On the Verification of Coordination*

Paul Dechering[1] and Izak van Langevelde[2]

[1] Hollandse Signaalapparaten B.V.
P.O. Box 42, 7550 GD Hengelo, The Netherlands
paul@dechering.net
[2] Centrum voor Wiskunde en Informatica
P.O. Box 94079, 1090 GB Amsterdam, The Netherlands
izak@cwi.nl

Abstract. Scenario-based verification is introduced as a technique to deal with the complexity of coordination languages, which feature both data manipulation and concurrency. The approach is exemplified by a verification study of the software architecture SPLICE that is used by Hollandse Signaalapparaten. A detailed specification of SPLICE, including the ETHERNET network that SPLICE is using, is written in the process-algebraic language μCRL and for a number of selected scenarios the transition system is automatically generated. For the resulting models, the properties of deadlock freeness, soundness, and weak completeness are automatically proven by model checking.

1 Introduction

Coordination languages are designed to facilitate the development of distributed systems by offering a coherent model of data and control through a clear-cut interface. However, it is far from obvious that the complex structure of distributed data and control, so nicely hidden from the coordination programmer, indeed constitutes the coherent model the programmer has in mind. It is exactly the combination of distributed data and distributed control which makes the verification of coordination a difficult problem.

In isolation, both data manipulation and distributed control have been subjected to verification studies. Classical theorem proving approaches [9] were already, at least theoretically speaking, suitable for proving properties of calculations in sequential programming languages. Furthermore, the study of concurrent systems made possible the verification of complex parallel systems, either through human theorem proving [11] or through semi-automated verification

* Supported by ORKEST: "Onderzoek naar Randvoorwaarden voor de Konstruktie van Embedded SysTemen"

Appeared in: António Porto and Gruia-Catalin Roman (Eds.), Proceedings of COORDINATION 2000, Lecture Notes in Computer Science 1906, pp.335-340, September 2000, Springer Verlag

techniques like model-checking (see [4] for a recent overview). However, real-life systems readily exceed the capacities of the human theorem prover, and the role of data is a well-known cause of the infamous state space explosion. The combination of existing approaches to bridge the gap between data and concurrency is a research field of growing interest (for instance, see [12], where the integration of theorem proving and model checking is discussed), but to date the state of the art of verification falls short when it comes to realistic coordination.

This paper addresses the verification of coordination by proposing a verification technique that lies between model-checking and testing. The idea is to formally specify the coordination language, but instead of generating one all-embracing model, which would be astronomically large in size, if not infinite, to generate and verify models for specific situations. Each of these so-called *scenarios* represents one of the common situations the system may encounter. This way, each scenario covers a slice of the system's behaviour and by considering more and more scenarios the system can be covered more and more.

A test, based on a test case, considers one of the possible courses of actions the system may execute in this test case. In a concurrent system, with a high degree of non-determinism, even for one test case a large number of test runs might not reveal one of those rare error situations. A verification, based on one scenario, will consider all possible courses of actions the system can execute, including possibly rare error situations. So, the approach of verifying all possible variations in system behaviour in isolated test cases is more general than testing, which covers isolated traces of system behaviour in isolated test cases. On the other hand, it is less general than model checking, which covers all variations of system traces for all possible test cases.

The approach of *scenario-based verification* is exemplified with the coordination language SPLICE [1]. A detailed specification of SPLICE was written in μCRL [8], a language based on process algebra with data, and transition systems were generated using the native μCRL tool set. The actual analysis was supported by the CÆSAR/ALDÉBARAN tool set [7].

The paper is organised as follows. Section 2 introduces the coordination language SPLICE and Sect. 3 describes the approach of scenario-based verification using μCRL. Section 4 describes the properties to be verified and Sect. 5 presents the verification results. Finally, Sect. 6 evaluates the approach presented. The full report, including all specifications, of the research presented in this paper is [6].

2 An Overview of SPLICE

SPLICE (Subscription Paradigm for the Logical Interconnection of Concurrent Engines) was introduced as an architecture for *control systems* [1]. Non-functional requirements, e.g. a certain level of fault-tolerance, real-time constraints, adaptability, and concurrency, imposed on this type of systems make the development of control systems a complex matter, and the solution SPLICE offers to reduce

this complexity is neatly tucked away behind its programming interface. Thus, SPLICE programmers can concentrate on the functional requirements.

The SPLICE kernel consists of a number of *agents* connected by a communication network. Each application running under the architecture is connected to an agent, which is responsible for the coordination with the other agents. A SPLICE system call by an application is processed by its agent, which typically sends requests over the network to the other agents, gathers responses and manages its local dataspace. Also, if one application does not respond anymore, it is replicated on another processor, making the system to some extent immune to processor and network failure. All this is implemented by the SPLICE agents, hidden away from the applications which just observe a stable and coherent environment of coordinated applications.

3 Scenario-Based Verification using μ

The SPLICE architecture was specified in detail in μCRL [8], a language based on process algebra with data. The full specification takes up about 54 KB, which can be found in the full version of this paper [6]. The corresponding labelled transition system was generated using the μCRL tool set and fed into CÆSAR/ALDÉBARAN [7].

The CÆSAR/ALDÉBARAN tool set is used to further analyse the system generated. First, the system is *reduced modulo weak bisimulation*, which means that all non-observable actions, i.e. the interaction between SPLICE agents and the network, and all network activity, are abstracted away, finally resulting in a much smaller transition system. Then, this reduced system is subjected to model checking. This verification technique consists of checking whether properties, expressed as a theory in some temporal logic, are satisfied by a transition system, interpreted as a Kripke structure. The strong point of model checking as a verification technique is that it can be efficiently automated.

However, even for simple SPLICE applications, the size of the transition system generated already exceeds practical limits by several magnitudes, so somehow a restriction has to be imposed on the systems tested. The restriction that underlies the verification technique promoted in the current paper is to verify the application in certain well-defined situations or 'scenarios'. A scenario is a limited environment interacting with the application, for instance by reading and writing data, or by issueing application commands. Scenarios play the role in verification that is played by test cases in testing.

Summarising, scenario-based verification using μCRL consists of specifying both the SPLICE architecture and the scenarios in μCRL, to generate for each scenario the labelled transition system of this scenario and the architecture, and to reduce and model-check the resulting system, to verify the desired properties.

4 The Properties of Splice

The three properties of Splice studied in this paper are deadlock freeness, soundness, and completeness. The first speaks for itself, the second and third state that everything that can be read has been written and everything that is written can be read. However, generally the three properties do not hold.

Splice was designed to enable a high level of fault-tolerance, but in exceptional circumstances the system might break down. For instance, in the situation where one Splice component receives data at a higher rate than it can handle, an internal queue overflows and data is lost. The properties to be verified were weakened to apply to those situations where no exceptional disasters, flagged by 'panic' actions, happen.

Even with this restriction, completeness does not hold in its strong formulation (i.e. "all that is written can subsequently be read") for a number of reasons. First, it takes time for the record to be transferred over the network from the writer to the reader, so only eventually will the reader be able to access this record. Second, it is possible that the record is overwritten by another record with the same key, even if this second record was written before the first one. All that can be guaranteed is weak completeness.

deadlock freeness as long as no 'panic' or proper termination occurs the system is able to proceed with some action

correctness a record that is read was written in the past

weak completeness as long as no 'panic' occurs, it is *possible* that eventually a record can be read that was written and not overwritten by another record with the same key

These formulations are an informal rendering of the properties verified. The exact specification in temporal logic in the Evaluator [7] syntax is given in [6].

5 Experiments

For the verification of Splice itself, irrespective of any particular application, the scenarios must cover the characteristics of the architecture by making typical combinations of API calls. It goes without saying that the number of possible scenarios is huge, but to give an idea of the principle of scenario-based verification, a small number of scenarios consisting of simple combinations of read and write actions is used.

The scenarios that have been verified all consist of two applications, parameterised by: 1) the application reads any or each record that satisfies a query, or it does not read at all; 2) the application writes or not; 3) the application loops or not. The six scenarios are summarised in Table 1; the full specifications are presented in [6].

For each of the scenarios verified, the three properties of deadlock freeness, soundness, and weak completeness were proven to hold, which presents a modest

Table 1. The verified scenarios with the size of their transition systems and the CPU time to generate these (The experiments were performed on an 300 MHz MIPS R12000 processor)

	application 1			application 2			generation metrics		
scenario	reads	writes	loops	reads	writes	loops	# generated	# reduced	CPU time
1	any	yes	no	any	yes	no	846360	961	6h42m
2	each	yes	no	each	yes	no	554707	702	4h20m
3	any	no	yes	any	no	yes	477392	463	3h30m
4	each	no	yes	each	no	yes	474394	363	3h27m
5	any	yes	yes	any	yes	yes	4561900	3789	37h38m
6	each	yes	yes	each	yes	yes	4458013	2471	32h03m

support for the claim that these properties hold in general. This evidence needs to be strengthened by verifying more scenarios.

An indication of the cost of the scenario-based verification of SPLICE is presented in Table 1, which shows for each scenario the size of the transition system initially generated, the CPU time of the generation process, and the size of the system after reduction modulo weak bisimulation. Two features of these metrics catch the eye. First, transition system generation is expensive for SPLICE, even for the simple scenarios considered. Second, the reduction modulo weak bisimulation, that is abstracting from all internal behaviour, results in a transition system that is several orders of magnitude smaller. As such, Table 1 nicely supports the claim that SPLICE realises complex communications which can be abstracted from to a relative simple level. However, the size of the transition system initially generated appears to be a bottleneck in the analysis, since currently there is no way known to directly generate the smaller reduced system from the specification. A technique like on-the-fly model checking, used in SPIN [10], might bring relief, but it is not available in the current setting of the μCRL tool set.

6 Conclusions and Related Work

Scenario-based verification was introduced as a technique for the verification of coordination languages. The rationale of the technique is to benefit from the exactness of formal verification, while avoiding the state space explosion. The approach was exemplified by the verification of SPLICE using the process-algebraic language μCRL, the μCRL tool set, and the CÆSAR/ALDÉBARAN tool set.

Related approaches all focused at SPLICE at a more abstract level, not aiming at automated verification, and not including the detailed specification of the network used in this paper. An operational semantics is defined in [2] and a process-algebra for SPLICE is defined in [5]. However, the fact that these models are less detailed and not geared towards automated verification does not imply that they are inferior to the approach of the current paper. This is clear from [3],

where manual theorem proving based on an abstract transition relation is used to establish equivalence results for a number of coordination models, including the one found in SPLICE. Conclusions at this level of generality cannot be drawn with a detailed model as is presented in this paper.

Scenario-based verification is limited in applicability in that the initial generation of transition systems is a true bottleneck. As was demonstrated in Sect. 5, the size of these increase quickly with the complexity of the scenarios, making the verification of more interesting scenarios impossible.

The strength of the approach, however, is that it facilitates the sound analysis of the key features of a coordination language, which is where both formal verification and testing fall short. For the former, any realistic model is too complex to be analysed formally, while the scope of the latter is limited to isolated system traces. Scenario-based verification is a golden middle.

References

[1] M. Boasson. Control systems software. *IEEE Transactions on Automatic Control*, 38(7):1094–1106, July 1993.

[2] M. M. Bonsangue, J. N. Kok, M. Boasson, and E. de Jong. A Software Architecture for Distributed Control Systems and its Transition System Semantics. In J. Carroll, G. Lamont, D. Oppenheim, K. George, and B. Bryant, editors, *Proceedings of the 1998 ACM Symposium on Applied Computing (SAC '98)*, pages 159 – 168. ACM press, Feb. 1998.

[3] M. M. Bonsangue, J. N. Kok, and G. Zavattaro. Comparing software architectures for coordination languages. In P. Ciancarini and A. L. Wolf, editors, *Proceedings of Coordination '99*, volume 1594 of *Lecture Notes in Computer Science*, pages 150–165. Springer Verlag, 1999.

[4] E. W. Clarke, O. Grumberg, and D. A. Peled. *Model Checking*. The MIT Press, 1999.

[5] P. Dechering, R. Groenboom, E. de Jong, and J. Udding. Formalization of a Software Architecture for Embedded Systems: a Process Algebra for Splice. In *Proceedings of the Hawaiian International Conference on System Sciences (HICSS-32)*, Jan. 1999.

[6] P. Dechering and I. A. van Langevelde. Towards automatic verification of SPLICE. Technical Report SEN-R0015, CWI, May 2000. Available from `http://www.cwi.nl`.

[7] J.-C. Fernandez, H. Garavel, A. Kerbrat, R. Mateescu, L. Mounier, and M. Sighireanu. CADP (Cæsar/Aldébaran development package): A protocol validation and verification toolbox. In R. Alur and T. A. Henzinger, editors, *Proceedings of the 8th Conference on Computer-Aided Verification*, volume 1102 of *Lecture Notes in Computer Science*, pages 437–440. Springer Verlag, Aug. 1996.

[8] J. F. Groote. The syntax and semantics of timed μCRL. Technical Report SEN-R9709, CWI, June 1997. Available from `http://www.cwi.nl`.

[9] C. Hoare. An axiomatic approach to computer programming. *Commun. ACM*, 12(10):576–583, Oct. 1969.

[10] G. J. Holzmann. *Design and Validation of Computer Protocols*. Prentice Hall, 1991.

[11] Z. Manna and A. Pnueli. *Temporal Verifications of Reactive Systems – safety.* Springer Verlag, 1995.

[12] S. Rajan, N. Shankar, and M. Srivar. An integration of model checking with automated proof checking. In P. Wolper, editor, *Proceedings of the 1995 workshop on Computer Aided Verification*, volume 939 of *Lecture Notes in Computer Science*, 1995.