International ICSC Congress on Intelligent Systems & Applications (ISA'2000), vol. 1, pages 242-248.
Univ. of Wollongong, Australia. International Computer Science Conventions (ICSC), Canada 2000

Checking object systems via multiple observers*

Hans-Dieter Ehrich and Ralf Pinger

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

Abstract. Verification of concurrent systems is usually done from the viewpoint of a sequential observer outside the system. We show that it may be useful to employ several communicating observers where the observers may be situated outside or inside the system. Since each observer can be checked separately, state-space explosion can be reduced to some extent. Communication among observers, however, may become complex and is not included in the checking. On the other hand, this separation of concerns opens the possibility to check communication infrastructure (like middleware) separately without checking it over and over again with the object systems built on it.

1 Introduction

Objects in the sense of this paper are agents with the following minimal requirements. (1) Their local behaviour is sequential; (2) local checking conditions can be described in a suitable logic like CTL; (3) they are able to communicate via a synchronous RPC-like handshake mechanism. This does not exclude agent features like autonomy, reactivity or proactivity, and not mentalistic notions like beliefs, desires or intentions either. However, these are outside the scope of the problem considered here, so we prefer to stick to the term "object". By an object system we mean a community of objects operating concurrently and communicating as described above.

For formulating checking conditions of object systems, we employ a distributed concurrent logic built upon the given local logics. This logic can be used to specify local conditions with the added possibility to formulate properties of other objects during communication points. In this sense, the logic is "global" although all statements are bound to a locality. We capitalize on a result on translating distributed logics published in [ECSD98,EC00] and show how such global conditions can be distributed over the objects involved, separating the necessary communication and making it explicit.

The approach is orthogonal to the method used for checking: testing, automatic theorem proving, interactive theorem proving, or model checking. In order to be specific, we describe our ideas from the model checking point of view because model checking has proven to be practical and offers great potential for further development.

In current approaches, when model checking a concurrent system, the global condition is formulated in a temporal logic like CTL and checked against a sequential state-transition model of the entire system. This model represents concurrency by interleaving. In a sense, it represents the viewpoint of a single sequential observer outside the system. The idea of this paper is to show that it may be useful to employ several communicating observers where the observers may be situated outside or inside the system.

When model checking a temporal condition φ against an object system S, φ is checked against $M(S)$, a sequential state-transition model of the system allowing (in the first place) for all interleavings of executions of its sequential constituents. That amounts to building the product of the state-transition models of the sequential components. The resulting state space explosion is fought by a variety of techniques like decomposition, abstraction, and partial-order reduction. This way, very large systems have been successfully checked, demonstrating the power of the techniques. We refer to [CGP00] where a comprehensive description of the state of the art is given.

We may consider $M(S)$ as an *observer* of the system S: it observes S by synchronizing at each step with one or more objects in the system (more precisely, with one step in each of these objects).

We suggest that it may sometimes be useful to split the condition φ into $n > 0$ conditions $\varphi_1, \ldots, \varphi_n$, and to distribute these over n observers $M_1, \ldots, M_n$. The observers may be external, i.e., not part of the system, like $M(S)$, but they may as well be members of the object system itself.

The goal is to check $M(S) \models \varphi$ by checking $M_1 \models \varphi_1, \ldots, M_n \models \varphi_n$ individually—and to check communication separately.

Indeed, in order for this to work, the observers must communicate correctly. Checking this is a separate concern, and it may be complex. But there is an advantage:

* This work was partially supported by the EU under ESPRIT IV Working Group 22704 ASPIRE.

object systems may be checked *relative to communication*, i.e., based on the assumption that communication among observes works correctly. The latter may be somebody else's responsibility (for instance that of a middleware vendor). And the approach opens the possibility to check communication infrastructure (like middleware) once and for all without checking it over and over again with the object systems being built on it.

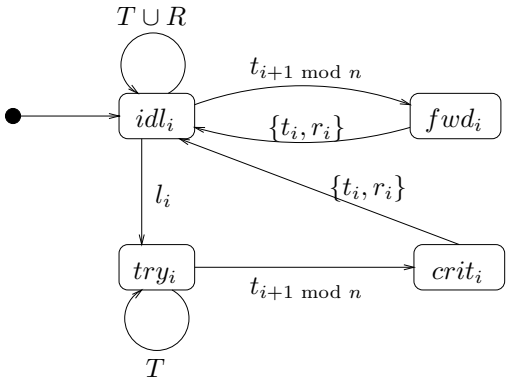
The established techniques as described in [CGP00] can of course be applied to each individual observer. In this sense, our technique is orthogonal and complimentary to the established techniques.

The only approach we are aware of that has some ideas in common with ours is the multi-agent approach presented in [BGS98]: each agent has its own view expressed as beliefs, desires and intentions about the system, expressed in a corresponding BDI logic. There is no explicit communication between agents in that approach.

In the next section, we give motivating examples to demonstrate the idea. In the 3rd section, we describe a multi-observer extension of CTL: each system condition is given from the viewpoint of one of the observers, using a local version of CTL enlarged by assertions about other observers. In section 4, we describe how to distribute these "global" assertions over the observers involved, introducing communication by a basic RPC-like mechanism.

2 Motivation

The idea is illustrated by the mutual exclusion example. In this example, we have one object for each process of which only one process is allowed to enter its critical section at the same time.



$T = \{t_j \mid j \neq i + 1 \bmod n\}, R = \{r_k \mid k \neq i\}$

Fig. 1. state diagram of process i

Example 1 (Mutual Exclusion). We present a "self-organizing" version of the n-process mutual exclusion problem that is advantageous in cases where the system is at rest most of the time because then there is no communication traffic. In the beginning, all processes $P_0, \ldots, P_{n-1}$ are idle, and the system is at rest. When k processes try to enter the critical region, $0 < k < n$, one is chosen nondeterministically and a token ring of all processes is initialized, giving all processes in turn a chance to enter the critical region if they want to. Whenever all processes are idle, the system may go at rest again, i.e., the token ring is put dormant.

A state diagram of P_i is shown in figure 1. Each process $P_i, i \in \{0, \ldots, n-1\}$ has four states: $\{idl_i, try_i, crit_i, fwd_i\}$. Process P_i has the token iff it is in one of its right-hand side states, $crit_i$ or fwd_i, respectively. A transition carrying a set of actions as a label is an abbreviation for a set of transitions between the same pair of states, one for each action in the set.

The token ring is established as follows. Each process P_i receives the token from its right-hand neighbor $P_{i+1 \bmod n}$ and sends it to its left-hand neighbor $P_{i-1 \bmod n}$. Transmitting the token is modelled by two actions for each process P_i: $t_{i+1 \bmod n}$ represents receiving the token, and t_i represents forwarding it. Of course, forwarding in P_i synchronizes with receiving in $P_{i-1 \bmod n}$, represented by sharing the same action t_i.

On receiving the token, a process enters the critical region if it wants to, i.e., if it is in its *try* state. Otherwise, i.e., if it is in its *idl* state, it simply forwards the token, moving to the *fwd* state when receiving the token and back to the *idl* state when forwarding it. The token is also forwarded when the process leaves the critical region, going into its *idl* state. The process may move from its *idl* state to its *try* state any time without synchronizing with other processes, indicating its spontaneous desire to enter the critical region.

When the system is at rest, one or more processes may spontaneously enter their *try* states, and one of these will proceed into the critical region, making sure by direct communication with all the others that they keep away. At the same time, the token ring is put into operation. Whenever a process terminates its critical region or forwarding state and all others are idle, the system may go at rest again, i.e., all processes are idle and the token ring is dormant. This is the case when the last active process P_i chooses its reset transitions r_i, synchronizing with the r_j reset transitions of all other processes $j \neq i$. The latter are only applicable in their *idl* states, ensuring that all processes are in their *idl* states after the move.

For the sake of simplicity, each process synchronizes with all the others when entering the critical region, also when the token ring is active when this is not really nec-

essary. In our solution shown in figure 1, this is accomplished by the loops at the *idl* and *try* states, synchronizing with token actions of all the other processes. Also, we abstain from enforcing that the system goes at rest again as soon as it can, the token ring may nondeterministically remain active for a while (or even forever).

We concentrate on checking safety and liveness.

safety
$P_i.\,\mathsf{A}\,\mathsf{G}(\mathit{crit}_i \;\Rightarrow\; \bigwedge_{j=1,j\neq i}^{n} P_j.\,\neg\,\mathit{crit}_j)$
for all $i \in \{0,\ldots,n-1\}$

liveness
$P_i.\,\mathsf{A}\,\mathsf{G}(\mathit{try}_i \;\Rightarrow\; \mathsf{A}\,\mathsf{F}\,\mathit{crit}_i)$ for all $i \in \{0,\ldots,n-1\}$

The assertions are written in the distributed version of CTL that we introduce in the next section. It should be intuitively clear what they mean. The liveness conditions are actually entirely local—which does not mean that they can be checked locally without considering communication with other processes. The i-th safety condition makes an assertion about all the other processes: they must not be in the critical region when the i-th process is.

In order to check safety, and assuming that the communication is correct, all we have to do is to prove the following.

$$P_i.(\mathit{crit}_i \;\Rightarrow\; t_{i+1 \bmod n})$$
for all $i \in \{0,\ldots,n-1\}$, and
$$P_j.(t_{i+1 \bmod n} \;\Rightarrow\; \neg\,\mathit{crit}_j)$$
for all $j \neq i$, $j \in \{0,\ldots,n-1\}$

The first condition says that P_i cannot be in the critical region unless it has been entered via action $t_{i+1 \bmod n}$. The second condition says that if $t_{i+1 \bmod n}$ has happened, no other process can be in the critical region. These conditions are local, they are obviously satisfied by our solution.

Liveness can be proved under the assumption that no process stays forever in the critical region: if, say, P_i leaves it, $P_{i-1 \bmod n}$ has the chance to enter. This way, each process will enter the critical region some time after it expressed its desire to do so by entering its *try* state.

We note in passing that also other correctness criteria (cf. [HR00]) are satisfied by our solution: *non-blocking* (a process can always request to enter the critical region) and *no-strict-sequencing* (processes need not enter the critical region in strict sequence).

It remains to prove that communication indeed works correctly. What must be shown is the following.

$$P_i.(a_i \;\Rightarrow\; P_j.a_i) \text{ and}$$
$$P_j.(a_i \;\Rightarrow\; P_i.a_i) \text{ for } i,j \in \{0,\ldots,n-1\}.$$

where a may stand for t or r: each token or reset action synchronizes with its counterparts in the other processes. For $i = j$, the formulae are trivial. The precise meaning of these formulae will be made clear in the next section.

The reader is invited to contrast this solution with the one-observer-outside-the-system approach: it would start by building the product state machine (or a state machine close to it). In this example, the arsenal of model checking techniques is certainly able to cope with the big state spaces as resulting from large n. But still: our method avoids building those big state spaces in the first place. However, the extra expense of introducing communication among observers must be taken into account.

3 Multi-observer assertions

Let I be a finite set of observers (or, rather, observer identifiers). Each observer $i \in I$ has an individual set A_i of action symbols expressing which action happens during a given transition, and an individual set Σ_i of state predicate symbols expressing which attributes have which values in a given state. Let $\Sigma = (I, \{A_i, \Sigma_i\}_{i\in I})$ be the observer signature. Our multi-observer logic is like the distributed logic D_1 described in [ECSD98,EC00] but instantiated with CTL, the computation tree logic due to E. Clarke and A.E. Emerson [CE81]. This logic $\mathsf{D}_1(\Sigma)$—or D_1 for short when Σ is clear from context—is defined as follows.

Definition 1. The *observer logic* D_1 is the I-indexed family of local logics $\{\mathsf{D}_1^i\}_{i\in I}$ where, for each $i \in I$, D_1^i is the set of formulae syntactically defined as follows.

$$\begin{aligned}
\mathsf{D}_1^i &::= i.\mathsf{H}_1^i \\
\mathsf{H}_1^i &::= A_i \mid \Sigma_i \mid \bot \mid \neg\mathsf{H}_1^i \mid \mathsf{H}_1^i \wedge \mathsf{H}_1^i \mid \\
&\qquad \mathsf{E}[\mathsf{H}_1^i\,\mathsf{U}\,\mathsf{H}_1^i] \mid \mathsf{A}\,\mathsf{F}\,\mathsf{H}_1^i \mid \mathsf{E}\,\mathsf{X}\,\mathsf{H}_1^i \mid \mathsf{CC}_1^i \\
\mathsf{CC}_1^i &::= \cdots \mid \mathsf{D}_1^j \mid \ldots \quad (j \in I, j \neq i)
\end{aligned}$$

H_1^i is the *home logic* of observer $i \in I$. It consists of CTL, presented by a minimal adequate set from which the other CTL formulae can be derived (cf. [HR00]), and enriched by *communication subformulae* CC_1^i which are home logic formulae of some other observer. Communication subformulae may be nested to any depth.

A given D_1 formula $i.(\ldots j.\psi \ldots)$ with a communication subformula $j.\psi \in \mathsf{D}_1^j, j \neq i$, means that i communicates—or synchronizes—with j and asserts that ψ is true for j at this moment of synchronization. The interpretation is more formally defined below.

Thus, each D_1 formula is bound to an observer, and each formula of another observer may serve as a subformula, representing communication with that observer.

For instance, the safety and liveness formulae in example 1 are in D_1. In the safety condition

$$P_i.\,\mathsf{A}\,\mathsf{G}(\mathit{crit}_i \;\Rightarrow\; \bigwedge_{j=1,j\neq i}^{n} P_j.\,\neg\,\mathit{crit}_j),$$

the subformulae $P_j.\,\neg\,\mathit{crit}_j, j \in \{1,\ldots,n\}, j \neq i$, are communication subformulae. The liveness formulae do not have communication subformulae, they are entirely local.

As shown in example 1 above, these assertions are transformed to purely local formulae (to be model checked) and primitive communication formulae (to be guaranteed otherwise),

$$\begin{array}{l} P_i.(a_i \;\Rightarrow\; P_j.a_i) \text{ and} \\ P_j.(a_i \;\Rightarrow\; P_i.a_i) \text{ for } i,j \in \{0,\ldots,n-1\}. \end{array}$$

All these formulae are in D_1. The latter communication subformulae have a particular practical relevance: they characterize in an abstract form RPC, the basic middleware communication mechanism. Formulae of this form therefore merit particular attention, they constitute the sublogic D_0 which we define in the next section.

First we give a more detailed and more precise definition of D_1 semantics. We assume a family $\mathcal{M} = \{\mathcal{M}_i\}_{i\in I}$ of models to be given, one for each observer, where $\mathcal{M}_i = (S_i, \rightarrow_i, L_i), i \in I$. For each observer i, S_i is its set of states, $\rightarrow_i \subseteq S_i \times A_i \times S_i$ is its action-labelled state transition relation, and $L_i : S_i \rightarrow_i 2^{\Sigma_i}$ is its state labelling function. We write $s \stackrel{a}{\rightarrow}_i t$ for $(s,a,t) \in \rightarrow_i$. As in example 1, we may extend the model to allow for finite sets B of actions as transition labels: $s \stackrel{B}{\rightarrow} s' \;\Leftrightarrow\; \forall a \in B : s \stackrel{a}{\rightarrow} s'$.

We assume that the reader is familiar with the conventional CTL semantics (cf., e.g., [HR00]): it defines the meaning of $\mathcal{M}_i, s_i \models_i \varphi$ for every observer i where $s_i \in S_i$ and $\varphi \in \mathsf{D}_1^i$, as long as φ does not contain any communication subformula—and as long as there are no action symbols, i.e. $|A_i| = 1$ for every observer i.

Action symbols are easily introduced by "pushing them into the states": given a model $\mathcal{M} = (S, \rightarrow, L)$ where $\rightarrow \subseteq S \times A \times S$ and $L : S \rightarrow 2^{\Sigma}$, we define the model $\mathcal{M}' = (S', \rightarrow', L')$, $\rightarrow' \subseteq S' \times S', L' : S' \rightarrow 2^{A_i \times \Sigma_i}$, by

$$\begin{array}{rcl} S' & = & S \times A \times S \\ (s_1 \stackrel{a_1}{\rightarrow} s_2) \rightarrow' (s_3 \stackrel{a_2}{\rightarrow} s_4) & \text{iff} & s_2 = s_3 \\ L'(s_1 \stackrel{a}{\rightarrow} s_2) & = & a \wedge L(s_2) \end{array}$$

for all $s_1, s_2, s_3, s_4 \in S'$ and all $a, a_1, a_2 \in A$. Of course, $s \rightarrow' t$ stands for $(s,t) \in \rightarrow'$. We use action symbols a also as action occurrence predicates meaning that action a occurs during a transition (in $\mathcal{M}$) or during a state (in $\mathcal{M}'$), respectively.

Our construction is different from the ACTL (action based version of CTL) construction given in [DV90] where a new state s_a is introduced on each transition labelled a. Our construction gives us states with action symbols and state predicates mixed; this is essential for the translation of D_1 to D_0 as described in the following section.

Capturing communication is not quite so easy. Note that with treating the actions as described above, we now synchronize states rather than transitions. However, whether a state of one observer synchronizes with a given state of another one does not only depend on the current state of the former but on what happened before in the latter. For instance, in example 1, if one process leaves the critical region, it synchronizes with all the others where some are in their *idl* state and some are in their *try* state, depending on local conditions. When the same process leaves the critical region again later on, it may synchronize with quite a different set of states in the others.

Therefore, we must redefine semantics in terms of *events* rather than states. Intuitively, an event is the transition through a state in a life cycle starting with an initial state. Thus, events carry the entire execution history and are unique during a life cycle. For the sake of simplicity, we assume that each model $\mathcal{M}_i$ has a unique initial state $s_i^0 \in S_i$. Formally, an event e_i in $\mathcal{M}_i$ is defined to be a path $e_i = s_0^i \rightarrow_i s_1^i \rightarrow_i \cdots \rightarrow_i s_{m_i}^i$ where $s_k^i \in S_i$ for $k \in \{1,\ldots,m\}$. For the conventional case without communication, we have

$$\mathcal{M}_i, e_i \models_i \varphi \;\text{ iff }\; \mathcal{M}_i, s_{m_i}^i \models_i \varphi,$$

i.e., the last state of the event satisfies φ.

The communication case can now be defined by the rule

$$\begin{array}{l} \mathcal{M}_i, e_i \models_i j.\varphi \;\text{ iff }\; \mathcal{M}_j, e_j \models_j \varphi \\ \qquad\qquad \text{for that event } e_j \text{ such that } e_j \equiv e_i \end{array}$$

where $i,j \in \{1,\ldots,n\}$, $i \neq j$. The relation $\equiv$ formalizes communication in the sense of event synchronization: $\equiv$ is a global equivalence relation on the events of all local models where no two events of the same local model are equivalent. Therefore, the e_j satisfying $e_j \equiv e_i$ is uniquely determined. Furthermore, if $e_i' \leq e_i$ (i.e., e_i' is a prefix of e_i), and $e_i' \equiv e_j'$ and $e_i \equiv e_j$, then $e_j < e_j'$ would constitute a "causal loop" and is therefore not allowed. Another way of putting this is that the set of all events factorized by $\equiv$ is a partial ordering with respect to causality $\leq$.

More details can be found in [EC00] where the semantics of D_1 is given in terms of event structures (albeit for a linear-time local logic). As for an introduction to

event structures and their relationship to other models of concurrency, we refer to [WN95].

4 Distributing assertions over observers

Now we demonstrate how D_1 formulae can be translated to D_0 formulae in a sound and complete way where $\mathsf{D}_0 \subseteq \mathsf{D}_1$ is an observer sublogic featuring a basic RPC-like communication mechanism. Full details of the translation as well as soundness and completeness proofs are given in [EC00]. We first introduce D_0, outline the translation rules, and show how the translation can be utilized to transform one-observer assertions (in D_1) into multiple-observer assertions (in D_0). This way, the communication primitives and the partial assertions for the observers involved are separated so that they together guarantee the original assertion.

For defining D_0, we assume again that a family of observer signatures $\Sigma = (I, \{A_i, \Sigma_i\}_{i\in I})$ is given which we do not show explicitly in our notation.

Definition 2. The *observer sublogic* $\mathsf{D}_0 \subseteq \mathsf{D}_1$ is the I-indexed family of local logics $\{\mathsf{D}_0^i\}_{i\in I}$ where, for each $i \in I$, D_0^i is the set of formulae syntactically defined as follows.

$$\begin{aligned}
\mathsf{D}_0^i &::= i.\mathsf{H}_0^i \mid i.\mathsf{CC}_0^i \\
\mathsf{H}_0^i &::= @I \mid A_i \mid \Sigma_i \mid \bot \mid \neg\mathsf{H}_0^i \mid \mathsf{H}_0^i \wedge \mathsf{H}_0^i \mid \\
&\qquad \mathsf{E}[\mathsf{H}_0^i \,\mathsf{U}\, \mathsf{H}_0^i] \mid \mathsf{A}\,\mathsf{F}\,\mathsf{H}_0^i \mid \mathsf{E}\,\mathsf{X}\,\mathsf{H}_0^i \\
\mathsf{CC}_0^i &::= \cdots \mid (A_i \Rightarrow j.A_j) \mid \ldots \quad (j \in I, j \neq i)
\end{aligned}$$

The predicates $i.@j$ mean the same as $i.j.\neg\bot$ in D_1, namely that i communicates with j. But this must be given an extra symbol in D_0 because $i.j.\neg\bot$ is not a formula in D_0. Note that $i.@i$ is trivially true: each observer synchronizes with itself all the time.

D_0^i is the logic at observer i with "action callings" to other observers. H_0^i is the home logic of observer i, it is precisely CTL with the underlying state predicates as given. CC_0^i defines the primitive communication formulae expressible in observer i. Note that communication formulae are stand-alone, they may not be embedded as subformulae of other formulae.

Examples of D_0 formulae have been given above. For the straightforward proof that D_0 is indeed a sublogic of D_1, we refer to [EC00].

We note in passing that it would be obvious to generalize D_0 so as to include Σ_i state predicates in the communication formulae: allowing for predicates of the form *attribute* $=$ *value* would express communication by shared variables.

The idea of the translation is as follows. Let an assertion $\varphi \in \mathsf{D}_1^i$ be given saying, somewhere in context, that an assertion ψ holds for j,

$$\varphi \;\Leftrightarrow\; i.(\ldots j.\psi \ldots),$$

where $j.\psi$ is in D_0, i.e., ψ does not have a communication subformula. The meaning is that i transmits a "message" q to j,

$$\overline{\varphi} \;\Leftrightarrow i.(\ldots q \ldots),$$

that is equivalent in j to the validity of ψ during communication with i,

$$\delta \;\Leftrightarrow j.(q \Leftrightarrow @i \wedge \psi)$$

q is a new action symbol in i as well as in j, i.e., there are local actions $i.q$ in i and $j.q$ in j "calling" each other. This communication is captured by

$$\begin{aligned}
\alpha &\;\Leftrightarrow i.(q \Rightarrow j.q) \\
\beta &\;\Leftrightarrow j.(q \Rightarrow i.q)
\end{aligned}$$

The resulting formulae δ, α and β are in D_0. $\overline{\varphi}$ may still be in $\mathsf{D}_1 - \mathsf{D}_0$, but the number of communication subformulae has decreased by one. With α and β as defined above, the translation step is sound (φ implies $\overline{\varphi}$ and δ) and complete ($\overline{\varphi}$ and δ imply φ).

The translation of arbitrary D_1 formulae is accomplished by iterating the above step inside–out until no communication subformulae are left. Together with the corresponding RPC-like communication formulae like α and β, the communication infrastructure is established for reducing global to local model checking. Clearly, the translation process terminates. Its soundness and completeness are proved in [EC00].

Note that new action symbols are introduced along with the translation. However, "new" in this context means that the symbols have not been in the formula before but may very well have been in the object signature. In practice where we have implemented the system and want to check assertions in retrospect, this means that we have to look for suitable communication symbols that are already implemented. We elaborate on this idea in the following example where we use mutual exclusion (example 1) again for illustration.

Example 2 (Mutual Exclusion cont.). First we concentrate on safety. In order to keep the example small but not too trivial, we take $n = 3$. The table in figure 2 shows the local formulae and communication primitives as generated in iterations 0 to 8.

Iteration ...

0 states the safety formula for P_0;
1 replaces the 1st communication subformula by q_{01} and defines q_{01} for P_1;
2 replaces the 2nd communication subformula by q_{02} and defines q_{02} for P_2.

iteration	observer	local formulae	communication
0	P_0	$\mathsf{A}\,\mathsf{G}(\mathit{crit}_0 \Rightarrow P_1.\neg\mathit{crit}_1 \wedge P_2.\neg\mathit{crit}_2)$	
1	P_0	$\mathsf{A}\,\mathsf{G}(\mathit{crit}_0 \Rightarrow q_{01} \wedge P_2.\neg\mathit{crit}_2)$	$q_{01} \Rightarrow P_1.q_{01}$
	P_1	$q_{01} \Leftrightarrow @P_0 \wedge \neg\mathit{crit}_1$	$q_{01} \Rightarrow P_0.q_{01}$
2	P_0	$\mathsf{A}\,\mathsf{G}(\mathit{crit}_0 \Rightarrow q_{01} \wedge q_{02})$	$q_{02} \Rightarrow P_2.q_{02}$
	P_2	$q_{02} \Leftrightarrow @P_0 \wedge \neg\mathit{crit}_2$	$q_{02} \Rightarrow P_0.q_{02}$
3	P_1	$\mathsf{A}\,\mathsf{G}(\mathit{crit}_1 \Rightarrow P_0.\neg\mathit{crit}_0 \wedge P_2.\neg\mathit{crit}_2)$	
4	P_1	$\mathsf{A}\,\mathsf{G}(\mathit{crit}_1 \Rightarrow q_{10} \wedge P_2.\neg\mathit{crit}_2)$	$q_{10} \Rightarrow P_0.q_{10}$
	P_0	$q_{10} \Leftrightarrow @P_1 \wedge \neg\mathit{crit}_0$	$q_{10} \Rightarrow P_1.q_{10}$
5	P_1	$\mathsf{A}\,\mathsf{G}(\mathit{crit}_1 \Rightarrow q_{10} \wedge q_{12})$	$q_{12} \Rightarrow P_2.q_{12}$
	P_2	$q_{12} \Leftrightarrow @P_1 \wedge \neg\mathit{crit}_2$	$q_{12} \Rightarrow P_1.q_{12}$
6	P_2	$\mathsf{A}\,\mathsf{G}(\mathit{crit}_2 \Rightarrow P_0.\neg\mathit{crit}_0 \wedge P_1.\neg\mathit{crit}_1)$	
7	P_2	$\mathsf{A}\,\mathsf{G}(\mathit{crit}_2 \Rightarrow q_{20} \wedge P_1.\neg\mathit{crit}_1)$	$q_{20} \Rightarrow P_0.q_{20}$
	P_0	$q_{20} \Leftrightarrow @P_2 \wedge \neg\mathit{crit}_0$	$q_{20} \Rightarrow P_2.q_{20}$
8	P_2	$\mathsf{A}\,\mathsf{G}(\mathit{crit}_2 \Rightarrow q_{20} \wedge q_{21})$	$q_{21} \Rightarrow P_1.q_{21}$
	P_1	$q_{21} \Leftrightarrow @P_2 \wedge \neg\mathit{crit}_1$	$q_{21} \Rightarrow P_2.q_{21}$

Fig. 2. Safety conditions transformed to D_0

The remaining iterations do the same for P_1 introducing communication symbols q_{10} and q_{12}, and for P_2 introducing communication symbols q_{20} and q_{21}.

observer	local formulae	communication
P_0	$\mathsf{A}\,\mathsf{G}(\mathit{crit}_0 \Rightarrow t_1)$	$t_1 \Rightarrow P_1.t_1$
		$t_1 \Rightarrow P_2.t_1$
	$t_2 \Leftrightarrow @P_1 \wedge \neg\mathit{crit}_0$	$t_2 \Rightarrow P_1.t_2$
	$t_0 \Leftrightarrow @P_2 \wedge \neg\mathit{crit}_0$	$t_0 \Rightarrow P_2.t_0$
P_1	$\mathsf{A}\,\mathsf{G}(\mathit{crit}_1 \Rightarrow t_2)$	$t_2 \Rightarrow P_2.t_2$
		$t_2 \Rightarrow P_0.t_2$
	$t_0 \Leftrightarrow @P_2 \wedge \neg\mathit{crit}_1$	$t_0 \Rightarrow P_2.t_0$
	$t_1 \Leftrightarrow @P_0 \wedge \neg\mathit{crit}_1$	$t_1 \Rightarrow P_0.t_1$
P_2	$\mathsf{A}\,\mathsf{G}(\mathit{crit}_2 \Rightarrow t_0)$	$t_0 \Rightarrow P_0.t_0$
		$t_0 \Rightarrow P_1.t_0$
	$t_1 \Leftrightarrow @P_0 \wedge \neg\mathit{crit}_2$	$t_1 \Rightarrow P_0.t_1$
	$t_2 \Leftrightarrow @P_1 \wedge \neg\mathit{crit}_2$	$t_2 \Rightarrow P_1.t_2$

Fig. 3. Optimized safety conditions

Now we optimize the local conditions and communication formulae. We do this by inspection since there is no general technique (yet). If we compare the communication primitives in figure 2 with the communication symbols introduced in example 1, we see that $q_{i(i+1 \bmod 3)}$ and $q_{i(i-1 \bmod 3)}$ match well with $t_{i+1 \bmod 3}$ for $i \in \{0,1,2\}$: the former two can be safely identified because they always occur together in observer i, and they can be identified with the latter because they synchronize with the other two observers in the same way.

As a result, only the local and communication formulae for each process shown in figure 3 must be checked in order to guarantee safety. If we assume that communication works correctly (or it is not *our* business to prove this) we only have to verify the local formulae in the middle column of figure 3. Their validity is obvious in this case.

observer	local formulae	communication
P_0	$\mathsf{A}\,\mathsf{G}(\mathit{try}_0 \Rightarrow \mathsf{A}\,\mathsf{F}\,t_1)$	$t_1 \Rightarrow P_1.t_1$
		$t_1 \Rightarrow P_2.t_1$
	$\mathsf{A}\,\mathsf{G}(t_1 \Rightarrow \mathsf{A}\,\mathsf{F}\,t_0)$	$t_0 \Rightarrow P_1.t_0$
		$t_0 \Rightarrow P_2.t_0$
P_1	$\mathsf{A}\,\mathsf{G}(\mathit{try}_1 \Rightarrow \mathsf{A}\,\mathsf{F}\,t_2)$	$t_2 \Rightarrow P_2.t_2$
		$t_2 \Rightarrow P_0.t_2$
	$\mathsf{A}\,\mathsf{G}(t_2 \Rightarrow \mathsf{A}\,\mathsf{F}\,t_1)$	$t_1 \Rightarrow P_2.t_1$
		$t_1 \Rightarrow P_0.t_1$
P_2	$\mathsf{A}\,\mathsf{G}(\mathit{try}_2 \Rightarrow \mathsf{A}\,\mathsf{F}\,t_0)$	$t_0 \Rightarrow P_0.t_0$
		$t_0 \Rightarrow P_1.t_0$
	$\mathsf{A}\,\mathsf{G}(t_0 \Rightarrow \mathsf{A}\,\mathsf{F}\,t_2)$	$t_2 \Rightarrow P_0.t_2$
		$t_2 \Rightarrow P_1.t_2$

Fig. 4. Liveness conditions

For proving liveness, we have to check $P_i.\mathsf{A}\,\mathsf{G}$ $(\mathit{try}_i \Rightarrow \mathsf{A}\,\mathsf{F}\,\mathit{crit}_i)$ for $i \in \{0,1,2\}$. As mentioned before, we have to make the obvious assumption that no process stays forever in the critical region (or in any of its states) without making a transition. The proof idea is simple: if, say, P_i leaves the critical region, $P_{i-1 \bmod n}$ has the chance to enter; by this circular token passing mechanism, each process has a chance to enter the critical re-

gion when it is its turn. Formally, what must be checked locally and with respect to communication is shown in the table in figure 4. The validity of these conditions is obvious in this case.

5 Concluding remarks

We have shown by example how model checking via multiple concurrent interacting observers works in principle. We also have outlined the underlying theory of distributed logics which is understood to some extent but needs much further study. The investigation of the idea presented in this paper is at its beginning, further study is necessary in order to explore how the idea can best be utilized in model checking technology. There are essentially two things that need to be done: (1) developing the technique, especially with respect to optimization as indicated in example 2, and (2) finding out how the method can best be combined with currently available techniques. Among the former issues, the right number of observers, their distribution and their location inside or outside the system deserves further study. Also, appropriate tools have to be developed.

One promise of our approach is that state explosion may be considerably reduced, at least in some cases. However, there is a price to pay: state explosion has to be balanced with "communication explosion" among observers as introduced by our method.

But here we also see another promise: our approach separates communication from correctness issues. And a kind of communication among observers is introduced that corresponds with the basic RPC-like mechanism used in current middleware. Indeed, checking an object system built on top of a standard communication infrastructure should not involve checking this infrastructure—somebody else has to take responsibility for its relyability and correctness.

Although we described our approach for communicating object systems, we think that it would be possible to adapt it to communicating agents or to multi agents as well. This would be another subject of further study.

Distributed logics as well as their model theory are not easy (this holds in particular for their proof theory which we do not touch in this paper). As for model theory, we encountered the need to abandon the usual interpretation structures, transition systems, in favor of event structures because otherwise communication cannot be captured adequately.

The original motivation for studying D_1 and D_0 and their translation was to use them for modeling and specification [ECSD98]: D_1 is more suitable for specification, and D_0 can be more easily implemented, e.g., translated to C^{++} or JAVA and ORACLE database schemata. We use this in the TROLL language and method for constructing information systems developed in our group [HDK$^+$97]. Translation to C^{++} and ORACLE database schemata has been implemented for animation purposes [GKK$^+$98].

References

[BGS98] M. Benerecetti, F. Giunchiglia, and L. Serafini. Model Checking Multiagent Systems. *Journal of Logic and Computation*, 8(3):373–400, 1998.

[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.

[DV90] Rocco De Nicola and Frits Vaandrager. Action versus State based Logics for Transition Systems. In I. Guessarian, editor, *Semantics of Concurrency*, volume 469 of *Lecture Notes in Computer Science*, pages 407–419, 1990.

[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.

[GKK$^+$98] A. Grau, J. Küster Filipe, M. Kowsari, S. Eckstein, R. Pinger, and H.-D. Ehrich. The TROLL Approach to Conceptual Modelling: Syntax, Semantics and Tools. In T.W. Ling, S. Ram, and M.L. Lee, editors, *Proc. of the 17th Int. Conference on Conceptual Modeling (ER'98), Singapore*, pages 277–290. Springer, LNCS 1507, November 1998.

[HDK$^+$97] P. Hartel, G. Denker, M. Kowsari, M. Krone, and H.-D. Ehrich. Information systems modelling with TROLL formal methods at work. In *Information Systems*, volume 22, pages 79–99, 1997.

[HR00] Michael R. A. Huth and Mark D. Ryan. *Logic in Computer Science - Modelling and reasoning about systems*. Cambridge University Press, 2000.

[WN95] G. Winskel and M. Nielsen. Models for concurrency. In S. Abramsky, D. M. Gabbay, and T. S. E. Maibaum, editors, *Handbook of Logic in Computer Science*, volume 4, pages 1–148. Clarendon Press, 1995.