Performance Evaluation of Complex Real-time Systems: A Case Study*

Jinfeng Huang[1], Jeroen Voeten[2], Piet van der Putten[2], Andre Ventevogel[3], Ron Niesten[3] and Wout vd Maaden[3]
[1]Eindhoven University of Technology, Eindhoven Embedded Systems Institute
P.O. Box 513, 5600MB Eindhoven, the Netherlands
E-mail: J.Huang@tue.nl
[2]Eindhoven University of Technology, Faculty of Electrical Engineering
[3]TNO Industrial Technology, 5600HE Eindhoven, the Netherlands
E-mail: A.Ventevogel@ind.tno.nl

Abstract— **With the increase of complexity of modern industrial systems, it is more difficult than ever to predict their qualitative (correctness) properties and quantitative (performance) properties. In order to cope with the complexity of these systems and shorten their design time, system-level design has become one of the most important parts in the modern design flow. The core of a system-level design method is an expressive and well-founded modelling language, providing ample means for developing adequate system models. POOSL (Parallel Object-Oriented Specification Language) is one of such modelling languages which enables to construct succinct executable models for complex real-time hardware/software systems.**

This paper describes our experience with applying the POOSL language to model the industrial MA3 (a More Accessible Micro Assembler with a Modular Architecture) system and to evaluate its performance. The system can be characterized as a distributed real-time system consisting of independent processing units physically interconnected by an Ethernet. The POOSL model of the system examines its real-time performance from a system-level point of view. By evaluating the response delay of the system against the capacity (the number of processing units) of the system, proper design decisions can be made about the architecture of the system.

Keywords—**System-level Design, Performance Evaluation, Parallel Object-Oriented Specification Language (POOSL).**

I. Introduction

The industry has suffered for many years from coping with the contradiction between the complexity of real-time systems and the limited design time and cost. Many design methods have been proposed to alleviate this pain, such as the "waterfall model" documented in 1970 by Royce [1], the "spiral model" presented in the middle of 1980s [2], and Rose RT introduced in 1990s [3], In general, most of these methods agree on spending sufficient time and cost on building and evaluating models of the system before carrying out the implementation phase, because finding design errors in the modelling phase is much more economic in time and cost than in the implementation phase. However, most of these methods can no longer keep pace with the system complexity, due to the lack of efficient ways to handle new features of current generation systems, such as concurrency, distribution, real time and complex functionality.

System-level design methods are gradually being accepted by industry to cope with the complexity of modern industrial systems and shorten their design time. The core of a system-level design method is an expressive and well-founded modelling language providing ample means to reason about the properties of complex concurrent real-time system at the right abstraction level. POOSL is such a modelling language. It enables designers to construct a succinct executable model for complex real-time hardware/software systems [4], [6]. Such a model is of great benefit to overcoming design problems mentioned above. First of all, the model describes the investigated system at a high level by abstracting from unnecessary details, saving cost and time. Secondly, the model enables designers to detect design errors of the system at a very early stage, preventing expensive and time-consuming design iterations. Thirdly, the model helps designers to make design decisions at the system level, for example, rapid evaluation of the system performance based on different system architectures.

MA3, a distributed real-time assembler, is used to integrate different process modules and build a MST (Mi-

*This research is supported by PROGRESS, the embedded systems research program of the Dutch organisation for Scientific Research NWO, the Dutch Ministry of Economic Affairs and the Technology Foundation STW.

cro System Technology) production line [8]. This assembler hides details of control, transport, communication and monitor functions and provides a concise interface for integration. Because of its complexity, evaluating the performance of this system is quite difficult.

This paper describes our experience on modelling and analyzing the industrial MA3 system using the system-level language POOSL. Section 2 gives a brief introduction of the POOSL language. In section 3, a description of MA3 is given together with the performance issues. Several important modelling issues are also discussed. In section 4, the evaluation results are presented and design decisions are made based on these results. Conclusions are given in section 5.

II. The POOSL Language

Modelling languages should possess adequate expressive power for designing industrial systems. Modern industrial systems often have concurrent, real-time and distributed features. However, traditional modelling languages, such as C++, Pascal, and Fortran, are used to express sequential algorithms and do not support above features directly. The POOSL language, on the other hand, integrates a process part based on CCS (Calculus of Communicating Systems) with a data part based on traditional Object-oriented language. Therefore, a series of features are supported in the POOSL language. In this section, a brief overview of the expressivity of the POOSL language and its tool support are introduced.

A. Distribution

In a POOSL model, processes are the basic independent execution units. The distribution of these processes is expressed by clusters and static channels. A cluster is composed of processes and other clusters and acts as an abstraction of them, and static channels are used to combine two independent execution units (processes and clusters). The composition of processes and clusters derives from the rename, relabel, and parallel compositional combinators of CCS.

B. Concurrency

Concurrency is supported in the process part of the POOSL language at two different granularity levels. Firstly, processes or clusters asynchronously perform their activities and synchronously communicate with each other through static channels. Secondly, inside a process, a coarse grain concurrency mechanism is provided for operating on shared data object without violating mutual exclusion [6].

C. Real Time

The POOSL language is equipped with a mathematical semantics. The semantics is based on a two-phase execution model [5]. The state of the system can either change by asynchronously executing actions (Phase 1) or by letting the time pass synchronously (Phase 2) . Therefore, behaviors, depending on (hard) real-time concepts such as time-outs, watchdogs and deadlines, can be expressed in POOSL models. The formalization of the real-time feature of the POOSL language can be found in [5].

D. Data

The data part of the POOSL language is based on traditional object-oriented programming languages such as C++ and Smalltalk and is also equipped with formal semantics. Data objects are used to describe complex functional behaviors of processes. Moreover, data objects can be exchanged between processes, which makes the POOSL language suitable for modelling both control-flow and data-flow oriented systems.

E. Expressivity

Besides the above features, the POOSL language has the following powerful constructs representing the behavior of a complex hard real-time system:

- synchronous message passing primitives;
- choice,loop and select primitives;
- guarded commands;
- parallel and sequential composition;
- method abstraction;
- tail recursion;
- interrupt, abort and delay primitives.

Equipped with all these constructs, the POOSL language is capable of modelling dynamic hard real-time behavior and static distribution of complex hardware/software systems in an effective way.

F. Tool Support

A number of tools have been developed for building and analyzing a POOSL model. Some of them are shown in Fig. 1. An interactive simulator tool allows designers to incrementally specify and modify classes of data, processes and clusters and easily express hierarchical and topological structure of the complex systems. Furthermore, it can create log files for performance evaluating purposes. An interaction diagram tool helps designers to inspect the history of all messages exchanged between different entities by generating interaction diagrams automatically during simulation, These diagrams can be used for validation purposes of a model.

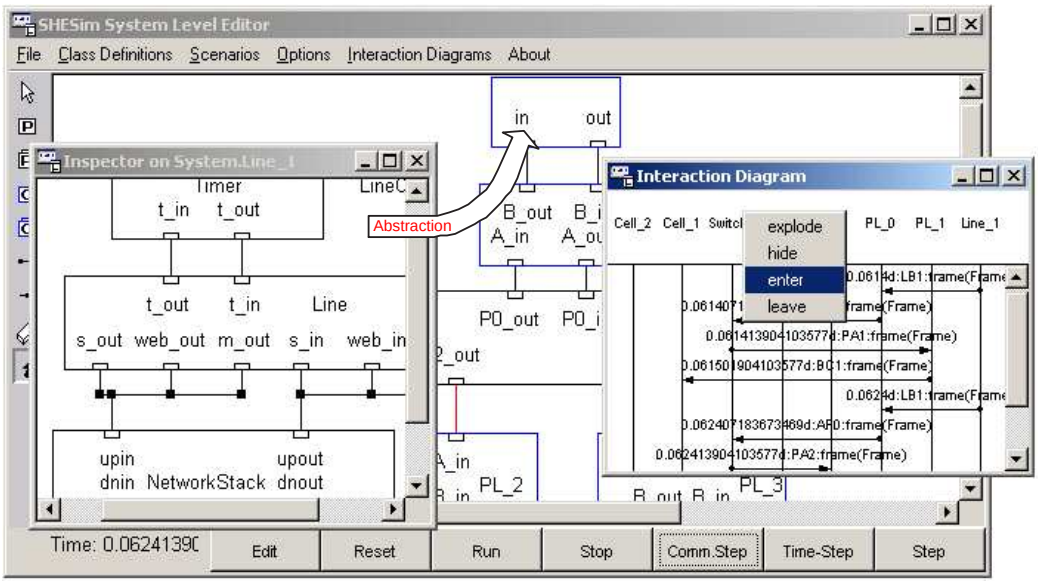

Fig. 1. The Simulation Environment for POOSL

III. Modelling MA3 MST Assembler

A. MA3 MST Assembler Description

MA3 MST assembler, a universal assembler for MST production systems, is produced by TNO industrial technology. The purpose of the assembler is to (semi)automatically assemble MST production systems by plugging in necessary process modules, which is an expensive and time-consuming process at present. MA3 MST assembler provides a concise interface for integration of process modules by hiding complex integrative functions such as control, transport, communication, and monitor functions. For example, when a process module is plugged into the MA3 MST assembler, the assembler can automatically detect this module, send control commands to it, synchronize its work with other process modules, and monitor its working states through the network. Fig. 2 illustrates a prototype of the MA3 MST assembler.

Fig. 2. A Prototype of MA3 MST Assembler

B. Performance Issues

The MA3 MST assembler encapsulates complex functions for integrating MST process modules and building a MST production line. Due to the strict requirements of the MST production line such as high-sensitivity, -quality, and -speed, the real-time performance of the MA3 MST assembler becomes one of the key points for its success. In particular, we investigate the following two real-time performance metrics of the assembler in the POOSL model:

1. The response delay distribution of process modules to a control command;
2. the update delay distribution of the monitor system.

In principle, the delay distribution can be obtained after visiting all possible states of the model, which contains all possible scenarios of the MA3 MST assembler. However, it's impossible to explore such a huge state space in the limited time and memory space. Therefore, in practice, the following two steps are carried out for obtaining a useful result:

1. According to the performance metric to be investigated, a adequate abstraction of the MA3 MST assembler is made, which considerably reduces the the state space of the model by removing irrelevant states.
2. After the first abstraction, the state space of the model is still too huge for exhaustive performance analysis. The use of simulation is therefore unavoidable. Simulation provides a way for estimating the performance figures of the system. A method for accuracy analysis is discussed in [7], where a general batch-means technique is introduced for evaluating it.

C. Modelling Experience

A model, an abstract representation of a certain design realization, can result in a creditable analysis only if it includes adequate information. A more adequate representation of the realization is usually achieved by adding more details. However, building such a detailed model is often time-consuming, expensive and error-prone, and its complexity hampers the (automatic) analysis of the model. Therefore, for an accurate and effective analysis, the model should be adequate for the properties relevant for making a design decision and at the same time be as abstract and simple as possible.

To satisfy these modelling requirements, an expressive modelling language is necessary for making a succinct model. Besides a powerful modelling language, the experience and knowledge of the designer also plays an important role on making a succinct and adequate model. This is largely due to the fact that the design of modern industrial systems usually involves multidisciplinary knowledge and requires participation of experts from different fields. For example, the MA3 MST assembler consists of several Hw/Sw technology components such as microprocessors, digital signal processors, sensors, network switches, PLC, RTOS kernels, databases, communication protocols and network applications. The model developer needs to communicate with the experts of each field and sufficiently understand the function of every component. Furthermore, the developer should be capable of distinguishing irrelevant and unimportant information according to the properties to be investigated. All these skills are indispensable for making an adequate and succinct model of a complex real-time system.

D. Model Validation

An analysis is based on a model of the system. The analysis is unreliable without an adequate model. It is therefore necessary to validate the adequacy of a model carefully. The validation of POOSL models is supported by the interaction diagram tool of SHESim [4] which allows designers to inspect every simulation step. In our case, the model is also validated by comparing it with a rapid prototype EPT of the MA3 MST assembler, because the rapid prototype implements most of the details of the realization and provides good estimations of the system performance.

E. Rapid Prototype and System-level Model

Both a rapid prototype and a system-level model are used to evaluate the system. Comparing to the rapid prototype, the system-level model has many merits. Not only does it take less expense and time, it also provides more flexibility for the designer to check different design solutions and different scenarios. Examples include:

- The capacity of the MA3 MST assembler is designed for assembling a maximum of 256 process modules grouped into different cells. The EPT prototype only integrates four process modules in two cells. In order to check the performance metrics for different capacities, it is not practical to insert more process modules into the EPT prototype because of physical constraints and cost. On the other hand, it only takes several minutes to reconstruct another POOSL model for the MA3 MST assembler with a different capacity.
- Some ”rare” events may greatly change the behaviors of the system and are difficult to capture and repeate in the prototype. For example, in the MA3 MST assembler, when a user instructs a monitor process to inspect another process module, the monitor process first asks for a web page from the process module, which slows down the command response time of the process module for a short period of time. In some particular situations, a number of monitor processes are started to inspect the same process modules at (almost) the same time, which results a considerably longer delay of the process module. However, this scenario can hardly be observed in the EPT prototype, because of its inability to simultaneously perform an operation on a number of physically distributed monitor processes. On the other hand, in the POOSL model, it is easy to simulate the behavior that a number of monitor processes are simultaneously to inspect the same process module at exactly the same time or in a very short time interval (e.g. 10 ms).

IV. Performance Analysis of the MA3 MST Assembler

In order to validate the adequacy of the POOSL model with regard to the investigated performance metric, we measure the delay of the response of a process module to a certain control unit in the rapid prototype EPT and the POOSL model. The results are shown in Fig. 3 in which diagram (a) is from EPT and diagram (b) is from the POOSL model. In both diagrams, the packet response delay stays around 0.025 second except a peak caused by web page transfer. It can be seen that the the POOSL model exhibits a similar behavior to EPT.

Fig. 4 illustrates the performance analysis results based on a POOSL model which has 64 process modules. Fig. 4(a) shows the web page request occurrences for a certain unit A by the elapse of time. Web requests in Fig. 4(a) results in corresponding peaks in responding delay in Fig. 4 (b). The more web request are made within a small time interval, the larger response delay will be.

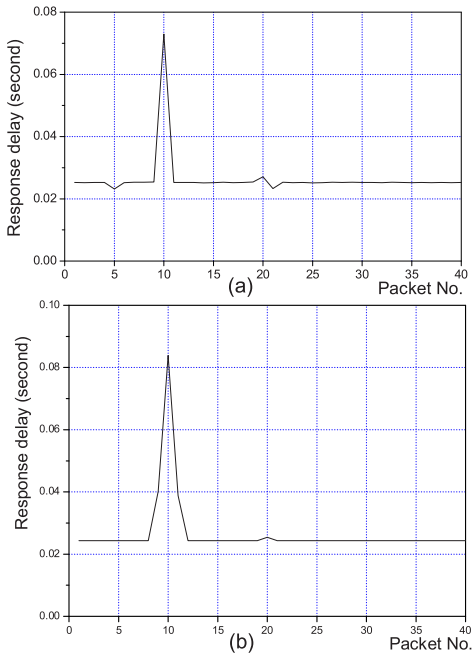

Fig. 3. The Response Delay of a Process Module

There are two ways to reduce the response delay and improve the performance of the system. One is to decrease the frequency of the web page request. This is only performed by decreasing the number of monitor processes in the system, which may cause inconvenience to users. The other way is to reduce the size of web pages, which is easier to accomplish during the design phase.

Fig. 5 illustrates the performance analysis results based on a POOSL model in which the web page size is reduced by 50% (from 102K bytes to 51K bytes). Comparing the delay distribution between Fig. 4(b) and Fig. 5(b), it can be seen that the delay of the peak positions is considerably shortened.

V. Conclusion

This paper presents our experiences with applying the POOSL language for modelling a complex real-time system, the MA3 MST assembler. POOSL is an expressive language with unambiguous syntax and mathematic semantics. The expressive power makes POOSL especially suitable for describing the behavior of complex real-time systems at a high level of abstraction without implementing details in term of hardware and software. Our experience shows that the POOSL language not only can be used to model complex real-time systems but also provides a low-cost and flexible solution for evaluating the performance of such systems.

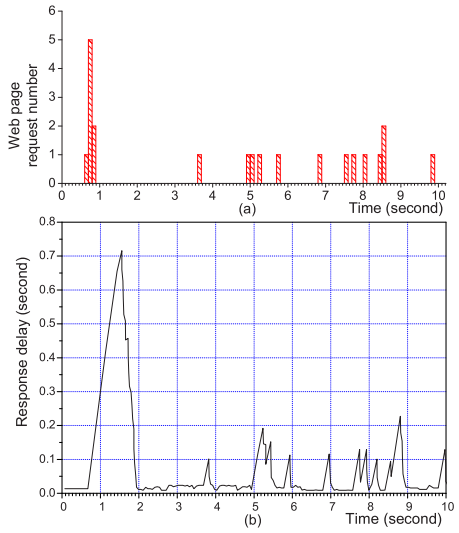

Fig. 4. The Response Delay of the Unit A (web page size: 102K bytes)

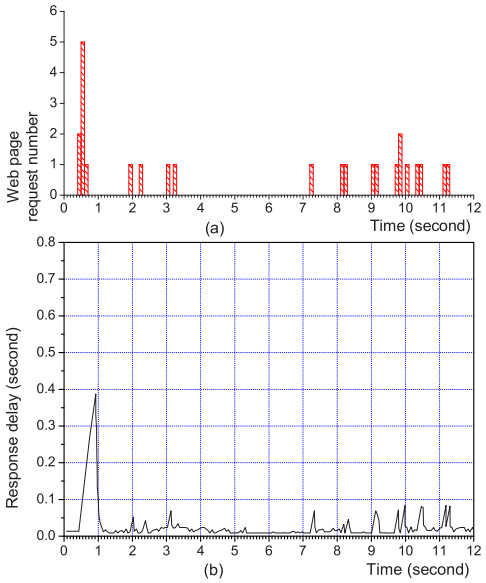

Fig. 5. The Response Delay of the Unit A (web page size: 51K bytes)

References

[1] W. W. Royce, Managing the development of large software systems: concepts and techniques, *Proceedings of IEEE WESTCON*, pp. 1–9, Los Angeles, 1970.

[2] B. Boehm, A Spiral Model of Software Development and Enhancement, *ACM SIGSOFT Software Engineering Notes*, August 1986.

[3] B. Selic, G. Gullekson, P. T. Ward, Real Time Object-Oriented Modelling, John Wiley, April 1994, ISBN 0471599174.

[4] P.H.A. van der Putten and J.P.M. Voeten, Specification of Reactive Hardware/Software Systems - The Method Software/Hardware Engineering, Ph.D. Thesis, Eindhoven University of Technology, The Netherlands, 1997.

[5] M.C.W. Geilen and J.P.M. Voeten, Real-Time Concepts for a Formal Specification Language for Software / Hardware Systems *Proceedings of ProRISC 1997*, STW, Technology Foundation, Utrecht, 1997.

[6] M.C.W. Geilen, J.P.M. Voeten, P.H.A. van der Putten, L.J. van Bokhoven and M.P.J. Stevens, Object-Oriented Modelling and Specification using SHE, *Journal of Computer Languages*, special issue for VFM'99, Vol. 27, Issues 1-3, April-October 2001.

[7] B.D. Theelen, and J.P.M. Voeten, Y. Pribadi, Accuracy Analysis of Long-Run Average Performance Metrics *Proceedings of PROGRESS'01*, pp. 261-269, ISBN 90-73461-27-X Utrecht (Netherlands): STW Technology Foundation, October 2001.

[8] TNO Industrial Technology, TNO Industrial Technologies–The MA^{3}MST assembly solution *http://www.ind.tno.nl /mechatronics/ma3/home.htm*, March 2001.