can answer the main question and advice on what to do with scenario-based scheduling. Our tests with the VarioPrint 6250 can be considered a failure. We contribute this failure to the fact that CPU usage is very low for the VarioPrint 6250 and the direct-stop scenario, therefore the effect of different scheduling techniques is minimal. So for systems and scenarios with similar characteristics, we suggest not to use scenario-based scheduling. However we can conceive of different systems were the impact of scenario-based scheduling is bigger and more positive. Systems in which resources are scarce and systems with scenarios that are computation intensive. Our advice for these systems is therefore to first explore whether the 'overhead-problem' can be reduced and then to apply our measurement model to these other systems.

9.3 Side effects

This research has provided an implementation of scenario-based scheduling that can handle large and complex projects. Furthermore the impact of this implementation has been analyzed. Apart from these main goals, this research also provided valuable knowledge not directly related to the goals of this research. The following tools were developed or additional knowledge was acquired during this research project:

- Océ still uses a Rose RT run-time system based on the original RTS from 2001. The main reason for not migrating to the RTS of 2003 was the uncertainty of whether everything would still work. Our implementation of scenario-based scheduling used the RTS from 2003. The test-project with the VarioPrint 6250 did therefore not only showed that the machine could handle the new scheduling technique, it also showed that it could handle the new RTS. Part of Océ’s uncertainty with regard to migrating to the new run-time system has therefore been removed.

- Prior to this research, Océ had no means of accurately analyzing the processor usage of machines using a configuration similar to that of the VarioPrint 6250. We developed these tools for analyzing the impact of scenario-based scheduling, but they can also be used under different circumstances. We therefore provided Océ with the means to accurately analyze the processor usage of its machines.
- We did not improve performance with regard to the direct stop of the VarioPrint 6250, however we did provide more insight into this direct stop. Before this research was conducted, it was not clear to Océ what was going on during a direct stop, they expected high processor loads. We have shown that this is not the case. The direct stop being too slow has nothing to do with the processor and its scheduling on the main node.

10 References

[1] *Java*, Sun microsystems, http://java.sun.com.

[2] *Microsoft Development Network (MSDN)*, Microsoft, http://msdn.microsoft.com.

[3] *Python*, Python Software Foundation http://www.python.org.

[4] *Rational Rose RealTime User Guide*, IBM Rational Software Corporation, 2003.

[5] *The Round Robin Scheduling Algorithm*, OSDEV Community, http://www.osdcom.info/e107_plugins/content/content.php?content.2.

[6] *Unified Modelling Language (UML)*, Object Management Group (OMG), http://www.uml.org.

[7] *VxWorks OS Libraries API Reference*, Wind River Systems, Alameda, CA, 2003.

[8] *WindView User's Guide*, Wind River Systems, Alameda, CA, 1999.

[9] V. R. Basili, *Using Measurement for Quality Control and Process Improvement*, *Second Annual SEPG Workshop*, Carnegie Mellon University, Pittsburgh, Pa, 1989.

[10] V. R. Basili and H. D. Rombach, *The TAME Project: Towards Improvement-Oriented Software Environments*, IEEE Transactions on Software Engineering, 14 (1988), pp. 758-773.

[11] D. Gaudreau and P. Freedman, *Temporal Analysis and Object-Oriented Real-Time Software Development: a Case Study with ROOM/ObjecTime, Proceedings of the 2nd IEEE Real-Time Technology and Applications Symposium (RTAS '96)*, 1996, pp. 110-118.

[12] D. Kalinsky and M. Barr, *Priority Inversion*, Embedded Systems Programming (2002), pp. 55-56.

[13] S. Kim, M. Buettner, M. Hermeling and S. Hong, *Modeling scenarios in scenario-based multithreading for Real-time object-oriented modeling, International SoC Design Conference*, 2004, pp. 358-361.

[14] S. Kim, S. Cho and S. Hong, *Automatic Implementation of Real-Time Object-Oriented Models and Schedulability Issues, Proceedings of Sixth International Workshop on Object-Oriented Real-Time Dependable Systems (WORDS '01)*, 2001.

[15] S. Kim, J. Park and S. Hong, *Scenario-Based Multitasking for Real-Time Object-Oriented Models*, Journal of Information and Software Technology (2005).

[16] M. H. Klein, J. P. Lehoczky and R. Rajkumar, *Rate-Monotonic Analysis for Real-Time Industrial Computing*, IEEE Computer, 27 (1994), pp. 24-33.

[17] C. L. Liu and J. W. Layland, *Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment*, Journal of the Association for Computing Machinery, 20 (1973), pp. 46-61.

[18] D. L. Massart, J. Smeyers-Verbeke, X. Capron and K. Schlesier, *Visual Presentation of Data by Means of Box Plots*, LC-GC Europe, 18 (2005), pp. 215-218.

[19] J. Masse, S. Kim and S. Hong, *Tool Set Implementation for Scenario-based Multithreading of UML-RT Models and Experimental Validation, Proceedings of the 9th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'03)*, 2003.

[20] Océ, *Corporate Information*, http://dl.oce.com/downloads/En/Pdf/about/worldwide2005.pdf, 2005.

[21] Océ, *Jaarverslag 2005*, Venlo, 2005.

[22] Océ, *Press release - Océ VarioPrint 6250*, Venlo, Netherlands, 2006.

[23] R. E. Park, W. B. Goethert and W. A. Florac, *Goal-Driven Software Measurement - A Guidebook*, Carnegie Mellon University, Pittsburgh, 1996.

[24] D. J. Paulish and A. D. Carleton, *Case Studies of Software-Process-Improvement Measurement*, Computer, 27 (1994), pp. 50-57.

[25] R. S. Pressman, *Software Engineering - a Practitioner's approach*, McGraw-Hill, Berkshire, England, 2000.

[26] J. M. Roche, *Software Metrics and Measurement Principles*, Software Engineering Notes, 19 (1994), pp. 77-85.

[27] H. D. Rombach and B. T. Ulery, *Improving Software Maintenance Through Measurement*, *Proceedings of the IEEE*, 1989, pp. 581-595.

[28] M. Saksena, P. Freedman and P. Rodziewicz, *Guidelines for Automated Implementation of Executable Object Oriented Models for Real-Time Embedded Control Systems*, *Proceedings of the 18th IEEE Real-Time Systems Symposium (RTSS '97)*, 1997, pp. 240-251.

[29] M. Saksena and P. Karvelas, *Designing for Schedulability Integrating Schedulability Analysis with Object-Oriented Design*, *Proceedings of the Third IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC 2000)* 2000, pp. 360-367.

[30] M. Saksena, A. Ptak, P. Freedman and P. Rodziewicz, *Schedulability analysis for automated implementations of real-time object-oriented models*, *Proceedings of the IEEE Real-Time Systems Symposium*, 1998.

[31] B. Selic, G. Gulekson and P. T. Ward, *Real-Time Object-Oriented Modeling*, John Wiley & Sons, New York, NY, 1994.

[32] B. Selic and J. Rumbaugh, *Using UML for Modeling Complex Real-Time Systems*, ObjecTime, 1998.

[33] L. Sha, R. Rajkumar and J. P. Lehoczky, *Priority inheritance protocols: An approach to real-time synchronization*, IEEE Transactions on Computers, 39 (1990), pp. 1175-1185.

[34] R. v. Solingen and E. Berghout, *Improvement by goal-oriented measurement*, *Proceedings of the European Software Engineering Process Group conference (E-SEPG)*, Amsterdam, The Netherlands, 1997.

[35] W. Stallings, *Operating Systems: Internals and Design Principles*, Prentice-Hall, New Jersey, 2001.

[36] Y. Wang and M. Saksena, *Scheduling Fixed-Priority Tasks with Preemption Threshold*, *Proceedings of Real-Time Computing Systems and Applications (RTCSA'99)*, Hongkong, 1999.

[37] E. W. Weisstein, *Standard Deviation*, MathWorld - A Wolfram Web Resource, http://mathworld.wolfram.com/StandardDeviation.html.

A. Introduction to concepts

This research is about real-time and scheduling. In this appendix we give a short description of these concepts. Readers familiar with these concepts can skip this appendix.

A.1. Real Time

Real time is a property of a certain system. That system can consist of hardware, software or a combination of both. A real-time system is a system that must oblige some timing constraints. This means that whenever something happens to the system, it has to react within certain time bounds.
Consider for example the lighting of a room. When someone pushes the light switch, the light should switch on within one second, not after one minute. And another example, a word processor. When the user types a character on his keyboard, it should appear on his screen within several milliseconds.
The printers and copiers Océ produces also have many of these timing constraints. Their interface must react immediately to user commands, users demand jobs to be finished within a certain time, etc… But also the internal hardware poses timing restrictions. When the paper gets stuck or another malfunction occurs, the machine must react quickly to prevent further damage. If it reacts too late, it could require a mechanic to come by. Until then, the machine is inoperable which costs a lot of money. Or when printing starts, getting the job into the printer's memory has a high priority, it determines when printing can start and thus influences the time printing takes. When the paper arrives in the printing section, printing should start quickly. And when the machine is turned on, it should be ready to print/scan/copy within a certain time.

A.2. Predictable scheduling

Predictable scheduling refers to software. In a computer, there are many pieces of software that must share a limited amount of hardware. There can be multiple programs that want to access the Internet over a single connection, or there can be hundreds of programs being executed on a single processor. The system must ensure that all pieces of software can use the desired hardware at a certain point, the system has to make a schedule for the hardware. It determines when some software can use a certain piece of hardware. This is called scheduling.
When scheduling is predictable, it means that some properties with regard to scheduling can be proven. It can, for example, be guaranteed that a program has to wait at most one second before it gets access to the processor. Other examples are that a certain computation requires at most ten seconds to complete, that a system can run one hundred programs without any problems. And in our case, that it can be guaranteed that all real-time requirements are met.

A.2.1. The importance

As shown above, predictable scheduling can be used to prove that a system will meet its real-time requirements. These requirements must be met to ensure the quality of the system. When the scheduling algorithm is not predictable, other ways have to be found to ensure that the real time constraints are met. This includes extensive testing, careful analysis and tuning, and over-dimensioning (e.g., using a twice as fast processor). All these additional options are either time consuming or expensive. Furthermore they only make it more likely that the constraints are met, they still offer no guarantees. Therefore predictable scheduling is the best option to ensure real-time constraints.

B. Scheduling techniques

There are several scheduling techniques available. They are used at the operating system level to determine which task can use the processor. Each technique suits its own specific needs. Some allow pre-emption, others allow priorities, etc… Most scheduling techniques have certain laws/rules associated with them, they offer some guarantees. With regard to these laws, the techniques are predictable. In this appendix, the most important ones and the ones relevant to this research, will be discussed together with their guarantees. This provides the reader with some background information about what's going on at the lowest level.
Off-line scheduling techniques are not considered here. The real-time systems Océ deals with involve a lot of event-driven, a-periodic, tasks. For such situations, off-line scheduling is not suited [16].

B.1. Round Robin

This is probably the simplest online scheduling technique, it's also very easy to implement. When a certain task wants to use the processor, it is placed in a queue. The dispatcher (program that handles the scheduling) takes the first task out of the queue and lets it run for a predefined amount of time, called a quantum. If the task does not finish its execution within that time, it will have to wait for its next 'time slice' [5, 35]. Depending on how the quantum is chosen, this scheduling technique can have low overhead. There are no complex calculations in the dispatcher; it simply takes the first element from a queue. Therefore the set of tasks can use the CPU very effectively.
The only guarantee this scheduling technique offers is that all tasks are executed eventually. They can only move forward in the queue, so after a while they will end up in front. Response time is the biggest issue of this technique. Even the most important tasks have to wait for all the other tasks in the queue. High priority tasks can be blocked by several insignificant, low priority tasks for quite a while. Because of that, 'round robin' is not suitable for handling tasks with high priorities and short deadlines; it is not suited for most of the real-time systems.

B.2. Fixed Priority, non-pre-emptive

To solve the priority issue of 'round robin', priorities can be introduced. Tasks are assigned a certain priority, a level of importance. Multiple tasks can have the same priority. Each priority level gets its own processor queue. When a task requires use of the processor, it is placed in the queue corresponding to its priority. The dispatcher first looks at the most important queue, if it is empty, the next queue will be considered. It takes the first task of the highest priority and lets it execute till completion. Complexity of this scheduling technique is still low, so the dispatcher will not cause a huge overhead. It is however important to note that, compared to round robin, more computation time is going to the dispatcher instead of to the actual tasks.
How should we assign the priorities when using this scheduling technique? Often, a good strategy is to look at the deadlines of the tasks, when we require the tasks to be finished after emerging. Priorities should be assigned based on these deadlines, the shortest deadline gets

the highest priority. Note that priorities are fixed, the deadlines of the tasks should be known in advance and the priority of the tasks will not increase when the deadline approaches.
Because of the priorities, the more urgent tasks get executed earlier. The response time of these high priority tasks will decrease significantly and they will more likely meet their deadlines. However there are still several disadvantages. Due to the fact that a task is allowed to execute till it is finished, high priority tasks can still be blocked. If a low priority task just started execution, and has a very long execution time, a high priority task that emerges has to wait for this 'very long execution time'. Therefore there still is a high risk of important tasks not meeting their deadlines. The lower priority tasks are another problem. If there are several tasks with a higher priority, the processor might not get to execute the lower priority tasks and therefore these tasks will miss their deadlines. This is called starvation. It can be solved with some modifications.
So using fixed priorities and not allowing pre-emption still results in a high risk of missing deadlines. Therefore this technique is still not very suited for complex real-time systems.

B.3. Rate Monotonic Scheduling (RMS)

To reduce the blocking times of high priority tasks, and thus lowering their response times, we should allow pre-emption. Fixed priority, pre-emptive scheduling is also referred to as 'Rate Monotonic Scheduling (RMS)'. The idea dates back to 1973 [17]. There are several articles about RMS, the articles of Klein et al. [16] and Gaddreau et al. [11] are of particular interest because they show examples of RMS being used in complex real-time systems and address the problem of blocking.
RMS still uses the priority queues discussed above. The difference is that when a tasks requests the processor and has a higher priority then the tasks currently active, the active task is interrupted and will be placed back in its queue. The new task takes control of the processor immediately and does not have to wait for the low priority task.
So the previous problem of high priority tasks missing their deadlines because of a low priority task blocking access to the CPU has been tackled. But again, starvation of low priority task can still occur and there are other ways in which a low priority task can block a high priority task. Shared resources are the cause of that. For some resources like files, mutexes, I/O devices, etc, it's important that only one task a time has access (read/write) to it. When a task has just accessed such a resource and is pre-empted by a higher priority task, this new task cannot access that same resource. It has to wait, is blocked, till the low priority task releases the resource. In some cases, even a priority inversion can occur. That's a chain of events that could block the high priority task even longer. Details about priority inversion can be found in [12, 33]. You can counter this problem by protecting the critical resources with 'Priority Inheritance' or 'Priority Ceiling' protocols [12, 33].
We should also start considering the overhead of scheduling at this point. Pre-emption requires task switching, which is expensive with regard to CPU usage. The task currently running and its complete state should be stored in memory, it should be replaced with the new task, and eventually the original task has to be loaded again to the CPU. Furthermore the usage of 'priority inversion'-prevention protocols causes extra overhead. More CPU time needs to be invested in scheduling, which leaves less for the actual tasks.
One of the neat things of RMS is that it offers some scheduling guarantees, which makes it predictable. These guarantees are expressed in a theorem called 'Rate Monotonic Analyses (RMA)' [11, 16, 17]. Using RMA, the maximum response times can be calculated for tasks

and it can therefore be guaranteed that they will not miss their deadline. However, modern real-time systems quickly get too complex to make these calculations for all tasks. The theorem also states another property that can be used to ensure schedulability:

$$\sum_{i=1}^{m}\left(C_i / T_i\right) \leq m * \left(2^{1/m} - 1\right)$$

Where *m* is the number of tasks, *Ci* is the computation time required by task *i* and *Ti* is the period of task *i*. For event-driven tasks, the minimal bound in which the event can reoccur can be considered its period *Ti*. The left part of the equation sums the processor utilization factors of all tasks. In the right part of the equation, the least upper bound is calculated for the amount of tasks. If the total processor utilization is less or equal than the least upper bound, the equation holds, then it is guaranteed that the set of tasks is schedulable using RMS. For a large amount of tasks, this means that when processor utilization is below or equal to 69%, it can be guaranteed that deadlines are met. Note that if the utilization is higher, the set of tasks might still be schedulable, however that is not guaranteed anymore by RMA.
As can be seen, there are still some drawbacks in using RMS. In most cases however, RMS is the best option. Therefore most real-time operating systems make use of it.

B.4. Fixed Priority with Pre-emption Threshold

To tackle some of the overhead introduced by RMS, a pre-emption threshold can be added to the scheduling technique [36]. The goal of this threshold is to reduce the amount of pre-emptions, and thus the amount of expensive task switches. ‘Fixed Priority scheduling with pre-emption threshold’ looks like RMS. Only now a task has two priorities. The first priority functions like the priorities discussed above. The second priority indicates the minimum priority another task should have to pre-empt this task. A low priority task can thus be set that it can be pre-empted by a high priority task, but not by a medium priority task. Using pre-emption thresholds allows task to finish, before it is being pre-empted by a slightly more important task. Using pre-emption thresholds also makes the scheduling technique more powerful than the two fixed priority techniques discussed before. If a set of task can be scheduled with one of the techniques discussed before, it can also be scheduled when using the pre-emption thresholds. The opposite is not true, there are task sets that cannot be scheduled with non-pre-emptive scheduling or RMS, but can be scheduled using pre-emption thresholds[36].
The advantage over RMS is that the overhead caused by task switching is reduced and thus more CPU can be allocated to the actual tasks. Also the technique is able to successfully schedule more task sets than the other ones discussed before. A problem with the technique is that using it becomes complex, how should the pre-emption thresholds be chosen? For a small task set this can be done by hand, but for a larger set it will not be feasible. Wang [36] offers an algorithm to derive them, however it is based on assumptions that cannot be met by a serious real-time system; blocking by shared resources is not allowed, tasks cannot be suspended and task switching causes no overhead. Therefore it will be very difficult to use the technique to its full potential. But it can be use to ‘tune’ the RMS scheduling technique; set all pre-emption thresholds equal to the priority (which is in fact RMS) and modify the ones that cause problems.

B.5. Earliest Deadline First (EDF)

‘Earliest Deadline First (EDF)’ scheduling was – similar to RMS – introduced in 1973 by Liu and Layland [17]. Theoretically, this scheduling technique offers the best performance. If a set of tasks is schedulable in any way, it is schedulable using EDF. It uses priorities and pre-emption, like RMS, but the difference is that EDF uses dynamic priorities based on the time till the actual deadline. The closer the deadline, the higher the priority. Since time progresses, the deadlines keep changing dynamically and hence also the priorities.
It can be proven that this scheduling technique is theoretically optimal [17]. However in practice it is not. A huge amount of overhead is introduced. Task priorities change and have to be recalculated each time a new task requires access to the processor. Furthermore we still have to deal with the expensive task switching associated with pre-emption. In practice, the results are worse than RMS and therefore almost all real-time operating systems use RMS instead of EDF.

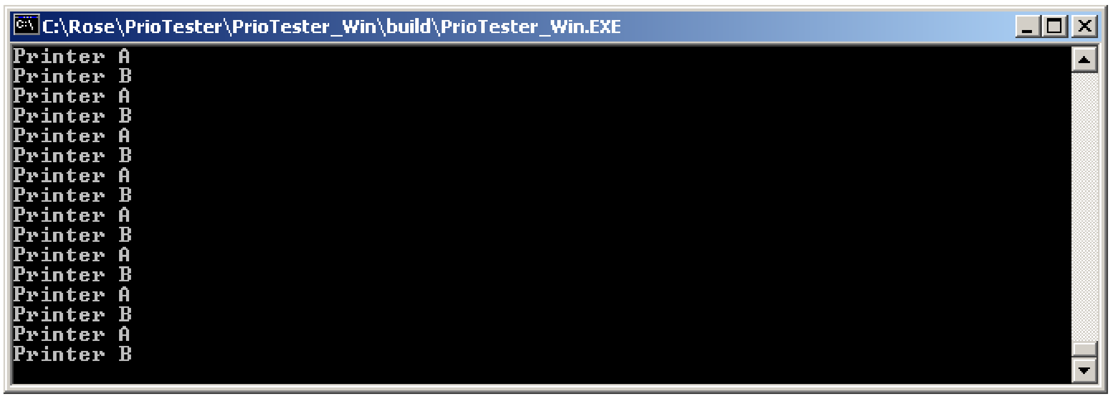
All of the discussed scheduling techniques have their advantages and disadvantages. Which one is best depends on the situation. The most common one used is ‘Rate Monotonic Scheduling’, for most systems it gives the best results.

C. A simple test model

We have used a simple test model for discovering the limitations and bugs in both the original and the new implementation of scenario-based scheduling. It was also used in testing the implementations. This model uses a wide range of the functionality offered by Rose RT to ensure that the implementation can handle them. It also needed to be very simple, you should be able to manually predict the outcomes of all actions to be able to signal inconsistencies. Furthermore it had to clearly show the effect of different scheduling techniques and parameters. This chapter will discuss that model. We start with describing the general idea. After that we take a closer look at the model.

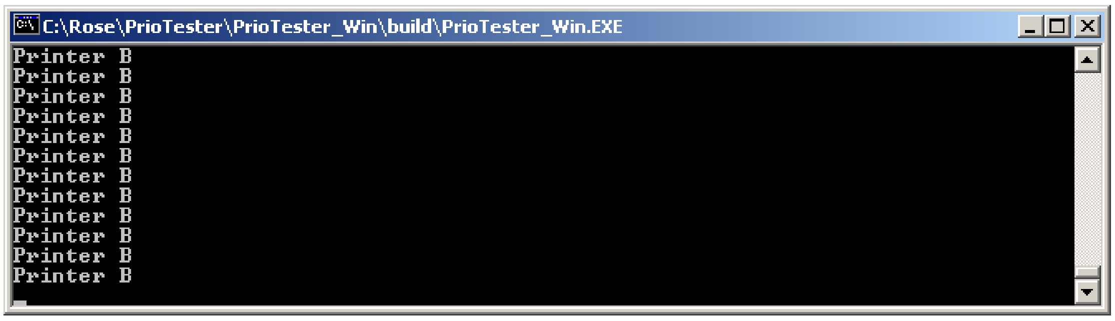
C.1. Basic idea

The basic idea of the test model is that there are two threads or scenarios. Each of them repeatedly displays a distinct text on the screen. As an observer you can now see which thread is active by looking at the messages displayed on your screen. Under normal conditions, when priorities are equal for both scenarios, you should see an equal amount of messages by each of the scenarios. When one of the scenarios runs at a higher priority, you should only see messages from that scenario.
So when we do not use scenario-based scheduling or set thread priorities, the output should look like Figure 15. When two scenarios with a different priority are used, Figure 16 should resemble the results.



```
C:\Rose\PrioTester\PrioTester_Win\build\PrioTester_Win.EXE
Printer A
Printer B
Printer A
Printer B
Printer A
Printer B
Printer A
Printer B
Printer A
Printer B
Printer A
Printer B
Printer A
Printer B
Printer A
Printer B
```

Figure 15: Test model output without priorities



```
C:\Rose\PrioTester\PrioTester_Win\build\PrioTester_Win.EXE
Printer B
Printer B
Printer B
Printer B
Printer B
Printer B
Printer B
Printer B
Printer B
Printer B
Printer B
Printer B
```

Figure 16: Priority of scenario B is increased

C.2. The model

Lets start with looking at one of the threads that print the messages. It consists of two identical capsules that print a message and then order the other one to print theirs. This goes back and forth and after a certain amount of prints they stop. One of the capsules can be seen in Figure 17.

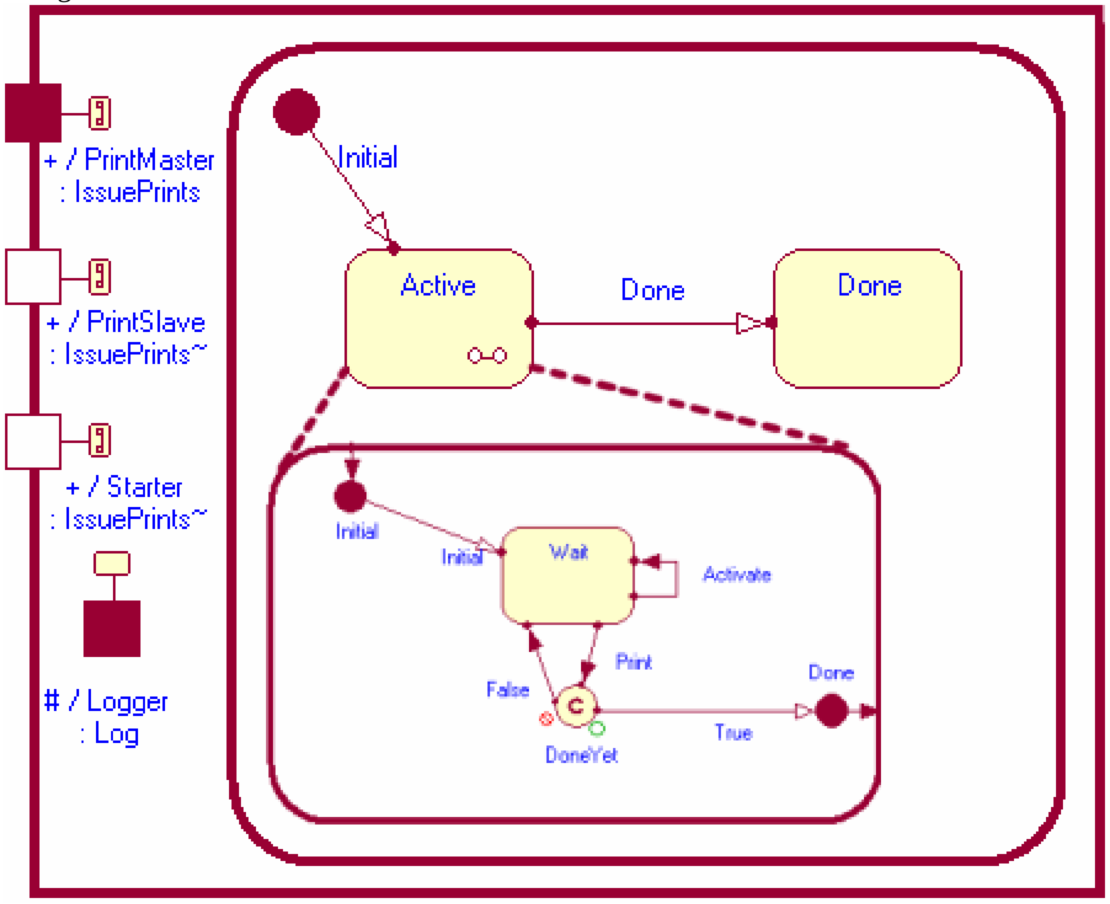


Figure 17: The printer capsule (structure and state)

How does it work? At the ‘Starter’-port a message can be send in to trigger the system. After that, a message is print and if we still have other messages to print (choicepoint) another capsule will be triggered using the ‘PrintMaster’-port. This port connects to the other capsules ‘PrintSlave’-port after which all is repeated. The results could have been achieved in multiple easier ways, but the goal was to test most of the Rose RT features. This capsule utilizes several of them. There are composite states, choicepoints, junctionpoints and there is communication between capsules.
Now lets move up a level, to the complete system. The structure of this system is depicted by Figure 18. On the left, the thread, or scenario, that prints the B’s can be seen. To start this scenario, a timer expires which sends a message over an unwired port to the ‘Starter’-port of one of the capsules. On the right, there is the A-scenario. One of its capsules is contained in a container that simply relays the messages at the ports. The other capsule is an optional one, it will be incarnated on a different thread by conventional means. Again the system is more complex then it needs to be in order to touch many of Roses’ features.

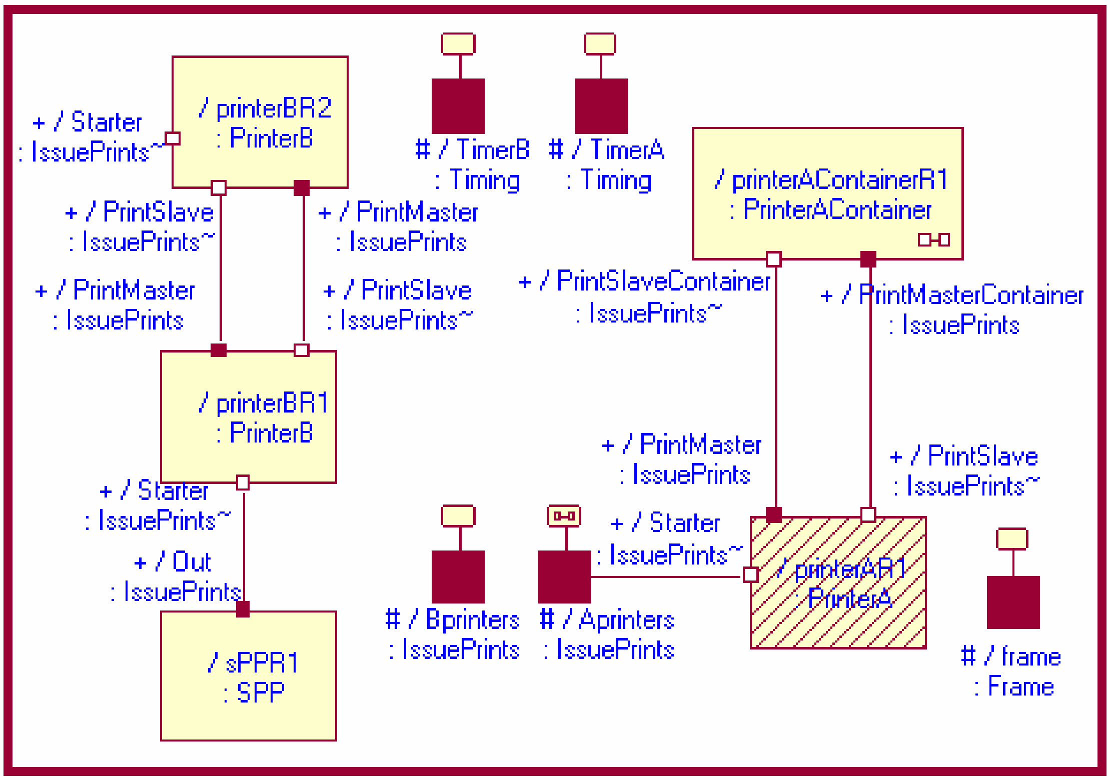


Figure 18: Overview of the model used for testing

D. Establishing the measurements

In chapter 5 we have created a model to assess the software development process and the resulting product in order to determine the effects of scenario-based scheduling. The measurement plan for that purpose was derived using Parks' guidebook [23]. This appendix documents the step-by-step approach of that book. We start with general business goals and gradually move down to the measurement goals and the measurements themselves. An overview of this process is depicted by Figure 13 in chapter 5.

D.1. Identifying Business Goals

The first step that has to be taken is to identify one or more of the principal business goals. Those are the highest level goals. In our case there is just one main goal and purpose:

Assess the real-time software development process in order to determine the impact of scenario-based scheduling.

D.2. Identifying what you want to know

Next, we need to identify what we want to know, which questions arise with regard to our business goal. Whether or not scenario-based scheduling is a good thing depends mainly on two things, the software it produces and the effect it has on the process producing it.
Three questions have been identified, the first two are related to the output of the development process; the produced real-time system, and the last one considers the actual development process.

1. *Does the quality of the produced product increases?*
2. *Are the hardware requirements of the produced product lowered?*
3. *Is the software development process more efficient?*

D.3. Identifying Sub-goals

The questions we have just created need to be grouped into related topics. These topics address aspects that need to be answered in order to reach our business goal. They are sub-goals and should be formulated as such.
We have already identified a grouping; product related and process related. The first two questions are related to the produced real-time product, the third one is about the development process. That results in the following sub-goals:

1. *Asses the quality of the produced real-time software*
2. *Asses the performance of the real-time-software development process*

Of these two goals, the first one is most important. The main concern for Océ is to improve the produced product. However doing that may not have too much negative impact on its development process. Therefore the priority lies with first goal and the measurements derived from it.

D.4. Identifying Entities and Attributes

In the fourth phase we need to define the entities of interest, related to the sub-goals. We should also identify their attributes. In short, what elements need to be studied in order to come to our sub-goals?
The following entity/attribute sheets were constructed for our case:

Sub-goal:	Asses the quality of the produced real-time software
Question: Entity: Attributes:	Does the quality of the produced product increases? The produced real-time software • Critical tasks (with their response times) • All tasks (with their deadlines)

Almost all literature on real-time software acknowledges the attributes ‘response time’ and ‘missing of deadlines’ as the most important ones. In many articles you also come across ‘blocking time’ as another important attribute. We will leave that one out; the blocking time is also part of the response time. In our case, it is sufficient to only know the response time.

Sub-goal:	Asses the quality of the produced real-time software
Question: Entity: Attributes:	Are the hardware requirements of the produced product lowered? Hardware for the produced real-time software • CPU usage • Memory usage

Sub-goal:	Asses the performance of the real-time-software development
Question: Entity: Attributes:	Is the software development process more efficient? The software development process for real-time software • Man-hours • Testing resources (BBO/EPT)

The BBO and EPT are machines used by Océ for testing purposes. A BBO is a simulator for the machine Océ is developing; it simulates the hardware (sensors and actuators). BBO’s can be used for testing the software without the risk of damaging hardware when it fails. EPT’s are prototypes of the machine under development.

D.5. Formalizing Measurement Goals

The prior steps were an addition to the GQM approach [9, 10, 27]. By executing those steps, we are now in a situation were we can effectively construct the measurement goals, the first phase of GQM. Furthermore, these prior steps provide the link between measurement goals and business goal. The measurement goals have to be well structured; they require an entity, a purpose, a perspective and a description of the environment and constraints [23].
For our case we have the following measurement goals:

Object #1:	**Critical tasks in the produced real-time software**
Purpose:	Analyze the critical tasks in order to assess the quality of the produced real-time software
Perspective:	Examine the response times from the point of view of the developer
Environment:	Océ, Real-time software, scheduling, software development, Rose RT, Vx-Works, scenario-based scheduling, RMA, RMS, pre-emption thresholds

Object #2:	**All tasks in the produced real-time software**
Purpose:	Analyze all the tasks in order to assess the quality of the produced real-time software
Perspective:	Evaluate whether all tasks meet their deadlines from the point of view of the developer
Environment:	Océ, Real-time software, scheduling, software development, Rose RT, Vx-Works, scenario-based scheduling, requirements

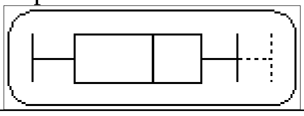
Object #3:	**Hardware requirements of the produced real-time software**
Purpose:	Analyze the hardware requirements in order to assess the quality of the produced real-time software
Perspective:	Examine the hardware usage from the point of view of the developer
Environment:	Océ, Real-time software, scheduling, software development, Rose RT, Vx-Works, scenario-based scheduling, hardware, processors, memory

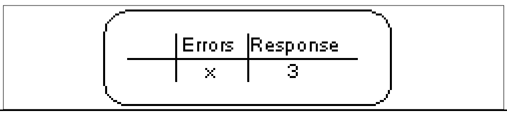
Object #4:	**Man-hours spend on the real-time software development process**
Purpose:	Analyze the invested man-hours in order to assess the efficiency of the real-time software development process
Perspective:	Examine the invested man-hours from the point of view of the project manager
Environment:	Océ, Real-time software, scheduling, software development, Rose RT, Vx-Works, scenario-based scheduling, testing

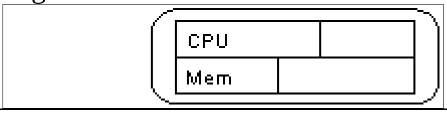
Object #5:	**Testing resources spend on the real-time software development process**
Purpose:	Analyze the testing resources spend in order to assess the efficiency of the real-time software development process
Perspective:	Examine the amount of invested testing resources from the point of view of the project manager
Environment:	Océ, Real-time software, scheduling, software development, Rose RT, Vx-Works, scenario-based scheduling, testing, BBO, EPT

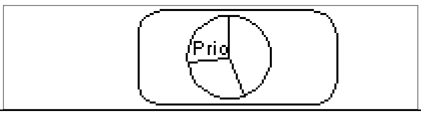
D.6. Identifying Quantifiable Questions and Indicators

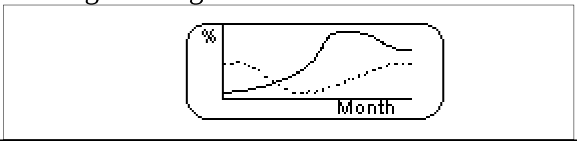
After establishing the measurement goals, we move on to the second phase of GQM. We have to pose quantifiable questions that arise based on these measurement goals. Furthermore, we should also formulate indicators, visualizations of the answers to the questions. These indicators are not part of the original GQM approach, but will aid in finding which measurements that are required.
In our case, there will be several questions and one indicator for each of the discussed measurement goal.

Measurement goal:	**#1: Critical tasks in the produced real-time software**
Questions:	1. For all critical tasks, what is the maximum response time encountered? 2. For all critical tasks, what is the average response time encountered? 3. For all critical tasks, what is the minimum response time encountered? 4. For all critical tasks, what is the theoretical upper bound on the response time? 5. For all critical tasks, are response times stable? What is the standard deviation of the response times [37]?
Indicator:	For each critical task, there will be a box-plot [18] that displays information about the response times. 

Measurement goal:	**#2: All tasks in the produced real-time software**
Questions:	1. Are deadlines, specified by the requirements, being missed? 2. Do errors with regard to scheduling (i.e. buffer overflows) occur?
Indicator:	A table will be used that shows whether or not the failures described above occur.  Errors Response x 3

Measurement goal:	**#3: Hardware requirements of the produced real-time software**
Questions:	1. What is the CPU usage of the hardware on which the produced real-time software is running? 2. What is the memory usage of the hardware on which the produced real-time software is running? 3. For all critical tasks, what is the maximum response time encountered? (i.e. are we already operating at our limits?)
Indicator:	An indicator will be used that shows for each kind of resource the percent that is being used. 

Measurement goal:	**#4: Man-hours spend on the real-time software development process**
Questions:	1. What is the total amount of man-hours spend on the entire project? 2. What is the amount of man-hours spend on the scheduling related aspects?
Indicator:	The indicator that we are going to use is a pie chart. It shows how much time is spend on scheduling, related to the total number of hours spend on the project. 

Measurement goal:	**#5: Testing resources spend on the real-time software development process**
Questions:	1. What is the amount of hours spend testing on a BBO? 2. What is the amount of hours spend testing on an EPT?
Indicator:	A graph describing the usage of both resources in time. 

D.7. Identifying Data Elements

In phase 7 we are going to identify the data elements that are required to create the previously discussed indicators. The following data elements and their relationships with the indicators have been identified.

	Indicator				
Data required	**#1**	**#2**	**#3**	**#4**	**#5**
Response times of tasks	X				
Theoretical upper bounds on tasks response time	X				
Missed deadlines and scheduling related errors		X			
CPU usage			X		
Memory usage			X		
Total number of man-hours				X	
Number of man-hours related to thread and priority allocation				X	
Testing hours on a BBO related to scheduling					X
Testing hours on an EPT related to scheduling					X

D.8. Defining Measures

The data elements discussed in D.7 are too vague and ambiguous to be real measurements. Therefore we are now going to construct a more precise definition of the data elements which can be measured and should not be open to different interpretations.

- **Response times of tasks:** The response time of a task, in our case that of a scenario, is the time that has elapsed between the point were the trigger was send that initiated the task and the time at which the task finished its execution. In the response time is therefore included the execution time, the blocking time before the task is allowed to start and the blocking that can occur during execution (due to higher priority tasks or shared resources). This is the commonly used definition of response time [15, 17, 19, 28, 29, 30, 36].
- **Theoretical upper bound on tasks response time:** Response time is defined above. The upper bound is the maximum possible value for this response time, based on theoretical analysis. Normal, continuous, operation of the system should be assumed (i.e. the system does not crash etc…). Theory about RMA [11, 16, 17] and pre-emption threshold scheduling [36] can be used in this analysis.
- **Missed deadlines and scheduling related errors:** To verify whether or not the rest of the system meets its deadlines and does not contain scheduling related errors, it suffices to use Océ's standard tests. Any aspect of the test that fails needs to be recorded.
- **CPU usage:** Average percentage of CPU usage measured over a period of time in which the system is active.
- **Memory usage:** The numbers of bytes of memory allocated at a certain point in time.
- **Total man-hours:** The total amount of man-hours booked for the development of the system. All hours related to the development process are counted; developing the architecture, programming, testing, etc…

- **Number of man-hours related to thread and priority allocation:** The number of man-hours that can reasonably be accounted to the scheduling related actions for the project. Included is the time spend on the original design of the thread and priority allocation, modifications made to these parameters during the project either by the designers or developers, implementation of thread and priority related things, the testing for schedulability and the analysis of the results from these tests.
- **Testing hours on a BBO related to scheduling:** The number of hours is counted when someone is using a BBO for testing whether or not the schedule holds. If during a non-scheduling related test errors with regard to scheduling occur, it will also be counted. Someone is using the BBO when no one else can use the BBO, so setting up the BBO for the test is also considered testing here.
- **Testing hours on an EPT related to scheduling:** For EPT-hours goes the same as for the BBO.

D.9. Analysis, Diagnosis and Action

By now we have clear measures. In the current phase we are going to identify sources for acquiring the desired data. Whenever possible, we will try to use existing information sources. That will not work for all measures, in those cases we will define new information collection methods. When defining these methods of data collection, we keep Roche's [26] advice in mind to prefer the use of automated collection methods.

- **Response times of tasks:** There is currently no source readily available from which this information can be derived. We should measure it ourselves. This data collection can be fully automated. Logging should be added to the produced software, when the triggering action is executed, the time should be logged and when the scenario finishes, time should also be logged. A script can then be used to analyze these logs and extract the required information.
- **Theoretical upper bound on tasks response time:** These upper bounds will have to be manually calculated based on RMA [11, 16, 17] and pre-emption threshold [36] theory. For complex projects, like almost all projects at Océ, this will probably only be feasible for the highest priority tasks. However that was our domain, the critical tasks. Therefore it is important not to mark too much tasks as critical. The information that is required for these calculations resides in the UML-RT model and the scenario parameters.
- **Missed deadlines and scheduling related errors:** Océ's standard tests will be used. These tests should be run on the product with its final scheduling configuration, all occurring scheduling related errors have to be recorded.
- **CPU usage:** Both real-time operating systems used by Océ, WinNT and Vx-Works, provide functionality to measure the CPU usage. For WinNT there is the 'task manager' and Vx-Works has WindView [8]. Using one of these tools allows automated gathering of the required data.
- **Memory usage:** The information required here can also be collected using the 'task manager' or WindView.

So far the product-related measurements. All these measurements are based on the final product. During this research, we will redo an existing Océ project with the use of scenario-

based scheduling. Both the original and the new project produce such a final product. Therefore a solid comparison of both products will be possible.
The process-related measurements, which we come to next, will not be that easy to compare. Typical software projects at Océ take months to develop by large teams. It is not feasible to redo such an entire project. Therefore we will only redo the thread and priority related aspects of that development process. Original UML-RT models and transition code will be reused. The original development process and the new one differ significantly, for example, the original one is based on an incremental development strategy and the new process cannot be. Furthermore it is not possible to determine the time spend on scheduling afterwards. It is too hard to identify which test where related to scheduling. Implementing and adjusting scheduling parameters is often done when working on another related task and are therefore not neatly documented. Because of these facts, we cannot do a founded comparison on the software development processes. We will however continue on developing the measurements so that Océ can use them to do a thorough research themselves if desired. The process measurements for the test project will also be collected as far as possible to provide a rough indication of the effects of scenario-based scheduling on the process.

- **Total man-hours:** Océ has a database, TOCK, which contains the required data. The database administrates the amount of man-hours spend on certain projects for all their personnel. Information with regard to the total amount of man-hours spend on a project should be extracted from this database.
- **Number of man-hours related to thread and priority allocation:** The existing TOCK database can also be used for recording the data to measure. For the project currently under development a separate entry should be added for the scheduling related tasks. Personnel should specify the hours worked on threads and priorities here. Afterwards, the total number of hours spend on scheduling can be extracted from this database.
- **Testing hours on a BBO related to scheduling:** To use a BBO, a reservation has to be made. There is a reservation sheet for that purpose. This sheet can be extended with a field that states the purpose of using the BBO. If this sheet is filled in correctly, the amount of hours related to the testing of scheduling properties can easily be determined.
- **Testing hours on an EPT related to scheduling:** For EPT's goes the same as for the BBO. Separate reservation sheets are maintained for them.

D.10. Preparing Your Measurement Plan

In the last phase, an action plan that implements the defined measures has to be produced. Our action plan can be found in section 5.2 of this thesis.