Increasing maintainability in complex industrial real-time systems by employing a non-intrusive method

Christer Norström[1], Anders Wall[2], Johan Andersson[1] and Kristian Sandström[2]

[1] ABB Robotics, Västerås, Sweden
Christer.Norstrom@mdh.se
[2] Mälardalen University, Department of Computer Science and Engineering,
Västerås, Sweden

Abstract. It is well known that systems that have been evolving over a long period of time become costly to maintain, i.e., the productivity when adding new functionality and fixing bugs is low. In this paper we present a non-intrusive method for increasing the maintainability of complex industrial real-time systems. This method aims to create a formal model of an existing system and using that model for analyzing the effect of changes. The challenge of this approach is to develop a valid model with a minimum of complexity and to show that the model is valid for use. The method focuses on analyzing timing and synchronization properties.

1 Introduction

In this paper we give an introduction to a non-intrusive method for increasing the maintenance of complex industrial real-time systems, such as process control systems, industrial robot control systems, automotive systems, and tele communication systems. By complex industrial real-time systems we mean systems that have a high and increasing maintenance cost. These systems typically

- have been in operation for at least some years,
- have evolved considerably since their first release,
- were initially developed by a small group of people that understood the system in detail.
- have a staff today were the major part was not involved in the first development of the system,
- require a substantial amount of maintenance,
- have no formal model or formal analysis.

It is well known that systems that have been evolving over a long period of time become costly to maintain, i.e., the productivity when adding new functionality and fixing bugs is low. The basic reasons for that are

- the increased system complexity that depends on that the system has grown (more functionality has been added) and it has become almost impossible to understand the full impact of a change for any single person,
- the architecture has degenerated and that the original architects (key persons) that really understood the design rationale behind the system have left the company or got new challenges. The architecture degeneration depends on that the system most often has been extended with features that was not foreseen when the system was initially designed. When such a feature is added it has to be forced in to the system, i.e., a lot of design rules are broken. Further, the degeneration also depends on the misuse of the system design principles due to time pressure or bad competence or bad documentation.

The systems we consider are normally built from a limited set of subsystems. Each subsystem is quite well known and under control. The major problem is to get control of the behavior of the complete system and especially to predict the behavior at design time, since a faulty design that is detected late is much more costly to correct compared to if it had been detected at an early stage. The basic approach to handle this problem from a technical[1] point of view is to introduce a formal model, which can be used to analyze dynamic system properties such as timing, synchronization and communication at an early phase of a project.

A formal model can be used for design, analysis, test and code generation, but also for system education, which we will not elaborate more on in this paper.

We see in principle two different approaches to increase productivity:

1. Intrusive methods which aims to change the code or redesign the complete system to increase the maintenance.
2. Non-intrusive methods which introduce analyzability without changing the code.

The intrusive methods are risky in the meaning that we will change the code and they will require a much higher investment in time. An illustration of the risk is for example during code improvement projects (called quality improvement projects) that aims to fix a lot of bugs new bugs are introduced. The statistics from industrial systems tell us that for each five bugs that are corrected a new one is introduced. A non-intrusive method does not change the code and consequently there is no risk of introducing new bugs. Therefore, we will consider the most common case: a system that can neither be re-designed and nor be remodeled due to cost and too high risk of introducing new bugs. The approach we present in this paper is to reintroduce analyzability by developing a model of the particular system, which is later used for

[1] In this paper we do not consider process, documentation and management related improvements for increasing the productivity.

impact analysis of a change (maintenance operation). However, the outcome of this method is often input to a redesign activity of the code.

Our approach is based on a framework for initially verifying dynamic properties of the system such as timing and synchronization. The framework includes:

- A SW architecture model. This model describes both the static structure of the system, i.e. tasks, message queues, semaphores, etc. and the behavior of each task in the system on an abstract level.
- A HW architecture model. This model describes the used HW architecture, processors, I/O and communication links.
- Requirements property definitions. All requirements of interest are described as formulas.
- An analysis method. The analysis method verifies that the stated properties are fulfilled by the modeled system. The analysis can be either mathematical or simulation based, or both.

In this short paper we will present one instance of this framework and particularly the method and the model validation problem of using this method. As mention earlier we focus on dynamic properties such as timing and synchronization. This instance of the framework was developed for system that utilize priority based real-time operating systems such as VxWorks provided by WindRiver. We will briefly describe the modeling language, the analysis technique and the query language, full presentations can be found in [1]. The approach has been evaluated on an industrial robot control systems containing about 2500 kloc, presented in [1].

The modeling language is called ART-ML (Architecture and Real Time behavior Modeling Language). ART-ML allows the designer to model both the static structure and the dynamic behavior of the system and then use a simulator to analyze the model. The static structure includes such entities as tasks, message queues, and semaphores. The dynamic behavior of tasks can be modeled on any level of abstraction, in the most abstract case only the execution-time distribution and priority is used, but very detailed descriptions of the task behavior is also possible. An ART-ML model can also include semantic relations between tasks, i.e. when a specific behavior in one task leads to a specific set of behaviors in another task.

We have decided to initially focus on temporal properties of a system to grasp the dynamic behavior of the system. Thus, temporal requirements are specified in a query language PPL, the *probabilistic requirement property language*. The objective of this language is to provide the possibility to state the requirements that are of interest to be checked when the system is changed. The language supports specification of hard and soft real-time requirements, task execution order issues, message queues requirements, and complex compound expressions. For example we can state that a specific tasks response time on incoming messages shall be less than 10 time units in 75% of the cases.

The analysis shall verify that the requirements are fulfilled. In our current approach we use simulation for analyzing that the requirements are fulfilled. The basic reason

for that was that the current model and analysis available could not be used for the complex system we have studied. The current analysis is basically not expressive enough, i.e. can not express the semantics of the studied applications in a meaningful way. For example using Fixed Priority Scheduling (FPS) [2] on the robot control system studied will only yield that the system is unschedulable, but the system works properly in reality. The basic reason for not using FPS is that the semantic dependencies such as if one task executes its worst-case execution time another task will not cannot be expressed. ART-ML supports the description of such dependencies.

The rest of the paper is organized as follows. Section 2 presents the method of creating a valid model and Section 3 concludes the paper.

2 The method of creating a valid model

The introduction of an analyzable model of a system brings a continuous activity of maintaining the model. The model has to be consistent with the current implementation of the system, i.e. the implementation should be a true refinement of the model. Consequently, our method must be integrated in the company's development process. We will briefly describe the activities associated with creating a formal model. Figure 1 depicts the general activities required in our method.

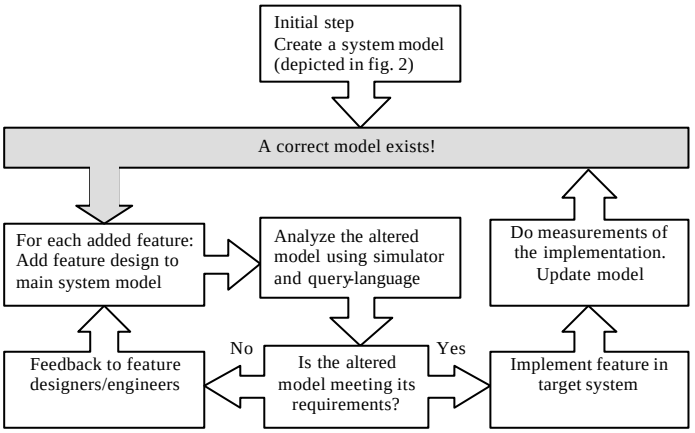

Fig. 1. The process of constructing and maintaining an analyzable system.

Note that the process described here only concerns the method we are proposing. Important activities such as verification and validation of the implementation are omitted. The first activity in making an existing system analyzable with respect to its temporal behavior is the re-engineering of the system. Typically, the re-engineering activity includes identifying the structure of the system, measuring data such as execution times and populating the model. By comparing the result from analyzing the system using the formal model with the temporal behavior of the real system

confidence in the model can be established. This is exact the same procedure as used in developing models for any kind of systems. As the system evolves, each new feature should be modeled and the impact of adding it to the existing system should be analyzed. This enables early analysis, i.e. before actually integrating the new feature into the system. Note, that this approach requires a modeling language that support models on different level of abstractions. ART-ML has this property. Modeling of new features should be part of the company's design phase. Finally, when the new feature has been implemented and integrated into the system we need to feed back information from the implementation into the model so it can be refined. Hence, a more precise model is implemented. This activity is typically performed in conjunction with the verification phase of a company's development process.

2.1 The model creation process

When creating an initial model, M_0 of an existing system S_0, several distinct activities which are depicted in Figure 2 are required.

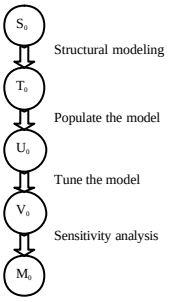

Fig. 2. The model creation process

First, the structure has to be identified and modeled, i.e. the tasks in the system and synchronization and communication among them. In the next step we measure the system and populate the structural model with data about the temporal behavior. Moreover, information needed in the validation phase is collected e.g. response times. When tuning the model we simulate the initial model and compare the results with the validation data collected in the previous step. In this step we may have to introduce more details about the tasks behavior in order to capture the systems behavior more accurately. There is a potential risk that we cannot model the systems behavior without introducing too many details. For instance when there are so many implicit relations among the tasks that we cannot make a valid model without modeling the complete behavior of the tasks involved. This, however, unveils the complexity of the existing architecture. Consequently, the solution is rather to redesign the complex architecture. Up until this point, the work of making a model is quite straightforward. To validate the usefulness of the model we have to perform a sensitivity analysis. The sensitivity analysis should be based on foreseen potential changes in the particular system. In the systems we have studied the following typical changes were identified

- Change existing behavior of a task which results in changes in the execution time distribution.
- Add a task to the system.
- Change the periodicity of an existing task.
- Change the priority of an existing task.

By introducing the exact same change in the model as in the system and then comparing their behavior we can increase the confidence in the created model. Any divergence between the behavior of the simulated model and the system indicates that the model is missing vital information, in which case more details must be introduced in the model.

> **Example:** The system S contains two tasks, Task A and Task B. Furthermore, it contains a binary semaphore protecting a shared resource. A timeout occurs if a task has been waiting on the semaphore for a certain time, which causes an exception in the system. The timeout is never supposed to occur. If the execution time of Task A is increased, the timeout can however occur. If the timeout is left out of the model, the model M and system S will not behave in the same way anymore if the execution time is increased. This indicates that the semaphore timeout behavior has to be introduced in the model as well.

The accuracy of the model is dependent on the quality of the measured data (probability distributions of tasks execution time). The measuring of the data should affect the system as little as possible. Too big probe effect on the system will result in an erroneous model and might cause wrong decisions regarding future developments.

When measuring execution times, it is possible to get a quality estimate of the measurement by studying the effects of changing the size of the overhead caused by the software probes used for the measuring. If small adjustment in the overhead size causes large effects in the system behavior, the measurements can not be considered reliable, since it is likely that the introduction of probes had a large impact on the system behavior. On the other hand, if only very small effects can be observed, the measurements can be considered more reliable, but we don't know anything for sure, since the probes has a minimum overhead size > 0.

2.2 Model validity

In this section we will discuss how to assure model validity, i.e., the activity of establish confidence in the constructed model. This is an important and necessary part of constructing models. Existing analytical methods determines if or not the temporal behavior of a system is safe given that the formal model is correct, e.g. that the estimates of the worst case execution time (WCET) of each tasks is safe [3][4]. In order to be safe, the WCET is assigned a value that is as tight as possible but slightly larger than the actual WCET. Such a method however tends to over-constrain the

system as the worst-case always is considered. 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 meets its requirements. It is also possible to use fixed, worst case, values in the simulation instead of the measured distributions. This will result in a worst-case analysis similar to the fixed priority analysis introduced in [7], but with the advantage of knowing semantic relations between the tasks, thereby excluding "impossible" execution scenarios and thus lowering the pessimism. However, finding a safe but not overly pessimistic worst case execution time is hard, but that is not in the scope of this paper.

There may be many valid models of one single system. Observing and measuring a systems behavior may give a system model that is valid given that the assumptions do not change. We will exemplify the phenomenon with a small physical experiment.

> **Experiment**. The experiment aims at deciding an equation (a model) for calculating how high a ball bounces of the ground after being dropped from a certain height. Repeatedly dropping the ball from different heights and measuring the height of the bounce determine the equation. The resulting equation could relate the bounce proportionally to the height from which the ball is dropped. This is a completely valid model as long as nothing is changed. We can even change the size of the ball without making the model invalid. However that model is too simple for capturing changes in e.g. the material of the ball or the material in the ground for that matter. We can transfer the physical experiment onto our method for analyzing the temporal behavior of a complex system. We can convince ourselves that the model is valid by comparing the output from the simulator with the values measured in the system. However by changing the model in order to analyze the impact of adding new features to a system can potentially invalidate the model. Whether or not the model is completely valid becomes evident only after implementing the new feature, i.e., when we have something to compare with. However, the more confidence we have in the model the more confident can we be in the simulation results, i.e., before implementing the new feature. Continuously maintaining and validating the model as part of the development process is the way in which the model is iteratively refined and kept consistent.

To exemplify the model validity problem consider a computer system with two tasks A and B. Task A sends a message to task B. This message passing is modeled in ART-ML. The simulation results indicate that the system is correct. Now task C is added to the system which also sends a message to task B. This changes the temporal behavior of task B. However, we only model task C as an execution time distribution leaving the message passing out. As a consequence the simulations of task B diverge from what we can observe in the changed system. The model that initially was correct is now incorrect due to lack of details.

2.3 System identification

System identification is a technique used in the domain of control theory [5]. By measuring and observing the input-output relationship between signals in the process a model can be determined in terms of a transfer function. Validating models based on the system identification approach is somewhat related to testing. Typically output signals produced by a simulation of the model are compared with the output signals of the physical process. Hence, the model is regarded as correct if the simulations and the physical process generate approximately the same output. Moreover, a method called residual analysis can be applied on models of continuous systems; observing weather or not the residual (the error in the prediction) and the input signal are independent. If the error depends on the input, it indicates that there are dynamics in the system that is not in the model. Testing the model with different input signals and comparing the prediction with the signals produced by the actual system is fine if the process is continuous in its nature. It is fair to assume that we can interpolate the behavior in between the tested signals. However, computer systems are not continuous; they are discontinuous systems meaning that the behavior may change dramatically as a result of small changes in the system. Our approach to model validation is similar to the one proposed in system identification. Our hypothesis is that potential discrepancies will be exposed if we introduce changes in the system and the corresponding changes in the model. Comparing the simulation results with the data measured in the system will give us the possibility to settle the validity of the model. The model validation process we propose is depicted in Figure 3.

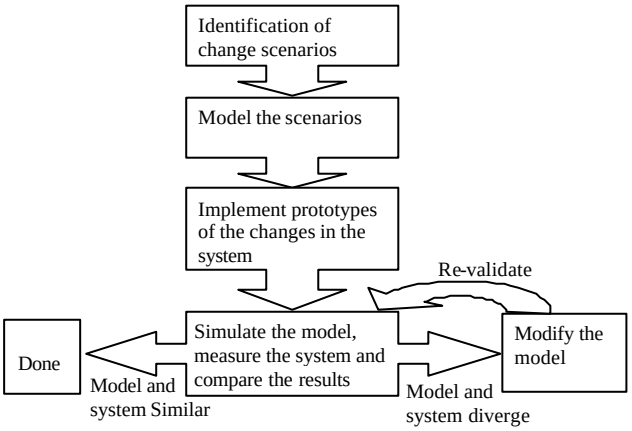

Fig. 3. The process of validating the model.

The first activity is to develop a set of change scenarios which should reflect typical, possible, and foreseen changes in the system. This corresponds to the scenario elicitation described in [6]. The set of scenarios are system specific, i.e., they are valid only for the system for which they were developed. After having selected a suitable set of appropriate and concrete change scenarios it is time to implement them in the model. By concrete we mean that scenarios such as change the priority of a task has to specify exactly what task to alter and what the new priority is supposed to be.

Moreover we have to introduce the proposed scenario in the system as a prototype that simulates the changes. Hence we do not implement a complete functional change. We aim at mimic the temporal behavior of the changes. For instance the scenario where the functional behavior of an existing task is changed we only have to inject code that simulate an increase or a decrease in execution times. Adding a new task is similar; a task containing only the temporal behavior is added to the model. Finally we compare the results from simulating the changed model to the behavior measured in the changed system. If they both behave the same way for every identified change scenario we have established confidence in the model. However if the behavior of the simulation and the system diverge we must tune the model. Typically more details have to be introduced in the model. Examples of such details are a more detailed model of tasks logical behavior and executional dependencies among task such as communication.

3 Conclusion

In this paper we present a non-intrusive method for increasing the maintainability of complex industrial real-time systems. This method aims to create a valid formal model of an existing system and using that model for analyzing the effect of changes. The challenge of this approach is to develop a valid model with a minimum of complexity and to show that the model is valid for use. We have presented our initial approach to this problem, which is based on identifying likely changes and incorporate the changes both in the model and target system and compare the outcome from the simulation and the real system.

4 References

[1] A. Wall, J. Andersson and C. Norström. Probabilistic Simulation-based Analysis of Complex Real-Time Systems. In proceedings of the 6th IEEE International Symposium on Object-oriented Real-time distributed Computing, May 14-16 2003, Hakodate, Hokkaido, Japan.

[2] 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.

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

[4] 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.

[5] R. Johansson, System Modeling Identification ISBN 0-13-482308-7 Prentice Hall.

[6] P O Bengtsson, Architectural Level Modiability Analysis PhD thesis Blekinge Institute of Technology. Sweden 2001.

[7] M. Joseph, P. K. Pandya, Finding Response Times in a Real-Time System, *The Computer Journal* 29(5): 390-395, 1986.