In D. Haneberg, G. Schellhorn, and W. Reif, editors,
Proc. FM-TOOLS 2002, pages 53-58, Augsburg, 2002.

Distributed Model Checking *

– Extended Abstract –

Hans-Dieter Ehrich, Maik Kollmann, and Ralf Pinger

Abteilung Informationssysteme, Technische Universität Braunschweig
Postfach 3329, D-38023 Braunschweig, Germany
{HD.Ehrich|M.Kollmann|R.Pinger}@tu-bs.de

Abstract. We present an efficient method for verifying global conditions for distributed object systems using conventional model checkers. A distributed variant of CTL is used for expressing global conditions, making the distribution structure sufficiently visible to enable automatic translation into local conditions for the single objects and for the communication between them. These can be verified one after the other using existing model checkers. The method avoids state space explosion if the amount of communication traffic between objects is limited. It is illustrated by a large example where our method shows a dramatic speedup over conventional model checking.

Keywords. distributed logic, model checking, modelling and design, object system, temporal logic, verification.

1 Introduction

By an object system, we mean a community of sequential objects operating concurrently and communicating synchronously in an RPC-like way.

When model checking such systems, the complexity grows exponentially with the number of concurrent components. Therefore, the state space easily "explodes" in size, putting very tight limits to practicality.

We show that these limits may be overcome, based on the observation that global checking conditions of practical interest often display enough information about the system's structure so that they can be automatically transformed into local conditions. This cuts the effort down to a complexity which is (roughly) linear in the number of components: instead of checking the global model which is the product of the local models, a network of communicating local models is checked one after the other.

* This work was partially supported by DFG under contract Eh 75/12 in the priority project SPP 1064.

Compositionality approaches [Pnu85,CLM89,GL94,dR97], in contrast, work on the grounds that no knowledge about the system's structure is available, so there is the crucial step of recognizing internal interfaces and equipping these with appropriate assumptions and guarantees. This cannot be automatized except for very restricted cases. [CGP00] gives an overview of these and other techniques for reducing state space explosion.

We have been experimenting with a 'global' distributed logic D_1 that is translatable to a 'local' distributed logic D_0 [ECSD98,EC00]. This idea was developed further into a practical distributed model checking method [EP00,PE01,Pin02]. The approach is independent of the choice of model checker and its logic.

In the work reported here, we used CTL, the computation tree logic proposed by E. Clarke and A.E. Emerson [CE81], and the SMV model checker ([McM96]). In this contribution, we demonstrate the practicality of the method by model checking a large example (about 10^{24} states) which took a few minutes with our approach while conventional global model checking took hours. In order to be able to compare, we chose the steam boiler example which is well known from the verification literature (cf. [ABL96]).

In the next section, we give a brief informal introduction to our approach; details can be found in [PE01,Pin02].

2 Distributing checking conditions

Let I be a finite set of object identifiers. Each sequential object $i \in I$ has an individual set P_i of state predicate symbols, its *signature*, expressing, e.g., which attributes have which values, which actions are enabled, which actions have occurred, etc.

We use the distributed logic D_1 as described in [ECSD98,EC00] but instantiated with local logics i.CTL over signature P_i for each object $i \in I$, with the added possibility of using formulae from j.CTL for any other object $j \in I$ within i.CTL. These constituents are called *communication subformulae*. An example is $i.\mathsf{AG}(\mathrm{p} \Rightarrow j.(q \vee k.\varphi))$ where $j.(q \vee k.\varphi)$ and $k.\varphi$ are communication subformulae.

The intended meaning is that, in a formula $i.(\ldots j.\psi \ldots)$, the communication subformula $j.\psi$ holds in i in a given state iff i synchronizes with j and ψ holds in j during this synchronization.

D_1 has an interesting sublogic called D_0; it is like D_1 without communication subformulae, but with separate communication formulae of the kind $i.(p \Rightarrow j.q)$. Additionally, we have propositional symbols $@j$ in P_i

for every $j \in I$: these are supposed to mean that i currently synchronizes with j (which is expressed by $j.\,true$ in D_1). D_0 mimics synchronous RPC as supported by current middleware: in i, p *(a call)* implies synchronization with j assuring q *(execution of the called procedure)* there, possibly moving data back and forth during execution.

In [EC00], a sound and complete translation $\mathsf{D}_1 \to 2^{\mathsf{D}_0}$ is given: working inside-out, every formula $\varphi = i.(\ldots j.\psi \ldots)$ with a communication subformula is replaced by the D_1 formula $\varphi' = i.(\ldots q_i \ldots)$ and the D_0 formulae $j.(q_j \Leftrightarrow @i \wedge \psi)$, $i.(q_i \Rightarrow j.q_j)$, and $j.(q_j \Rightarrow i.q_i)$ where q_i and q_j are new propositional symbols to be added to the signatures P_i and P_j, respectively. This step is iterated with φ' for φ, etc., until φ' is in D_0.

This result can be used to break a 'global' D_1 checking condition down into a set of D_0 conditions which can be checked locally.

The new propositional symbols generated during the process represent *communication requirements* that have to be matched against the communication structure of the distributed model to be checked. This is elaborated in [EP00,PE01,Pin02]. The algorithm roughly works as follows. First, all states in j where ψ holds are calculated using the model checker; then all communication symbols in j occurring only in these states and representing communication with i are calculated; then the corresponding partner symbols in i are retrieved; then we test whether any of these match the requirements. The last step is done by another call of the model checker.

Most interestingly, existing model checkers can be used not only for local model checking but also for checking the communication requirements.

3 The steam boiler example

We illustrate the practicality of the method by means of the steam boiler example as treated in [ABL96] (cf. figure 1). Duval and Cattel have a model checking approach there using SPIN [DC96]. Our attempt to model check that global model had to be aborted after 40 hours computing time in our equipment, because of lack of memory[1]. So we used a reduced example omitting, e.g., some of the technical error handling for water level and steam outlet control.

We checked the following conditions, given in natural language and in distributed logic D_1.

[1] The experiment was run on a Sun UltraSPARC with 2 processors, 360 MHz, 2 GB memory each.

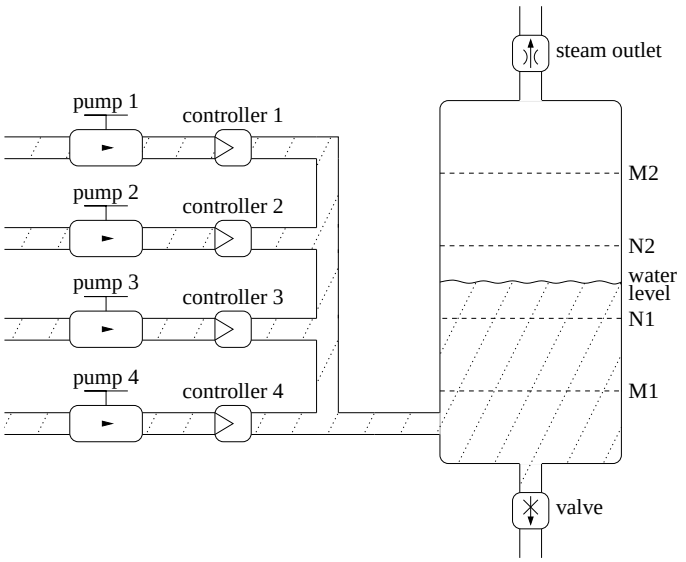

Fig. 1. Steam boiler

1. the valve is never open if control is in normal or restricted mode.
   $\text{control.}\mathsf{AG}((\text{mode} = \text{normal} \vee \text{mode} = \text{restr}) \wedge @\text{boiler} \Rightarrow \text{boiler.}(\text{valve} \neq \text{open}))$
2. after the initialization phase, the water level is ok or control stops operation.
   $\text{control.}\mathsf{AG}(\text{mode} = \text{ini} \Rightarrow \mathsf{AF}\ (\text{boiler.}(\text{waterlevel} \geq \text{N1} \wedge \text{waterlevel} \leq \text{N2}) \vee \text{mode} = \text{stop}))$
3. in normal or restricted mode, the water level is never below M1 or above M2.
   $\text{control.}\mathsf{AG}((\text{mode} = \text{normal} \vee \text{mode} = \text{restr}) \wedge \text{clock} = \text{get-water-level} \Rightarrow \text{boiler.}\neg(\text{water-level} < \text{M1} \vee \text{water-level} > \text{M2}))$
4. if a pump is defective, it will resume normal operation when receiving a repair message.
   $\text{pump}_i\text{.}\mathsf{AG}(\text{state} = \text{defective} \Rightarrow \mathsf{AF}(\text{action} = \text{getrepaired} \Rightarrow \text{state} \neq \text{defective}))$
5. in normal mode of control, no pump is defective.
   $\text{control.}\mathsf{AG}(\bigvee_{i=1}^{4} \text{pump}_i\text{.state} = \text{defective} \Rightarrow \text{mode} \neq \text{normal})$
6. if, in normal mode of control and with water between normal levels, the steam volume goes up to 20, then all pumps are successively switched on or the control leaves normal operation because of pump defects.
   $\text{control.}\mathsf{AG}(\ \text{mode} = \text{normal} \wedge \text{water-level} = \text{N1N2} \wedge \text{clock} = \text{send-pump}_1 \wedge \text{boiler.}(\text{steam-level} = 20) \Rightarrow (\mathsf{AX}(\text{pump}_1\text{.action} = \text{geton} \wedge \mathsf{AX}(\text{pump}_2\text{.action} = \text{geton} \wedge \mathsf{AX}(\text{pump}_3\text{.action} = \text{geton} \wedge \mathsf{AX}\ \text{pump}_4\text{.action} = \text{geton})))) \vee \mathsf{AF}\ \text{mode} \neq \text{normal})$
   $\text{pump}_i\text{.}\mathsf{AG}(\text{state} \neq \text{stop} \wedge \text{state} \neq \text{defective} \wedge \text{action} = \text{geton} \Rightarrow \text{state} = \text{on})$

	distributed				global
	control	boiler	pumps	total	total
#states	1.2×10^8	3.5×10^6	198		1.3×10^{24}
cond 1	$\approx$ 6 min 15 s	$\approx$ 42 s		$\approx$ 7 min	$\approx$ 2 h 57 min
cond 2	$\approx$ 4 min 38 s	$\approx$ 46 s		$\approx$ 5 min 20 s	$\approx$ 2 h 36 min
cond 3	$\approx$ 5 min 30 s	$\approx$ 46 s		$\approx$ 6 min 15 s	$\approx$ 2 h 52 min
cond 4			$<$ 1 s	$<$ 1s	$\approx$ 2 h 51 min
cond 5	$\approx$ 5 min 17 s		$<$ 1 s	$\approx$ 5 min 20 s	$\approx$ 2 h 50 min
cond 6	$\approx$ 4 min 27 s	$\approx$ 46 s	$<$ 1 s	$\approx$ 5 min 10 s	$\approx$ 2 h 43 min

Table 1. Results

Table 1 shows the results, comparing the distributed approach with the conventional global approach (last column).

The speedup is dramatic. Because we check component by component, there are no limits to the size of systems that can be checked by this method, as long as the components are small enough and communication traffic between these is not too dense.

4 Concluding remarks

Further study should explore in depth how the idea can best be utilized for checking object system designs. Appropriate tool support in combination with existing model checkers is essential. Tools for generating the SMV inputs from D_1 descriptions have been implemented [Pin02].

We note in passing that our approach employs an interaction mode that is more general than conventional ones. While the latter assume either global synchronization at each step or one-step-at-a-time interleaving throughout, we allow for an arbitrary choice between these at each step.

There are limitations to our technique: we cannot directly deal with indirect or asynchronous communication. Indirect communication like "*i*.(... *j will eventually receive this message* ...)" where the path of transmission is unmentioned, may in some cases be split into its direct communication parts, or an 'observer' object may be defined that communicates directly with the objects in question.

Systems using asynchronous communication may in principle be transformed to synchronous ones by introducing one-element message buffers between any two communicating objects. However, this is not very elegant and hardly practical. Finding a better solution would be important because asynchronous communication is prevalent in distributed systems.

References

[ABL96] Jean-Raymond Abrial, Egon Börger, and Hans Langmaack, editors. *Formal Methods for Industrial Applications: Specifying and Programming the Steam Boiler Control*, volume 1165 of *Lecture Notes in Computer Science*. Springer-Verlag, 1996.

[CE81] Edmund M. Clarke and E. Allen Emerson. Design and Synthesis of Synchronization Skeletons Using Branching Time Temporal Logic. *Lecture Notes in Computer Science*, 131:52–71, 1981.

[CGP00] Edmund M. Clarke, Orna Grumberg, and Doran A. Peled. *Model Checking*. MIT Press, 2000.

[CLM89] E. M. Clarke, D. E. Long, and K. L. McMillan. Compositional Model Checking. In *Proceedings fo the 4th Annual Symposium on Principles of Programming Languages*, pages 343–362, 1989.

[DC96] Gregory Duval and Thierry Cattel. Specifying and Verifying the Steam Boiler problem with SPIN. In [ABL96], pages 203–217.

[dR97] Willem-Paul de Roever. The Need for Compositional Proof Systems: A Survey. In Willem-Paul de Roever, Hans Langmaack, and Amir Pnueli, editors, *Compositionality: The Significant Difference*, volume 1536 of *Lecture Notes in Computer Science*, pages 1–22, September 1997.

[EC00] H.-D. Ehrich and C. Caleiro. Specifying communication in distributed information systems. *Acta Informatica*, 36(Fasc. 8):591–616, 2000.

[ECSD98] H.-D. Ehrich, C. Caleiro, A. Sernadas, and G. Denker. Logics for Specifying Concurrent Information Systems. In J. Chomicki and G. Saake, editors, *Logics for Databases and Information Systems*, pages 167–198. Kluwer Academic Publishers, 1998.

[EP00] H.-D. Ehrich and R. Pinger. Checking object systems via multiple observers. In *International ICSC Congress on Intelligent Systems & Applications (ISA'2000)*, volume 1, pages 242–248. University of Wollongong, Australia, International Computer Science Convetions (ICSC), Canada, 2000.

[GL94] Orna Grumberg and David E. Long. Model Checking an Modular Verification. *ACM Transactions on Programming Languages and Systems*, 16(3):843–871, 1994.

[McM96] Kenneth L. McMillan. *Symbolic Model Checking*. Kluwer Academic Publishers, second edition, 1996.

[PE01] R. Pinger and H.-D. Ehrich. Compositional Checking of Communication among Observers. In H. Hussmann, editor, *Fundamental Approaches to Software Engineering (FASE), Part of the Joint European Conferences on Theory and Practice of Software (ETAPS 2001), Genova*, volume 2029 of *Lecture Notes in Computer Science*, pages 32–44, 2001.

[Pin02] R. Pinger. *Kompositionale Verifikation nebenläufiger Softwaremodelle durch Model Checking*. PhD thesis, Institut für Software – Abteilung Datenbanken, TU Braunschweig, 2002.

[Pnu85] Amir Pnueli. In Transition From Global to Modular Temporal Reasoning about Programs. In Krzysztof R. Apt, editor, *NATO ASI Series*, volume F13, 1985.