Using Analytical Models of Complex Real-Time Systems for Temporal Impact Analysis

Anders Wall
Mälardalen University
Västerås, Sweden
awl@mdh.se

Joakim Fröberg
Volvo Construction Equipment
Eskilstuna, Sweden
joakim.froberg@volvo.com

Christer Norström
ABB Robotics
Västerås, Sweden
christer.e.norstrom@se.abb.com

Abstract

To predict the temporal impact of adding new functionality to large and complex real-time systems becomes more difficult the older the system gets. In this paper we describe the concept of analytical models and how we can use the analytical model for predicting the temporal impact of adding new or changing existing functions. We define tree levels of abstractions in the analytical model, requirements, components, and implementation. The concept has been applied on two large and complex industrial embedded real-time systems: a robot control system and a vehicle control system. The case studies unveil two different approaches to the construction, and maintenance of the analytical model of a system, "by construction" or "by re-engineering". Moreover, we show that both static analytical methods and simulation-based methods are applicable when analyzing a real-time system. It is the characteristics of the system and the correctness criterion that determines the most appropriate method.

1. Introduction

Large and complex computer systems usually have long lifetime and history. Thus, they are exposed to many changes found necessary during maintenance and due to the introduction of new features. Further, software in general has a longer lifetime than the hardware (HW), and thus the software will during its lifetime run on different HW platforms and different operating systems. Predicting the impact on the system behavior that new features will impose becomes increasingly difficult as the complexity of the system grows. This does not only depend on the pure system complexity, it also depends on the fact that many engineers have been involved in the development of the system and many of these persons may have continued their careers. Newly employed engineer may have great difficulties to understand a certain function in the system without a proper system model. The software itself provides poor support for such analyses of a system. Consequently, additional models of the system must be constructed that provides the information necessary for analyzing and predicting a system's behavior. As our focus is concerned with real-time systems, we are interested in the system's temporal behavior, i.e. is the system temporal correct. A real-time system is considered temporal correct if all its temporal requirements are fulfilled, e.g. deadlines, latency. We refer to a system model that provides information to make temporal analysis feasible as an *analytical model*.

The introduction of an analytical model of a system permits early predictions when adding new features or changing existing features in the system. The earlier the analysis is performed the more coarse-grained is the result. Nevertheless, early predictions are essential as they indicate whether the development is going in the right direction. Discovering that the system's behavior is incorrect in a late phase of the development is often very costly. It is important to continuously analyze the system as more information emerges when moving towards the implementation. We propose an analytical model consisting of three different levels of abstractions, the *requirements view*, the *component view*, and the *implementation view*. Naturally, we have more detailed information about an existing implementation than a new feature that shall be implemented and integrated in the system. Thus, different level of abstractions is necessary. The contents of our analytical model are primarily suitable for real-time systems, i.e. the model is concerned with temporal behavior and resource utilization.

As the system evolves, it is important that also the analytical model is updated to accurately reflect the system. The method outlined in this paper suggests a continuous maintenance of the analytical model as well as continuously feeding information back from the implementation view, via the component view, to the feature view. This will ensure a consistent high-level description of the systems architecture.

In this paper we describe the concept of analytical models and how we can use the analytical model for predicting the impact of adding new or changing existing features. The concept has been applied on two large and complex industrial embedded real-time systems, a vehicle control system for large construction equipments at Volvo Construction Equipment in Sweden and a robot control system at ABB Robotics also in Sweden. The case studies unveil two different approaches to the construction, maintenance and analysis of the analytical model of the system. We refer the two different approaches to constructing an analytical model as "*by construction"* or "*by re-engineering"*. If a system is constructed and maintained on the basis of the analytical system model, we say that the model is constructed by construction, this being the case for the Volvo system. In the constructive approach,

the model is a product of the development effort. On the other hand, if the models are constructed and populated by measuring an already existing implementation, it is constructed by re-engineering, this being the case with the Robotics system. We will elaborate further on the pros and cons of the different approaches in the following.

The two systems do not only differ in terms of how their analytical models are constructed. The correctness criteria and, consequently, the methods for analyzing the systems also vary. Traditional real-time analyses have a strong focus on explicit stated temporal requirements. In other systems, the temporal requirements are implicit defined, such as a specific message queue is never allowed to be empty which is the case in the robotic system we have studied.

In [7], impact analysis on real-time control systems is presented. Their approach analyzes the impact of changing a software component with respect to its input/output data ranges. The result from such an analysis is a set of affected components in the system. Li et. al. also propose a system model based on different level of abstractions. Other has also proposed methods for impact analysis [14][5]. However, these not consider impact on temporal behavior. The notion of analytical- and constructive models was introduced in [6]. They utilize analytical interfaces on components for predicting properties of component assemblies. There is only one level of abstraction in their model, which thus corresponds with our component view.

The contribution of our paper with respect to related work is a more expressive temporal model and methods for analyzing the temporal impact of adding new functionality to a system. We have demonstrated our methods in two large and complex real-time systems with satisfactory results. From the traditional real-time theories point of view we also contribute with results from modeling a system not built for explicit temporal analysis, including a modeling language and an analysis tool.

The outline of this paper is as follows: Section 2 discusses analytical models and defines the architectural views formally. Moreover, different approaches to analyzing a system based on an analytical model are presented. Section 3 presents the two case studies. In Section 4 are the results from the case studies compared. Finally, Section 5 concludes the paper.

2. Analytical models

In this section we describe the concept of analytical models and present a brief discussion concerning system analysis using those analytical models. A software architecture consists of a constructive model and an analytical model. The constructive model describes how components are interconnected through their constructive interfaces, i.e. control-, and data flow whereas the analytical model provides information required for analyzing certain properties of an architecture. For instance, in order to verify the temporal correctness of a real-time system with periodic tasks, the frequency, and the worst-case execution time (wcet), of the services are required in the analytical model.

2.1 Architectural views

We divide the system model into three different levels of abstractions: the *requirements*, the *component view*, and the *implementation view*. Each of the different views provides means for analyses and verification. Analyses are performed on an intra-view basis only, i.e. information from two different views is never utilized in the same analysis. Consequently, analyses performed based on information from the component view are more coarse-grained than on the implementation level of abstraction. However, verification of the implementation will eventually be verified with respect to the requirements. Thus, verifying that the correct system, i.e. according to the requirements, has been implemented is done based on information from the implementation view and the requirements view.

In Figure 1 is the different views depicted along with a workflow and brief descriptions of the activities performed in the process. Note that the evolutionary development may add new resources and new features may share components with other already existing features.

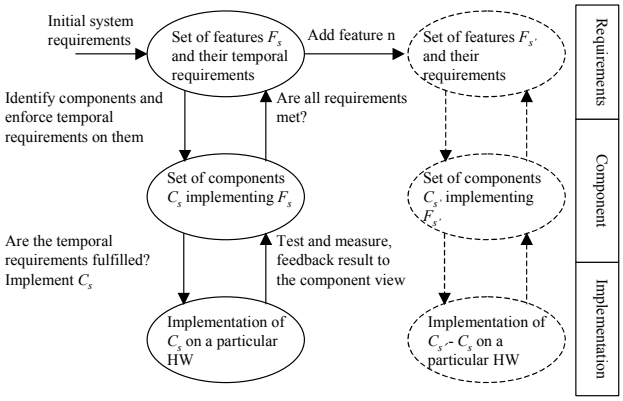

Figure 1 The different level of abstractions in the analytical model

Ideally, system construction starts at the requirement level of abstractions. The requirements are represented as a set of *features*. Each feature represents functional- as well as non-functional requirements. Hence, a feature defines a function's *explicit*- and *implicit temporal requirements*. An explicit temporal requirement is a clearly expressed requirement such as an end-to-end deadline. An implicit temporal requirement is derived from, e.g. a functional requirement or the controlled environment as for example an accuracy requirement for a robot must be translated to timing requirements for the involved services and thus the accuracy requirement is an implicit timing requirement.

As the requirements have been identified, the component view can be populated. Finally the system will be implemented; hence the implementation view is populated.

Until the implementation exists, all models are based on estimates, even though these estimates are based on experiences from other similar systems.

As the systems we have been studying are large and complex, they exhibit long service lives. Consequently, the cost of the initial development only contributes with a small fraction of the total cost. Maintenance is by far the major part of the cost as we include in maintenance such activities as error corrections, improving existing features, as well as implementing completely new features. Thus, the models proposed in this paper will also be utilized when evidence from an existing system already exists. The models are kept consistent with the implementation through feedback from testing and measuring the system.

In order to formalize the component view and the implementation view we first define a *system*. Our view of a system is depicted in Figure 2. The system formalization described is based on the result from studying two different existing systems. Hence, the definitions provide abstractions that are suitable for describing and comparing these systems. Basically, we say that a system consists of interconnected nodes.

A node n is a tuple $\langle cpu, m_s, m_d, m_p, F, I/O \rangle$, where

- *cpu* is the computational resource of the node,
- m_s is the static non-volatile memory,
- m_d is the dynamic memory,
- m_p is the persistent memory,
- F is the set of features whose components $C(F)$ is partially or completely allocated to the node n. Moreover, feature $f \in F$ is a function as experienced by system users, and it collects the functional-, and non-functional requirements.
- I/O is a set, possible empty, consisting of I/O-units.

Definition 1. A system $S \subseteq Node \times Bus$, where *Node* is the set of nodes in the system, *Bus* is the set of communication buses in case of a distributed system. □

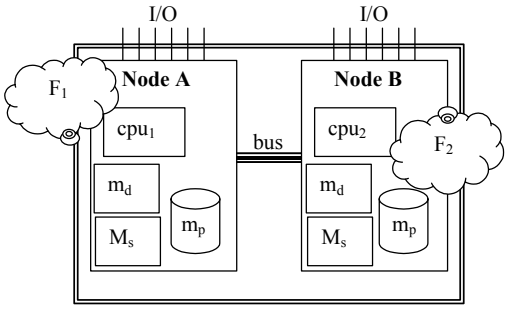

Figure 2 Our view of a system

The component view identifies the components in the system and their analytical model. We assume a component to be an encapsulation of a service implemented in software. The encapsulated service can be utilized through the components interfaces. For a more exhaustive description of our component concept we refer to [17]. An essential part of a components analytical model in real-time systems is the temporal attributes, e.g. period times. We will refer to temporal attributes in the component view as the *temporal analytical model,* which we will hereafter refer to as the analytical model. In scheduling theory, such an analytical model is referred to as a task model. This level of abstraction is suitable for analytical system analyses and for simulations of the system. Thus, a system's temporal requirements that initially were partitioned into features are implemented and verified in the component view. The appearance of the temporal analytical model corresponds with that of the real-time operating system (RTOS), and communication mechanisms in the system infrastructure. As the functional requirements are implemented in the chosen programming language, and non-functional requirements are implemented in the architecture, is the temporal requirements implemented by assigning temporal attribute in the task model provided by the RTOS. An example of such temporal attributes is period time and priorities. Consequently, the temporal attributes provided by the infrastructure are also part of the implementation view.

Definition 2. The *component view* of a real-time system is a set C_s of tuples $\langle c, ET(node), \tau_{analytic}, F \rangle$, where c is a component which is part of the implementation of the set of features F, $ET(node)$ is the execution time for a component given a particular hardware architecture specified by *node*, and $\tau_{analytic}$ is the temporal analytical model of the component □

The implementation view consists of the actual implementation. Thus, it provides necessary information for testing the system, as well as measuring execution times and response times.

When a real-time system is implemented on a particular hardware architecture, the model becomes valid for that particular instance only. The reason for this is that the temporal behavior of the components is affected when the hardware architecture is changed. Typically, the execution times will be affected. This phenomenon is typical for real-time systems and is a problem for large and complex systems that have a long lifetime.

As the temporal requirements are implemented through the task model provided by the RTOS, we consider it a part of the implementation view. However, task models in the RTOS may differ from the temporal analytical model. For instance, a task model in a RTOS may include priorities, period times, references to entry function, error handling, etc, while the analytical model may have deadlines, period times, etc.

Definition 3. The *implementation view* of a real-time system is a set I_s of tuples $\langle imp, ET(node, imp), \tau_{infrastructure}, c \rangle$, where *imp* is the implementation of component c in a programming language, $ET(node, imp)$ is the execution time given a hardware architecture *node*, and the implementation, *imp,* on that hardware, $\tau_{infrastructure}$ is the

task model provided by the RTOS, and *c* is the component implemented by *imp*.

The mapping of components onto features and implementation onto components are explicitly expressed in Definition 2 and Definition 3. The inverse relationship is also valid, i.e. features are implemented by a set of components which are implemented in some programming language.

Until the implementation view exists, the model is constructed based on estimates and interpretations of the system's requirements. Feedback from the implementation view is required in order to make the model, on every level of abstraction consistent with the implementation. The data that must be measured in the implementation are typically execution times of the components.

So far in this section we have discussed the ideal case of system development, i.e. the system is constructed through the model. However, if no analytical model exists, the model presented in this paper can be populated in a reverse-engineering manner starting by measuring the existing system. The absence of a correct system model could be due to, for instance, no model was ever constructed, or the initial model was never updated as the system evolved. The reverse-engineering activity starts at the bottom of the initial iteration depicted in Figure 1. Measured data is used for populating the component view and eventually also the requirements view.

Adding new features to the system is equivalent to specifying a new system with the distinction that the pre-existing system imposes restrictions on the new features in terms of available system resources. A rough analysis of whether the new feature indeed can be added is possible based on the existing component view and a decomposition of the added feature into components and their analytical models. It is also possible to perform such an analysis based on estimations of the resource utilization needed of the new feature and comparing that with the available resource capacity. The more details available, the more accurate are the system analyses. The scenario where new features are added is one of the main reasons for maintaining the model as the system evolves and keep it consistent through feedback from the implementation.

2.2 Analyses based on the analytical model

The main objective for introducing an analytical model of a system is to make possible early analyses of the system's behavior. In this paper we focus on the temporal behavior of systems.

We determine whether or not a system is temporal correct on the basis of the analytical model. To be temporal correct means that all temporal requirements are indeed fulfilled by the system. We will discuss two different approaches to analysis of the temporal behavior: the *analytical approach* and *simulation*. The two case studies presented in Section 3 illustrate the two different types of analyses. Existing analytical methods determines if or not the temporal behavior of a system is safe, given that the analytical model is correct, e.g. that the estimates of the wcet of the components are safe [4][3][8]. Such a method, however, tends to over-constrain the system as the worst-case always is considered. In Figure 3 are a system's temporal behavior depicted as sets. An analytical approach is pessimistic but safe, and simulation is realistic but not necessarily safe. Analytical models and analyses found in conventional scheduling theories are often too simple and therefore a real system cannot always be modeled and analyzed using such methods. Simulation is better from that point of view. By simulating the system with realistic distributions of the execution times, we can demonstrate that the system is correct. A disadvantage is that given the same correct analytical model, we cannot be confidant of finding the worst possible temporal behavior through simulation.

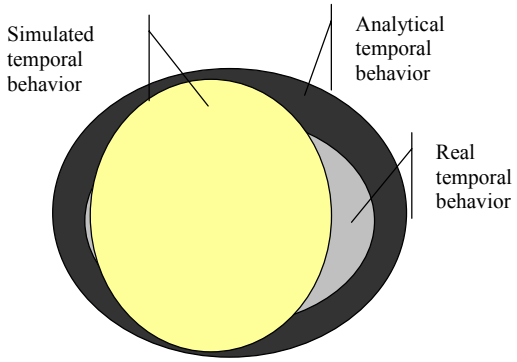

Figure 3 The confidence in analytical system analysis vs. simulation-based analysis

Yet another reason for using simulation is that the temporal correctness of some systems can be expressed in criterion different from meeting deadlines as proposed by traditional real-time analyses. An example of such criteria is non-empty message queues as discussed in Section 3. If the message queue never becomes empty, the system is considered temporally correct. The analytical approach as well as the simulation approach will be further discussed in conjunction with the two case studies.

3. Case studies

3.1 A Robotic control system

We have investigated a robotic control system at ABB Robotics initially designed in the beginning of the 90's. In essence, the controller is object-oriented and consists of approximately 2 500 000 LOC divided on 400-500 components organized in 15 subsystems. The system contains three closely connected nodes, a main node that generates the path to follow, the axis node, which controls each axis of the robot, and finally the I/O node, which interacts with external sensors and actuators. In this work we have studied a critical part in the main node with respect to control. Maintaining such a complex system requires careful analyses to be performed prior to adding new features or redesigning parts of the system to not introduce

unnecessary complexity and thereby increasing both the development and maintenance cost.

The robot control system investigated was not initially designed primarily to support temporal analysis; the focus was instead on a flexible, open and extendible architecture. For example, the system was designed to support easy porting to new HW-architectures and to be open to permit customers to add advance functionality. The temporal behavior of the system could, however, be predicted because of its simple structure. As the system evolved, more and more functionality was added and when a new feature is added to the system today, the temporal behavior is very difficult to predict. This depends not only on the system complexity itself but also on the fact that many of the engineers involved in the development of the system have continued their careers elsewhere and their knowledge is no longer available.

In studying the existing system we can identify a component view and the current implementation. In addition, the component view has no analytical model. We must therefore construct an analytical model by taking the re-engineering approach.

Several existing task models were considered but none satisfied the requirements we stipulated. For instance, traditional fixed priority analysis assumes static execution times for each task in the system [3]. In our case this assumption is too pessimistic. Furthermore, the RTOS VxWorks provides support for priorities on messages, which no existing analysis method supports, and therefore an analytical approach is inadequate. Some of the components analyzed have large variations in their execution time, and always assuming the worst possible case would give an over-utilization of the resources. Moreover, as opposed to traditional analytical models where the correctness criteria is defined in terms of meeting deadlines we also had to consider correctness criteria such as a specific message queue not being allowed to be empty.

We also studied different simulation tools such as STRESS [2], and DRTSS [15] but could use none of these because none of them provide support for execution time expressed as distributions. They, particularly STRESS, have however influenced our modeling language. The analysis we use is based on simulation since no analytical method, available today, can solve our problems.

When the modeling language was developed we created a model of the controller that was populated with execution times from measurements from a running system. The system we took measurements was considered temporal correct.

As mentioned above, there were wide variations in execution times for some components. The variations in execution times/response times are due to specific dependencies between components. Identifying those semantic relations and introducing them into the analytical model gave a more accurate model. Known dependencies between components reduce the amount of simulation necessary, i.e. the state space could be reduced.

When applying simulation techniques it is always important to decide carefully how long the system is to be simulated. Existing real-time theory often assumes cycle-times equal to a system's *least common multiple* (LCM) of the tasks period times. The basic idea is that the system reaches the same state after an LCM execution. To take aperiodic tasks and interrupts into consideration, we must run the LCM several times with different scenarios from aperiodic tasks. However, in this case, a "system cycle" is defined by the robot application and how long time it takes before a robot repeats the same movement again. Typical cycle times for a robot application are in the range from a few seconds to a couple of minutes. The cycle times varies between robot products and between robot applications.

The temporal behavior of a specific robot is dependent on the programmed behavior and the specific application type, e.g. arc welding, spot welding. To be able to build realistic models we have measured data for different types of application programs for arc welding but the plan also includes measurement of data for spot-welding and assembly applications.

The execution times were measured in order to determine their distributions. Moreover, the response times and message queue sizes was measured in order to validate our models and simulations. The measurement was obtained by recording required data at every task-switch through software probes in the source code. A tool was developed that extract information regarding execution times reported by the software probe and to calculate the execution time distributions accordingly [1].

3.1.1. Analytical task model. A language designated *ART-ML* (Architecture and Real-Time behavior Modeling Language) was developed to specify an analytical model [1]. The proposed analytical model supports multiple processors, inter-process communication, synchronization and fixed priority scheduling. Furthermore, the modeling language supports several different ways of activating components. Periodic and aperiodic terminating tasks can be described, as well as non-terminating tasks. It is possible to set a task to explicitly to be activated by an incoming message, and it is possible to assign a task a probability of activation. Such a task is activated by a certain probability every *x*'th time-unit in the simulation. A component's behavior can be modeled on various levels of abstraction from resource utilization to execution time distributions.

The construction of specific interest in ART-ML is *chance-statement* and *execute-statement.* The chance-statement functions in the same way as an if-statement, but instead of evaluating an expression, it can define the probability of executing a specific statement and thus adding the ability of modeling stochastic behaviors of a component. For instance, it is possible to specify that there is a 19% chance of sending a message:

```
chance(19)
   send(mbox1, msg)
```

The execute-statement consumes time in accordance with a distribution specified as one or more pairs of probability and execution time. In the example below, the execute statement will execute 10 time units with the probability of 19 % and 56 time units with the probability of 81 %:
`execute((19,10), (81, 56));`

The execute-statement is the only instruction which affects the simulation clock. It sets the clock to the time of the next event and thereby advances the simulation.

A rough model of a typical component in a control system can be constructed from three statements. First a "receive" that waits for a message, next one or several "execute" that consumes time and finally, a "send" statement that sends some data to another mailbox. An example of a specification of an analytical model can be studied in example 1. We can see that a component has one part for static attributes such as task type, priority and deadline, and behavioral part for specifying the temporal behavior of the task. This specification can be run in the simulator for specified time and a log of vital events with respect to execution and, for example, message queue sizes, is created.

Example 1: Specification of a toy system.

```
system
  processor MC
  mailbox mbox1 20;                    } c

  task tt_task_1
      trigger period 3000              }
      priority 0                       } τ_analytic
      deadline 3000                    }

  behaviour
      execute( (50,200),(50,250) );    } ET(MC)
      send(mbox1,2);

  task mailbox_task
      trigger startup
      priority 5
      deadline 10000
  behaviour
      variable incoming;

      while(1)
        execute ((100,70));
        recv(incoming, mbox1);
        execute ((50,1120),(50,1250));
endproc
endsys
```

The simulator is based on a discrete-event approach. The ART-ML model is compiled into an assembly language that executes on a virtual machine. A scheduler that is part of the simulator controls the execution of tasks. All events generated are logged. An event contains the event type, an identification of the component that generated the event and a time stamp. All statistical processing is performed off-line. For more information concerning the simulator and off-line tool see [1].

3.1.2. The robot model. We have modeled some critical components for the concrete robot system in the main node (see Figure 5). The main node generates the motor references and brake signals required by the axis computer. The axis node sends requests to the main node every 4'th millisecond and expects a reply in the form of motor references. This depends on three components: A, B and C. The tasks of two of these components, B and C have high priority, are periodic, and runs frequently. A executes mostly in the beginning of each robot movement and has lower priority. The final processing of the motor references is performed by C. C sends the references to the axis node. Moreover, C is dependent on data produced by B. If the queue between them becomes empty, C cannot deliver any references to the axis node. This state is considered as a critical system state and the robot halts. Component A sends data to B when a movement of the robot is requested. If the queue between A and B gets empty, the robot movement stops. In this state, B sends default references to C. The complete model is shown in [1]. All comments have been removed and variable names have been changed for business secrecy reasons. The model is not complete with respect to all components in the system. All components, other than A, B and C, have been grouped into two dummy tasks. One of the two dummy tasks has higher priority than A, and the other has lower priority than A. This is one way in which we can utilize different level of abstractions in our model.

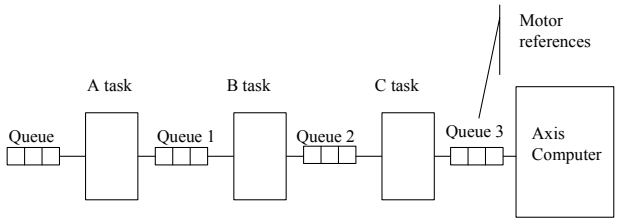

Figure 4 The task structure of the critical control part of the system

3.1.3. Simulation results. Considering the coarseness of the model of the system, the results are really promising. We have measured a system and, based on these measurements, built an analytical model on a much higher level of abstraction. We have compressed approximately 2 500 000 lines of code into two pages. While studying the system we observed that customization of each individual had great influence on temporal behavior of the system. The execution times and cycle-times, i.e. the time it takes for a robot to complete its task once, were particularly subject to variations.

The validity of the model we built and ran in our simulator was verified by comparing the output from the simulator with response times measured on the real system. The graphs in Figure 6 show the response time distribution both measured (Figure 6.a), and simulated (Figure 6.b) for a critical task in the system. We can see that the correlation

is quite close since there is two bands of response times, one approximately 12 ms and one approximately 1 ms. The real code, however, gives more variations in the response time, this being quite natural since we cannot capture all details of the code in the model. In particular, implicit dependencies between components in the system are not described in the model. Since the model will be refined during its lifetime, we expect a continuous improvement of the model as the system evolves. Several simulation and measurement results can be found in [1].

We can analyze, on the basis of this model, the consequences of adding a new feature to the controller or analyzing the effect of a redesign with respect to the critical control part. For example, assuming that we add one more component to the system, we can check how response times of the existing components are affected by the new component and, by using an automated tool, can present the results of the analysis to the designer in various representations such as graphs or average response times. This is related to the sensitivity analysis provided by e.g. Punnekkat et.al [PUN97], Yerrabali et.al[18], and Vestal [16] for fixed priority scheduling. These try to analyze the effects of an increase in the worst-case execution time on the schedulability. These techniques, however, are developed for systems on which it is possible to perform static analysis. Preliminary tests indicate that our approach is useful and adequate [1].

3.2 A construction equipment vehicle control system

The Volvo Construction Equipment AB (VCE) case study is an on-vehicle control system for heavy construction equipment. A construction equipment vehicle is equipped with a distributed computer system which controls and monitors gearbox, engine, brakes, hydraulics, etc. The systems are installed in volume products and the resources provided are therefore often limited. Typically, the system and its features have stringent requirements with respect to safety, reliability, and temporal behavior.

VCE has introduced an architecture which includes an analytical model for improved management of temporal behavior in development and in maintenance. The objective of the model and method introduced is to support the development of a high level design, including specification of properties such as temporal constraints, communication and synchronization. The model and method also support formal verification of these properties, early system integration, and permit regression testing.

As opposed to the Robotics case, the VCE case includes an analytical model as an integrated part of the development effort. Specifically, the temporal behavior of the system is addressed in the analytical model. Specification of high-level properties is an integrated activity in constructing the system and formal verification is automated in the build process. This is done to ensure that the analytical model is kept updated and that there is no discrepancy between design and implementation. Hence, the system is constructed largely using the *constructive approach* i.e. the system is constructed and maintained via the analytical model

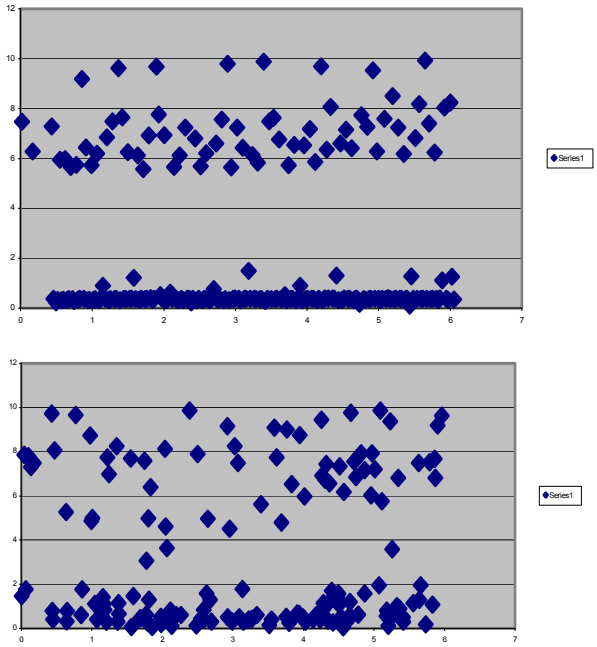

Figure 5 a) Simulated response times b) meassured response times

System features are obtained as sets of software components that together implement the feature. In the VCE case, the component is specified through its data interfaces, and the encapsulated functionality. It specifies no temporal behavior except for the execution time inherited from the function it encapsulates. A component instance defines the task that controls the execution of the component in addition to its configuration of communication, synchronization, and temporal behavior. VCE uses the Rubus RTOS which supports hard- and soft real-time tasks [12]. A tool named Configuration Compiler (CC), supports formal verification of the requirements.

The requirements are specified with a design language which supports the communication, synchronization, and temporal constructs described earlier. The formally described design can be checked for temporal correctness by using CC even if no actual implementation has been produced. The CC maps the design description to a resource structure. The CC is a pre-run-time scheduler which generates dispatch tables for running the tasks and the communication infrastructure for the system. This constitutes an application skeleton for the running and communication of the components. In addition to the mapping of the model, CC also supports specification of architecture-specific attributes such as HW-performance, resolution of the run-time dispatcher, communication times, and the number of nested preemptions permitted. The

implementation of the CC is based on a heuristic tree search strategy, similar to the one presented in [11]. The major difference is that this scheduler takes into account interrupts, preemption and architecture-specific attributes [13]. Soft real-time tasks are scheduled on-line and use the processor's spare capacity to execute.

3.2.1. Analytical model. In this section we describe the VCE analytical model and how the levels of abstraction introduced in Section 2 relate to VCE system development and maintenance. In VCE, the electronic system requirements are described in terms of abstract functions with no notion of hardware. When designing a new system, hardware is developed to achieve a high utilization given the system features. This process is based on experience from similar systems and therefore, in an early phase of a development project including a new hardware, the utilization estimates are approximate at best. The more frequent case, however, is to expand or change an existing system. In this case, the hardware has been chosen, and the CPU utilization is known through the analytical model.

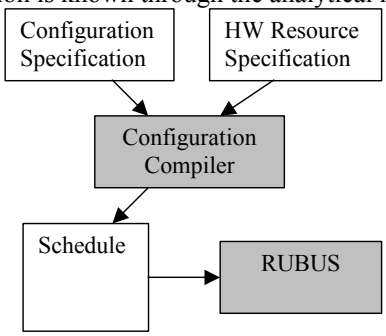

Figure 6 Verifying temporal requirements

In the development of volume VCE products, the utilization and estimates of resources are kept low to keep product cost down. There can be some slack to accommodate product variations, but generally very little slack for future features. Instead, when new features are wanted, the decision will be either to improve node hardware with more resources or to add another node to the system.

When a feature is added to the existing system, the model provides CPU utilization and estimations can be made on the new features resource utilization. Normally, these estimations can be sufficient to predict whether or not a redesign of the hardware is needed. An indirect benefit from the model is that stakeholders are provided with an estimate of whether or not the feature can be introduced on existing hardware. Decisions related to resource slack to accommodate product variation are also considered. VCE products are often developed to support configurable functionality. This could mean that some configurations use less resource than others and this is of course not optimal with respect to hardware cost (and thereby also product cost). Reuse of functionality and hardware in different products could lead to less than optimum resource usage. Business trade-offs must be made on product cost vs. development cost, and time-to-market issues. The analytical model provides utilization figures that help in this process.

Moving down to the component view, the feature is broken down to components. The analytical model is populated with components and their temporal requirements.

Consider adding a fuel level feature to an existing system. Components to implement the feature could be a sampler, a filter, and two actuator components. Temporal-, communication-, and synchronization requirements are input to the model. The VCE method involves estimating wcet for components that are not yet implemented. Estimates are based on experience from earlier systems and this has proven to work well. The system can then be analyzed for temporal correctness before the actual implementation. Existing components do already have their temporal requirements in the model. The analytical model provides automatic feasibility testing of the complete set of components. If the added components and their temporal requirements require an excessive usage of resources, the model will show the system to be unfeasible. The requirements can then be re-negotiated or resources can be added to the system. It is important in this connection that the temporal properties are formally verified before implementation. When all features have been broken down to components and the temporal properties have been negotiated and proved feasible by the model, the components can be implemented. Note the separation of high-level system properties from actual implementation. The estimated wcet:s are used as requirements for the implementation. The component must not exceed its time-budget when implemented. Given that the time-budgets are not violated during implementation, the system will fulfill its temporal requirements. If the time-budget for a component proves to be difficult to achieve, adjustments must be made in the component view of the model.

In order to verify that the wcet:s do not exceed the estimated time budgets, the wcet:s are measured during the testing phase. The measurements are used to fine-tune the time budgets in the model so components have an equal margin between the actual wcet and the time-budget.

Components communicate through *communication ports* which are defined by configuration language constructs. This make allowance for separation of communication from the implementation code.

Example 2 – High-level specification language

```
PORT aPortType {
CTYPE = "yyy";
TYPE = COMPLEX or SCALAR;
SIZE = number of bytes;
USAGE = CONSTANT or SIGNAL;
}
```

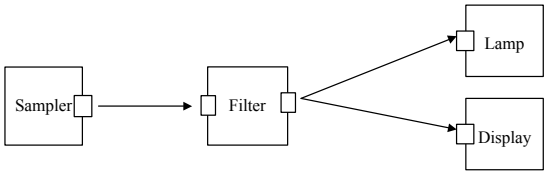

Figure 7 Fuel meter components

A component specifies the general structure of the software components. It is general in the sense that it is not yet instantiated in a real environment. For instance, if a particular period time is required in every instance of a component, this is specified in the component description. The reason for that could be related to control performance. Otherwise, each instance of a component is assigned its own task, i.e. its own temporal behavior. The execution time is related to the encapsulated function. Hence, it is always specified in the general component description.

```
// Definition of a Component
COMPONENT BrakeWarning { ⊢ c
  aPortType OilLevel;
  AnotherPortType Warning;

  OilLevel.DIRECTION = IN;
  Warning.DIRECTION = OUT;

  ENTRY = aCfunction;
  EXECUTIONTIME = 200; ⊢ ET(ndoe)
  PREEMPTION = DISABLED; }
```

The task definition specifies the run time instantiation of the component. The Component can be instantiated in several places in a system or in different systems. The task definition defines the run time specifics, like release time, and how a components communication ports are connected to other components in its environment.

```
//Definition of a Task -.
TASK Brakewarning {
  aPortType OilLevel;
  AnotherPortType Warning;

  Warning = "WARNING_NO";

  Warning.DIRECTION = OUT;
  OilLevel.DIRECTION = IN;

  PERIODTIME  = 100000;
  RELEASETIME =  30000;
  DEADLINE    = 100000;

  AnotherTask:::outputSignal->
BrakeWarning::OilLevel;
  BrakeWarning::Warning->
AthirdTask::inputSignal;}
```

3.2.2. Benefits and limitations. The model used allows for specification of temporal requirements such as deadline, period time, wcet, as well as synchronization-, and communication- behavior. Execution time budgets are estimated and the system is tested for temporal correctness before implementation. A tool is used to verify the temporal correctness given the specified temporal properties and temporal requirements of each component (cmp. $\tau_{analytic}$ in definition 2). If therefore, the system is temporal correct, the temporal requirements will hold during run-time. Temporally, the system is thus proved-by construction given that the execution-time estimates are valid. This means that almost no resource slack is needed in terms of CPU.

One positive effect of using this model and method observed by VCE is the infrequence of timing problems and the ease with which such are solved. The model ensures effective control of system's temporal behavior and of resource usage. This indirectly helps in predicting both project time consumption and provides a basis for communication with stakeholders. The analytical model provides updated measurements on CPU utilization resulting from changes in a system during its service life.

4. Reflections from the case studies

In this paper we have studied two industrial control systems which were originally designed with different objectives. The robot controller was designed to be open, flexible and easy to port and the Volvo system was designed to be easy to develop, maintain, port and formally analyzable.

The price Volvo had to pay when requiring an analyzable system is that it is difficult to add software that do not completely conform with the existing architecture. However, this may also be a benefit since resistance in the architecture itself will ensure that the initial design objectives implemented in the architecture is kept intact. In the robotics case the architecture is much easier to violate since only architectural guidelines exists. Moreover, there is no tool that checks that the architectural guidelines are followed.

The Volvo system inherently supports temporal analysis and a structured way to do impact analysis. Furthermore, since it is required when generating the system, the analytical model is guaranteed to always be consistent with the system. It is impossible to add functionality without also changing the model.

In the robotics case, we encountered classical problems such as discrepancy between the design documents and the implementation. The only true document we have for the system is the source code. Creating an analytical model as we did was quite a cumbersome task but the results we achieved, in addition to a model which describes the system in two pages, included a considerable increase in the knowledge of the system due to the re-engineering activity. The analytical model we developed is quit coarse in comparison with the precise Volvo model. Although the analytical model of the robot system can be used for impact analysis, it cannot be used for proving the temporal correctness of the system. However, by constructing a temporal analytical model of the complete system we can, with the assumptions discussed in Section 2.2, deem whether the system is correct or not from a temporal point of view. Note that the re-engineering approach is a continuous activity. To use this approach successfully the

analytical model must be maintained during the life cycle of the system.

From the perspective of temporal analyses, the Volvo approach is superior but from the flexibility point of view the Robotics approach has advantages. We can conclude that there exist a fundamental tradeoff between analyzability and flexibility. The more open and flexible the system is, the less analyzable it becomes and vice versa.

As the architectural requirements differ we cannot say that either of the approaches, i.e. analytical models by construction or analytical models by re-engineering, is preferable. We can only conclude that both approaches are valid for different type of systems. Nor, as it depends on several parameters such as the correctness criterion, and the system's characteristics in terms of its temporal behavior, is the question of using analytical methods versus simulations easily answered.

5. Conclusions

In this paper we have proposed an analytical model suitable for large and complex real-time systems, which was utilized for impact analysis on the temporal behavior. According to the proposed model, a real-time system can be described in three different level of abstraction: the requirement view, the component view, and the implementation view. The appropriateness of the proposed methods has been validated by two different case studies. As a result of these case studies we found that the model can be populated and maintained using two different approaches - either it is built by construction or by re-engineering. The result from the case studies is that the method is indeed applicable to large real-world systems.

We have also proposed two different approaches to analysis of temporal behavior and for impact analysis, analytical methods and simulation. Also when it comes to analysis method we cannot claim that any method is preferable. Suitability depends on several parameters such as the information available in the analytical model, the correctness criteria, and the system characteristics in terms of its temporal behavior. The two case studies demonstrates quite clearly that the fundamental architectural requirements will influence the choice of which approach to use for constructing an analytical model as well as what analytical method to use.

Future work includes continuing the construction of the analytical model for the robotic control system. The analytical temporal model of the robotic system made as part of this case study is by no means complete. Moreover, we will extend ART-ML to encompass also complete product lines. Finally, the proposed methodology should be integrated into a development process.

6. References

[1] J. Andersson and J. Neander. Timing analysis of a robot controller. Master thesis at Mälardalen University August 2002.

[2] N.C. Audsley, A. Burns, M.F. Richardson, and A.J. Wellings. STRESS: A Simulator for Hard Real-Time Systems. Software-Practive and Experience, 24(6):534,564, 1994.

[3] N. C. Audsley and A. Burns and R. I. Davis and K. W. Tindell and and A. J. Wellings, Fixed priority pre-emptive scheduling: An historical perspective,Real-Time Systems Journal, vol 8, no 2/3, 1995

[4] G. C. Buttazzo, Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications, Kluwer Academic Publisher, ISBN 0-7923-9994-3, 1997

[5] A. Cimitile, A.R. Fasolino, and G. Visaggio, A Software Model for Impact Analysis: a Validation Experiment, In Proceedings of 6th Working Conference on Reverse Engineering, Atlanta, USA, 1999

[6] S. A.Hissam, G. A. Moreno, J. Stafford, and K. C. Wallnau, Packaging Predictable Assembly with Prediction-Enabled Component Technology, report Technical report CMU/SEI-2001-TR-024 ESC-TR-2001-024, 2001.

[7] J. Li, and P.H. Feiler, Impact Analysis in Real-time Control Systems, In Proceedings of IEEE International Conference on Software Maintenance, Oxford, UK, 1999

[8] C. L. Liu and J. W. Layland, Scheduling Algorithms for Multiprogramming in hard-real-time environment, JACM, vol 20, no 1, pp 46--61, 1973

[9] M. Lindgren, H. Hansson, C. Norström, and S. Punnekkat. Deriving Reliability Estimates of Distributed Real-Time Systems. In Proceedings of IEEE Real-Time Computing Systems and Applications. Cheju Island, South Korea, 2000.

[10] S. Punnekkat, R. Davis, and A. Burns. Sensitivity Analysis of Real-Time Task Sets. Asian Computing Science Conference, Kahlmandu December 1997.

[11] K. Ramamritham. Allocation and Scheduling of Complex Periodic Tasks. In 10th Int. Conf. on Distributed Computing Systems, pages 108-115, 1990.

[12] Articus Systems, Rubus OS - Reference Manual, Stockholm, Sweden 2002

[13] K. Sandström, C. Eriksson and G. Fohler. Handling Interrupts with Static Scheduling in an Automotive Vehicle Control System. In the 5th Int. Conf, on Real-Time Computing Systems and Applications, Hiroshima, Japan 1998.

[14] H. M. Sneed, Impact Analysis of Maintenance Tasks for a Distributed Object-Oriented System, In Proceedings of IEEE Int. Conf. on Software Maintenance, Florence, Italy, 2001

[15] M.F. Storch and J.W. - S. Liu. DRTSS: a simulation framework for complex real-time systems. In Proceedings of the 2nd IEEE Real-Time Technology and Applications symposium (RTAS '96). Dept. of Comput. Sci., Illinois Univ., Urbana, IL, USA, 1996.

[16] S. Vestal. Fixed Priority Sensitivity Analysis for Linear Compute Time Models. IEEE Transaction on Software Engineering, April 1994.

[17] A. Wall, and C. Norström, A Component Model for Embedded Real-Time Software product-Lines, In *Proceedings of 4th IFAC conference on Fieldbus Systems and their Applications*, Nancy, France, 2001.

[18] R.Yerraballi et.al. Issues in Schedulability Analysis of Real-Time Systems. Proceedings of Seventh Euromicro Workshop on Real-Time Systems, June 1995.