

# Verifying Interlevel Relations within Multi-Agent Systems

Alexei Sharpanskykh and Jan Treur<sup>1</sup>

**Abstract.** An approach to handle the complex dynamics of a multi-agent system is based on distinguishing aggregation levels by structuring the system into parts or components. The behavior of every aggregation level is specified by a set of dynamic properties for components and interactions at that level, expressed in some (temporal) language. The dynamic properties of higher aggregation levels in principle can be logically related to dynamic properties of lower levels. This asks for identification and verification of such interlevel relations. In this article it is shown how this problem can be addressed using model checking techniques.

## 1 INTRODUCTION

Often dynamics of a multi-agent system is described by a behavioral specification, which consists of dynamic properties of elements of the multi-agent system (i.e., agents, interaction relations, and an environment). Usually, these properties are expressed as formulae in some (temporal) language. Even if the behavioral description of a single element is simple, the general dynamics of the whole multi-agent system is often difficult to analyze. In particular, with increase of the number of elements within the multi-agent system, the complexity of the dynamics of the system grows considerably. In order to analyze the behavior of a complex multi-agent system (e.g., for critical domains such as air traffic control and health care), appropriate approaches for handling the dynamics of the multi-agent system are important.

One of the approaches to manage complex dynamics is by distinguishing different *aggregation levels*, based on a clustering of a multi-agent system into parts or components with further specification of their dynamics and relations between them; e.g., [4]. At the lowest aggregation level a component is an agent or an environmental object (e.g., a database), with which agents interact. Further, at higher aggregation levels a component has the form of either a group of agents or a multi-agent system as a whole. In the simplest case two levels can be distinguished: the lower level at which agents interact and the higher level, where the whole multi-agent system is considered as one component. In the general case the number of aggregation levels is not restricted. At every aggregation level the behavior of a component is described by a set of dynamic properties. The dynamic properties of a component of a higher aggregation level can be logically related by an *interlevel relation* to dynamic properties of components of an adjacent lower

aggregation level. This interlevel relation takes the form that a number of properties of the lower level logically entail the properties of the higher level component.

Identifying interlevel relations is usually achieved by applying informal or semi-formal early requirements engineering techniques; e.g., *i\** [8] and SADT [7]. To formally prove that the identified interlevel relations are indeed correct, model checking techniques [1, 9] may be of use. The idea is that the lower level properties in an interlevel relation are used as a system specification, whereas the higher level properties are checked for this system specification. However, model checking techniques are only suitable for systems specified as finite-state concurrent systems. In the general case, at any aggregation level a behavioral specification for a multi-agent system component consists of dynamic properties expressed by possibly complex temporal relations, which do not allow direct application of automatic model checking procedures. In order to apply model checking techniques it is needed to transform an original behavioral specification of the lower aggregation level into a model based on a finite state transition system. In order to obtain this, as a first step a behavioral description for the lower aggregation level is replaced by one in executable temporal format. After that, using an automated procedure an executable temporal specification is translated into a general finite state transition system format that consists of standard transition rules. Such a representation can be easily translated into an input format of one of the existing model checkers. Then, using model checking techniques it is possible to prove (or refute) automatically that the interlevel relations between dynamic properties of adjacent aggregation levels expressed as specification in some temporal language hold.

The proposed approach has similarities with compositional reasoning and verification techniques [3, 4] in the way how it handles complex dynamics of a system. However, it differs from the latter in distinguishing multiple aggregation levels in a system specification and representing verification of system dynamics as a problem of checking logic entailment relations between dynamic properties of adjacent aggregation levels. Other approaches for verifying multi-agent system dynamics are based on modal temporal logics, automatically translated into executable format by techniques from [2], and verified by clausal resolution methods.

In the next section the concepts for formal specification of a multi-agent system's dynamics and the temporal language used are briefly introduced. After that, in Section 3 the transformation procedure from a behavioral specification into a finite state transition system description is described and illustrated by means of an example. The paper ends with a discussion in Section 4.

---

<sup>1</sup> Vrije Universiteit Amsterdam, Dept of AI, De Boelelaan 1081a, 1081 HV Amsterdam, The Netherlands. E-mail: {sharp, treur}@few.vu.nl URL: <http://www.few.vu.nl/~{sharp,treur}>

## 2 MODELING OF DYNAMIC PROPERTIES

Components can be active (e.g., an agent) or passive (e.g., database). An active component represents an autonomous entity that interacts with an environment and with other components. Active components interact with each other by communicating messages to each other (at the mental level) and performing actions (at the physical level). An environment can be considered as a set of passive components with certain properties and states. Components interact with each other via *input* and *output* (interface) states. At its input an active component receives observations from an environment or communications from other components whereas at its output it generates communications to other components or actions in an environment.

From the external perspective behavior of a component is characterized by a set of dynamic properties, which represent relations over time between its input and output states. Dynamic properties are specified in the Temporal Trace Language (TTL) [4, 11]. TTL is a variant of order-sorted predicate logic [6] and has some similarities with Situation Calculus [10] and Event Calculus [5]. Whereas the standard multi-sorted predicate logic is a language to reason about static properties only, TTL is an extension of such language with facilities for reasoning about the dynamic properties of arbitrary systems.

The state properties are expressed using a standard multi-sorted first-order predicate language with a signature, which consists of a number of sorts, sorted constants, variables, functions and predicates. Every component  $A$  has assigned an interaction state ontology  $\text{InteractionOnt}(A)$  for its input and output states. Specifically, using an ontology  $\text{InteractionOnt}$  one can define observations of state properties, communications, and actions.

For enabling dynamic reasoning TTL includes special sorts:  $\text{TIME}$  (a set of linearly ordered time points),  $\text{STATE}$  (a set of all state names of a system),  $\text{TRACE}$  (a set of all trace names; a trace or a trajectory can be thought of as a timeline with a state for each time point),  $\text{STATPROP}$  (a set of all state property names), and  $\text{VALUE}$  (an ordered set of numbers). Furthermore, for every sort  $S$  from the state language the following TTL sorts exist: the sort  $S^{\text{VARS}}$ , which contains all variable names of sort  $S$ ; the sort  $S^{\text{GTERMS}}$ , which contains names of all ground terms, constructed using sort  $S$ ; sorts  $S^{\text{GTERMS}}$  and  $S^{\text{VARS}}$  are subsorts of sort  $S^{\text{TERMS}}$ .

In TTL, formulae of the state language are used as objects. To provide names of object language formulae  $\phi$  in TTL the operator  $(*)$  is used (written as  $\phi^*$ ), which maps variable sets, term sets and formula sets of the state language to the elements of TTL sorts  $S^{\text{GTERMS}}$ ,  $S^{\text{TERMS}}$ ,  $S^{\text{VARS}}$  and  $\text{STATPROP}$ . The state language and TTL define disjoint sets of expressions. Therefore, in TTL formulae we shall use the same notations for the elements of the object language (i.e., constants, variables, functions, predicates) and for their names in TTL without introducing any ambiguity. Further we shall use  $t$  with subscripts and superscripts for variables of the sort  $\text{TIME}$ ; and  $\gamma$  with subscripts and superscripts for variables of the sort  $\text{TRACE}$ .

For the explicit indication of an aspect of a state for a component, to which a state property is related, sorts  $\text{ASPECT\_COMPONENT}$  (a set of the component aspects of a system; i.e., input, output, internal);  $\text{COMPONENT}$  (a set of all component names of a system);  $\text{COMPONENT\_STATE\_ASPECT}$  (a set of all names of aspects of all component states) and a function symbol

$$\text{comp\_aspect: ASPECT\_COMPONENT} \times \text{COMPONENT} \rightarrow \text{COMPONENT\_STATE\_ASPECT}$$

are used.

A state for a component is described by a function symbol  $\text{state}$  of type  $\text{TRACE} \times \text{TIME} \times \text{COMPONENT\_STATE\_ASPECT} \rightarrow \text{STATE}$ .

The set of function symbols of TTL includes  $\wedge, \vee, \rightarrow, \leftrightarrow$ :  $\text{STATPROP} \times \text{STATPROP} \rightarrow \text{STATPROP}$ ;  $\text{not: STATPROP} \rightarrow \text{STATPROP}$ ,  $\forall, \exists$ :  $S^{\text{VARS}} \times \text{STATPROP} \rightarrow \text{STATPROP}$ , which counterparts in the state language are boolean propositional connectives and quantifiers. Further we shall use  $\wedge, \vee, \rightarrow, \leftrightarrow$  in infix notation and  $\forall, \exists$  in prefix notation for better readability.

Notice that also within states statements about time can be made (e.g., in state properties representing memory). To relate time within a state property (sort  $\text{LTIME}$ ) to time external to states (sort  $\text{TIME}$ ) a function  $\text{present\_time: LTIME}^{\text{TERMS}} \rightarrow \text{STATPROP}$  is used. Furthermore, for the purposes of this paper it is assumed that  $\text{LTIME}^{\text{GTERMS}} = \text{TIME}$  and  $\text{LVALUE}^{\text{GTERMS}} = \text{VALUE}$  ( $\text{LVALUE}$  is a sort of the state language, which is a set of numbers). We shall use  $u$  with subscripts and superscripts to denote constants of sort  $\text{LTIME}^{\text{VARS}}$ . For formalising relations between sorts  $\text{VALUE}$  and  $\text{TIME}$  function symbols  $-, +, /, \bullet$ :  $\text{TIME} \times \text{VALUE} \rightarrow \text{TIME}$  are introduced. And for sorts  $\text{LVALUE}^{\text{TERMS}}$  and  $\text{LTIME}^{\text{TERMS}}$  the function symbols  $-, +, /, \bullet$  are overloaded:  $\text{LTIME}^{\text{TERMS}} \times \text{LVALUE}^{\text{TERMS}} \rightarrow \text{STATPROP}$ .

The states of a component are related to names of state properties via the formally defined satisfaction relation denoted by the infix predicate  $|=$  (or denoted by the prefix predicate  $\text{holds}$ ):  $\text{state}(\gamma, t, \text{output}(A)) | = p$  (or  $\text{holds}(\text{state}(\gamma, t, \text{output}(A)))$ ), which denotes that the state property with a name  $p$  holds in trace  $\gamma$  at time point  $t$  at the output state of component  $A$ . Sometimes, when the indication of a component aspect is not necessary, this relation will be used without the third argument:  $\text{state}(\gamma, t) | = p$ . Both  $\text{state}(\gamma, t, \text{output}(A))$  and  $p$  are terms of TTL. In general, TTL terms are constructed by induction in a standard way from variables, constants and function symbols typed with all before mentioned TTL sorts.

Transition relations between states are described by dynamic properties, which are expressed by TTL-formulae. The set of *atomic TTL-formulae* is defined as:

- (1) If  $v_1$  is a term of sort  $\text{STATE}$ , and  $u_1$  is a term of the sort  $\text{STATPROP}$ , then  $\text{holds}(v_1, u_1)$  is an atomic TTL formula.
- (2) If  $\tau_1, \tau_2$  are terms of any TTL sort, then  $\tau_1 = \tau_2$  is an atomic TTL formula.
- (3) If  $t_1, t_2$  are terms of sort  $\text{TIME}$ , then  $t_1 < t_2$  is an atomic TTL formula.

The set of *well-formed TTL-formulae* is defined inductively in a standard way using boolean propositional connectives and quantifiers. TTL has semantics of the order-sorted predicate logic. A more detailed specification of the syntax and the semantics for the TTL (including the axiomatic basis) is given in [11].

Dynamic properties to model a behavioral specification are assumed to be specified in the form of a logical implication from a temporal input pattern to a temporal output pattern. The consequent parts of dynamic properties do not contain any disjunctions in order to prevent non-determinism in behavior. It is a necessary assumption for enabling verification of a system using existing model checking techniques and tools. Past, interval and future statements are defined as follows:

- a) A *past statement* for a trace  $\gamma$  and a time point  $t$  over state ontology  $\text{Ont}$  is a temporal statement  $\phi_p(\gamma, t)$  in TTL, such that each time variable  $s$  different from  $t$  is restricted to the time interval before  $t$ : for every time quantifier for a time variable  $s$  a restriction of the form  $s \leq t$ , or  $s < t$  is required within the statement.
- b) A *future statement* for a trace  $\gamma$  and a time point  $t$  over state ontology  $\text{Ont}$  is a temporal statement  $\phi_f(\gamma, t)$  in TTL, such that for every quantified time variable  $s$ , different from  $t$  a restriction of the form  $s \geq t$ , or  $s > t$  is required within the statement.

c) An *interval statement* for a trace  $\gamma$  and time points  $t_1$  and  $t_2$  over state ontology  $\text{Ont}$  is a temporal statement  $\phi(\gamma, t_1, t_2)$  in TTL, that is a past statement for  $t_2$  and a future statement for  $t_1$ .

An executable specification of the dynamics of a component consists of a set of dynamic properties in an executable temporal language, representing temporal relations between a number of postulated internal states. Internal states of a component  $A$  are described using a postulated internal state ontology  $\text{InternalOnt}(A)$ . In cognitive sciences, which have been used as a source of inspiration, it is often assumed that an agent maintains a memory in the form of some internal model of the history. Furthermore, we assume that internal states are formed on the basis of (input) observations (sensory representations) or communications. For this the function symbol  $\text{memory: LTIME}^{\text{TERMS}} \times \text{STATPROP} \rightarrow \text{STATROP}$  is used. For example,  $\text{memory}(t, \text{observed}(a))$  expresses that the component has memory that it observed a state property  $a$  at time point  $t$ . Before performing an action or communication it is postulated that a component creates an internal preparation state. For example,  $\text{preparation\_for}(b)$  represents a preparation of a component to perform an action or a communication  $b$ . Each dynamic property in the internal behavioral specification is specified in one of the following *executable* forms:

- (1)  $\forall t \text{ state}(\gamma, t) \models X \Rightarrow \text{state}(\gamma, t+c) \models Y$  (states relation property)
- (2)  $\forall t \text{ state}(\gamma, t) \models X \Rightarrow \text{state}(\gamma, t+1) \models X$  (persistence property)
- (3)  $\forall t \text{ state}(\gamma, t) \models X \Rightarrow \text{state}(\gamma, t) \models Y$  (state relation property), where  $c$  is some integer constant,  $X$  and  $Y$  are (conjunctions of) names of state properties and  $X \neq Y$ .

### 3 TRANSFORMATION PROCEDURE

The procedure described in this section achieves the transformation of a behavioral specification for multi-agent system components into the executable format and subsequently into the representation of a finite state transition system. An external behavioral specification of a component is defined as follows.

#### Definition (External behavioral specification)

An *external behavioral specification* for a multi-agent system component consists of dynamic properties  $\phi(\gamma, t)$  expressed in TTL of the form  $[\phi_p(\gamma, t) \Rightarrow \phi_f(\gamma, t)]$ , where  $\phi_p(\gamma, t)$  is a past statement over the interaction ontology and  $\phi_f(\gamma, t)$  is a future statement. The future statement is represented in the form of a conditional behavior:  $\phi(\gamma, t) \Leftrightarrow \forall t_1 > t [\phi_{\text{cond}}(\gamma, t, t_1) \Rightarrow \phi_{\text{bh}}(\gamma, t_1)]$ , where  $\phi_{\text{cond}}(\gamma, t, t_1)$  is an interval statement over the interaction ontology, which describes a condition for some specified action(s) and/or communication(s), and  $\phi_{\text{bh}}(\gamma, t_1)$  is a (conjunction of) future statement(s) for  $t_1$  over the output ontology of the form  $\text{state}(\gamma, t_1+c) \models \text{output}(a)$ , for some integer constant  $c$  and action or communication  $a$ .

When a past formula  $\phi_p(\gamma, t)$  is true for  $\gamma$  at time  $t$ , a potential to perform one or more action(s) and/or communication(s) exists. This potential is realized at time  $t_1$  when the condition formula  $\phi_{\text{cond}}(\gamma, t, t_1)$  becomes true, which leads to the action(s) and/or communication(s) being performed at the time point(s)  $t_1+c$  indicated in  $\phi_{\text{bh}}(\gamma, t_1)$ .

The procedure is applied to the multi-agent system considered at any two adjacent aggregation levels. The behavioral specification of the lower aggregation level is constructed from the dynamic properties of the lower level components. By interlevel relations these properties can be logically related to the properties of higher level components. In order to enable automatic verification of such interlevel relations, the non-executable lower level behavioral specification is automatically replaced by an

executable one. Further, for performing model checking, the executable specification is automatically transformed into a finite state transition system description.

Let  $\phi(\gamma, t)$  be a non-executable dynamic property from an external behavioral specification for the component  $A$ , for which an executable representation should be found.

#### The Transformation Procedure

- (1) Identify executable temporal properties, which describe transitions from interaction states to memory states.
- (2) Identify executable temporal properties, which describe transitions from memory states to preparation states for output.
- (3) Specify executable properties, which describe the transition from preparation states to the corresponding output states.
- (4) From the executable properties, identified during steps 1-3, construct a part of the specification  $\pi(\gamma, t)$ , which describes the internal dynamics of component  $A$ , corresponding to property  $\phi(\gamma, t)$ .
- (5) Apply steps 1-4 to all properties in the behavioral specification of lower level components  $A$ . In the end add to the executable specification the dynamic properties, which were initially specified in executable form using an ontology, different than  $\text{InteractOnt}(A)$ .
- (6) Translate the identified during the steps 1-5 executable rules into the transition system representation.

The details of the described procedure are described by means of an example, in which a multi-agent system for co-operative information gathering is considered at two aggregation levels. At the higher level the multi-agent system as a whole is considered. At the lower level four components and their interactions are considered: two information gathering agents  $A$  and  $B$ , agent  $C$ , and environment component  $E$  representing the external world. Each of the agents is able to acquire partial information from an external source (component  $E$ ) by initiated observations. Each agent can be reactive or proactive with respect to the information acquisition process. An agent is proactive if it is able to start information acquisition independently of requests of any other agents, and an agent is reactive if it requires a request from some other agent to perform information acquisition.

Observations of any agent taken separately are insufficient to draw conclusions of a desired type; however, the combined information of both agents is sufficient. Therefore, the agents need to co-operate to be able to draw conclusions. Each agent can be proactive with respect to the conclusion generation, i.e., after receiving both observation results an agent is capable to generate and communicate a conclusion to agent  $C$ . Moreover, an agent can be request pro-active to ask information from another agent, and an agent can be pro-active or reactive in provision of (already acquired) information to the other agent.

For the lower-level components of this example multi-agent system, a number of dynamic properties were identified and formalized in TTL. For the purposes of illustration of the proposed transformation procedure the dynamic property that describes an information provision reactivity of the agent  $B$  has been chosen. Informally this property expresses that the agent  $B$  generates an information chunk (the constant  $\text{IC}$  of sort  $\text{INFORMATION\_CHUNK}^{\text{TERMS}}$ ) for the agent  $A$  if the agent  $B$  observes the  $\text{IC}$  at its input from the environment and at some point in the past  $B$  received a request for the  $\text{IC}$  from the agent  $A$ . According to the definition of an external behavioral specification the considered property can be represented in the form  $[\phi_p(\gamma, t) \Rightarrow \phi_f(\gamma, t)]$ , where  $\phi_p(\gamma, t)$  is a formula

$$\exists t_2 \leq t \text{ state}(\gamma, t_2, \text{input}(B)) \models \text{communicated}(\text{request\_from\_to\_for}(A, B, \text{IC}))$$

and  $\phi_f(\gamma, t)$  is a formula

$\forall t_1 > t$  [ state( $\gamma$ ,  $t_1$ , input(B)) = observed(provided\_result\_from\_to(E, B, IC))  $\Rightarrow$  state( $\gamma$ ,  $t_1+c$ , output(B)) = output(communicated(send\_from\_to(B, A, IC))) ]

with  $\phi_{cond}(\gamma, t, t_1)$  is

state( $\gamma$ ,  $t_1$ , input(B)) = observed(provided\_result\_from\_to(E, B, IC))

and  $\phi_{bh}(\gamma, t_1)$  is

state( $\gamma$ ,  $t_1+c$ , output(B)) = output(communicated(send\_from\_to(B, A, IC))) ,

where  $t$  is the present time point with respect to which the formulae are evaluated and  $c$  is some natural number.

### Step 1. From interaction states to memory states

The formula  $\phi_{mem}(\gamma, t)$  obtained by replacing all occurrences in  $\phi_p(\gamma, t)$  of subformulae of the form state( $\gamma, t$ ) =  $p$  by state( $\gamma, t$ ) = memory( $t$ ,  $p$ ) is called the *memory formula* for  $\phi_p(\gamma, t)$ . Thus, a memory formula defines a sequence of past events (i.e., a history; e.g., observations of an external world, actions) for the present time point  $t$ . According to Lemma 1 (given in [11])  $\phi_{mem}(\gamma, t)$  is equivalent to some formula  $\delta^*(\gamma, t)$  of the form state( $\gamma, t$ ) =  $q_{mem}(t)$ , where  $q_{mem}(t)$  is called the *normalized memory state formula* for  $\phi_{mem}(\gamma, t)$ , which uniquely describes the present state at the time point  $t$  by a certain history of events. Moreover,  $q_{mem}$  is the state formula  $\forall u$  [present\_time( $u$ )  $\rightarrow$   $q_{mem}(u)$ ]. For the considered example  $q_{mem}(t)$  for  $\phi_{mem}(\gamma, t)$  is specified as:

$\exists u_2 \leq t$  memory( $u_2$ , communicated(request\_from\_to\_for(A, B, IC)))

Additionally, memory state persistency properties are composed for all memory atoms. Rules that describe creation and persistence of memory atoms are given in the executable theory from observation states to memory states  $Th_{o \rightarrow m}$  (a general description of this and the following theories is given in [11]). For the example:

$\forall t$  state( $\gamma$ ,  $t$ , input(B)) = communicated(request\_from\_to\_for(A, B, IC))  $\Rightarrow$  state( $\gamma$ ,  $t$ , internal(B)) = memory( $t$ , communicated(request\_from\_to\_for(A, B, IC)))

$\forall t$  state( $\gamma$ ,  $t$ , internal(B)) = memory( $t$ , communicated(request\_from\_to\_for(A, B, IC)))  $\Rightarrow$  state( $\gamma$ ,  $t+1$ , internal(B)) = memory( $t$ , communicated(request\_from\_to\_for(A, B, IC)))

### Step 2. From memory states to preparation states

Obtain  $\phi_{cmem}(\gamma, t, t_1)$  by replacing all occurrences in  $\phi_{cond}(\gamma, t, t_1)$  of state( $\gamma, t$ ) =  $p$  by state( $\gamma, t_1$ ) = memory( $t$ ,  $p$ ). The condition memory formula  $\phi_{cmem}(\gamma, t, t_1)$  contains a history of events, between the time point  $t$ , when  $\phi_p(\gamma, t)$  is true and the time point  $t_1$ , when the formula  $\phi_{cond}(\gamma, t, t_1)$  becomes true. Again by Lemma 1  $\phi_{cmem}(\gamma, t, t_1)$  is equivalent to the formula state( $\gamma, t_1$ ) =  $q_{cond}(t, t_1)$ , where  $q_{cond}(t, t_1)$  is called the *normalized condition state formula* for  $\phi_{cmem}(\gamma, t, t_1)$ , and  $q_{cond}(t)$  is the state formula  $\forall u$  [ present\_time( $u$ )  $\rightarrow$   $q_{cond}(t, u)$  ].

For the considered example  $q_{cond}(t, t_1)$  for  $\phi_{cmem}(\gamma, t)$  is obtained as: memory( $t_1$ , observed(provided\_result\_from\_to(E, B, IC))) and  $q_{cond}(t)$ :  $\forall u$  [ present\_time( $u$ )  $\rightarrow$  memory( $u$ , observed(provided\_result\_from\_to(E, B, IC))) ].

Obtain  $\phi_{prep}(\gamma, t_1)$  by replacing in  $\phi_{bh}(\gamma, t_1)$  any occurrence of state( $\gamma, t_1+c$ ) = output( $a$ ) by state( $\gamma, t_1$ ) = preparation\_for(output( $t_1+c, a$ )), for some number  $c$  and action or communication  $a$ . The preparation state is created at the same time point  $t_1$ , when the condition for an output  $\phi_{cond}(\gamma, t, t_1)$  is true. By Lemma 1  $\phi_{prep}(\gamma, t_1)$  is equivalent to the state formula state( $\gamma, t_1$ ) =  $q_{prep}(t_1)$ , where  $q_{prep}(t_1)$  is called the *normalized preparation state formula* for  $\phi_{cond}(\gamma, t_1)$ . Moreover,  $q_{prep}$  is the state formula  $\forall u$  [present\_time( $u$ )  $\rightarrow$   $q_{prep}(u)$ ]. For the considered example  $q_{prep}(t_1)$  is composed as preparation\_for(output( $t_1+c$ , communicated(send\_from\_to(B, A, IC))))).

Rules, which describe generation and persistence of condition memory states, a transition from the condition to the preparation state, and the preparation state generation and persistence, are given in the *executable theory from memory states to preparation states*  $Th_{m \rightarrow p}$ . For the considered example:

$\forall t$  [ state( $\gamma, t$ , input(B)) = observed(provided\_result\_from\_to(E, B, IC))  $\Rightarrow$  state( $\gamma, t$ , internal(B)) =

[ memory( $t$ , observed(provided\_result\_from\_to(E, B, IC)))  $\wedge$  stimulus\_reaction(observed(provided\_result\_from\_to(E, B, IC))) ]

$\forall t$  state( $\gamma, t$ , internal(B)) = memory( $t$ , observed(provided\_result\_from\_to(E, B, IC)))  $\Rightarrow$  state( $\gamma, t+1$ , internal(B)) = memory( $t$ , observed(provided\_result\_from\_to(E, B, IC)))

$\forall t$  state( $\gamma, t$ ) =  $\forall u$  [ present\_time( $u$ )  $\rightarrow$   $\exists u_2$  [ memory( $u_2$ , communicated(request\_from\_to\_for(A, B, IC))) ] ]  $\Rightarrow$  state( $\gamma, t$ ) =  $\forall u$  [ present\_time( $u$ )  $\rightarrow$  [  $\forall u_1 > u$  [ memory( $u_1$ , observed(provided\_result\_from\_to(E, B, IC)))  $\rightarrow$  preparation\_for(output( $u_1+c$ , communicated(send\_from\_to(B, A, IC)))) ] ] ]

$\forall t, t$  state( $\gamma, t$ ) =  $\forall u$  [ present\_time( $u$ )  $\rightarrow$  [  $\forall u_1 > u$  [ memory( $u_1$ , observed(provided\_result\_from\_to(E, B, IC)))  $\rightarrow$  preparation\_for(output( $u_1+c$ , communicated(send\_from\_to(B, A, IC)))) ] ] ]  $\wedge$   $\forall u$  [ present\_time( $u$ )  $\rightarrow$  memory( $u$ , observed(provided\_result\_from\_to(E, B, IC))) ]  $\wedge$  stimulus\_reaction(observed(provided\_result\_from\_to(E, B, IC))) ]  $\Rightarrow$  state( $\gamma, t$ , internal(B)) =  $\forall u_1$  [ present\_time( $u_1$ )  $\rightarrow$  preparation\_for(output( $u_1+c$ , communicated(send\_from\_to(B, A, IC)))) ]

$\forall t$  state( $\gamma, t$ ) = [ stimulus\_reaction(observed(provided\_result\_from\_to(E, B, IC)))  $\wedge$  not(preparation\_for(output( $t+c$ , communicated(send\_from\_to(B, A, IC)))) ) ]  $\Rightarrow$  state( $\gamma, t+1$ ) = stimulus\_reaction(observed(provided\_result\_from\_to(E, B, IC)))

$\forall t$  state( $\gamma, t$ , internal(B)) = [ preparation\_for(output( $t+c$ , communicated(send\_from\_to(B, A, IC))))  $\wedge$  not(output(communicated(send\_from\_to(B, A, IC)))) ]  $\Rightarrow$  state( $\gamma, t+1$ , internal(B)) = preparation\_for(output( $t+c$ , communicated(send\_from\_to(B, A, IC)))) .

The auxiliary functions stimulus\_reaction( $a$ ) are used for reactivation of agent preparation states for generating recurring actions or communications.

### Step 3. From preparation states to output states

The preparation state preparation\_for(output( $t_1+c, a$ )) is followed by the output state, created at the time point  $t_1+c$ . Rules that describe a transition from preparation to output state(s) are given in the *executable theory from the preparation to the output state(s)*  $Th_{p \rightarrow o}$ . For the considered example the following rule holds:

$\forall t$  state( $\gamma, t$ , internal(B)) = preparation\_for(output( $t+c$ , communicated(send\_from\_to(B, A, IC))))  $\Rightarrow$  state( $\gamma, t+c$ , output(B)) = output(communicated(send\_from\_to(B, A, IC))) .

### Step 4. Constructing an executable specification

An executable specification  $\pi(\gamma, t)$  for the component  $A$  is defined by a union of the dynamic properties from the executable theories  $Th_{o \rightarrow m}$ ,  $Th_{m \rightarrow p}$  and  $Th_{p \rightarrow o}$ , identified during the steps 1-3. For the purposes of analysis of component dynamics the non-executable external behavioral specification is replaced by the executable behavioral specification. The justification of such substitution is based on the theorem in [11].

At **Step 5** an executable specification is constructed for the whole external behavioral specification of an agent.

### Step 6. Translation of an executable specification into a description of a transition system

For the purpose of practical verification a behavioral specification based on executable temporal logical properties generated at Step 5 is translated into a finite state transition system model (a general procedure is described in [11]).

To translate the executable specification of agent behavior into the transition system representation the atom present\_time is used. This atom is evaluated to *true* only in a state for the current time point. The executable properties from the executable specification, translated into the transition rules for the considered example are given below:

$$\begin{aligned}
& \text{present\_time}(t) \wedge \text{communicated}(\text{request\_from\_to\_for}(A,B,IC)) \rightarrow \\
& \quad \text{memory}(t, \text{communicated}(\text{request\_from\_to\_for}(A,B,IC))) \\
& \text{present\_time}(t) \wedge \text{observed}(\text{provided\_result\_from\_to}(E,B,IC)) \rightarrow \\
& \quad \text{memory}(t, \text{observed}(\text{provided\_result\_from\_to}(E,B,IC))) \wedge \\
& \quad \text{stimulus\_reaction}(\text{observed}(\text{provided\_result\_from\_to}(E,B,IC))) \\
& \text{memory}(t, \text{communicated}(\text{request\_from\_to\_for}(A,B,IC))) \rightarrow \\
& \quad \text{memory}(t, \text{communicated}(\text{request\_from\_to\_for}(A,B,IC))) \\
& \text{memory}(t, \text{observed}(\text{provided\_result\_from\_to}(E,B,IC))) \rightarrow \\
& \quad \text{memory}(t, \text{observed}(\text{provided\_result\_from\_to}(E,B,IC))) \\
& \text{present\_time}(t) \wedge \\
& \quad \exists u2 \leq t \text{ memory}(u2, \text{communicated}(\text{request\_from\_to\_for}(A, B, IC))) \rightarrow \\
& \quad \quad \text{conditional\_preparation\_for}(\text{output}(\text{communicated}(\text{send\_from\_to}(B,A,IC)))) \\
& \text{present\_time}(t) \wedge \\
& \quad \text{conditional\_preparation\_for}(\text{output}(\text{communicated}(\text{send\_from\_to}(B,A,IC)))) \wedge \\
& \quad \text{memory}(t, \text{observed}(\text{provided\_result\_from\_to}(E,B, IC))) \wedge \\
& \quad \text{stimulus\_reaction}(\text{observed}(\text{provided\_result\_from\_to}(E,B,IC))) \rightarrow \\
& \quad \quad \text{preparation\_for}(\text{output}(t+c, \text{communicated}(\text{send\_from\_to}(B,A,IC)))) \\
& \text{present\_time}(t) \wedge \\
& \quad \text{stimulus\_reaction}(\text{observed}(\text{provided\_result\_from\_to}(E,B, IC))) \wedge \\
& \quad \text{not}(\text{preparation\_for}(\text{output}(t+c, \text{communicated}(\text{send\_from\_to}(B,A,IC)))))) \rightarrow \\
& \quad \quad \text{stimulus\_reaction}(\text{observed}(\text{provided\_result\_from\_to}(E,B,IC))) \\
& \text{preparation\_for}(\text{output}(t+c, \text{communicated}(\text{send\_from\_to}(B,A,IC)))) \wedge \\
& \quad \text{not}(\text{output}(\text{communicated}(\text{send\_from\_to}(B,A,IC)))) \rightarrow \\
& \quad \quad \text{preparation\_for}(\text{output}(t+c, \text{communicated}(\text{send\_from\_to}(B,A,IC)))) \\
& \text{preparation\_for}(\text{output}(t+c, \text{communicated}(\text{send\_from\_to}(B,A,IC)))) \wedge \\
& \quad \text{present\_time}(t+c-1) \rightarrow \\
& \quad \quad \text{output}(\text{communicated}(\text{send\_from\_to}(B,A,IC))).
\end{aligned}$$

For automatic verification of relationships between dynamic properties of different aggregation levels by means of the model checking tool SMV, the obtained finite state transition system is translated into the input format of the SMV model checker. For the description of the translation procedure and the complete SMV specification for the considered example we refer to [11].

One of the possible dynamic properties of the higher aggregation level that can be verified against the generated SMV specification is formulated and formalized in CTL as follows:

**GP (Concluding effectiveness):** If at some point in time environmental component E generates all the correct relevant information, then later agent C will receive a correct conclusion.

**AG** (E\_output\_observed\_provide\_result\_from\_to\_E\_A\_info & E\_output\_observed\_provide\_result\_from\_to\_E\_B\_info  
 $\rightarrow$  **AF** input\_C\_communicated\_send\_from\_to\_A\_C\_info),

where **A** is a path quantifier defined in CTL, meaning “for all computational paths”, **G** and **F** are temporal quantifiers that correspond to “globally” and “eventually” respectively.

The automatic verification by the SMV model checker confirmed that this property holds with respect to the considered model of the multi-agent system as specified at the lower level.

## 4 DISCUSSION

The approach to analyzing behavior of a multi-agent system proposed in this paper is based on distinguishing dynamic properties of different aggregation levels and verifying interlevel relations between them using model checking techniques. To enable model checking an automated procedure has been developed, which allows transformation of a behavioral specification of a certain aggregation level first into an executable temporal specification, and subsequently into a description of a finite state transition system. Using this, model checking is performed in the environment SMV. The proposed approach has been evaluated by several case studies, in which verification of agent-based systems from natural domains has been performed.

Notice that an SMV-specification comprises constants, variables and state transition rules with limited expressiveness

(e.g., no quantifiers). Furthermore, for expressing one complex temporal relation a large quantity (including auxiliary) of transition rules is needed. Specification of multi-agent system behavior in the more expressive predicate-logic-based language TTL is much easier. TTL proposes an intuitive way of creating a specification of system dynamics, which still can be automatically translated into a state transition system description, as shown here.

The complexity of the representation of the obtained executable model is linear in size of the non-executable behavioral specification. More specifically, the non-executable specification is related to the SMV specification in the following linear way: (1) for every quantified variable from a non-executable specification a variable and an appropriate rule for its update are introduced; (2) for every nested quantifier an additional variable and an auxiliary executable rule are introduced, which establishes a relation between the quantified variables; (3) for every communicated and observed function from a past and a conditional formulae from dynamic properties, a corresponding memory state creation and a memory state persistence rule are introduced using the variables described in (1) and (2), and variables that correspond to external events; (4) for every non-executable dynamic property auxiliary variables  $f_{mem}$  and  $f_{prep}$  (i.e., the variables that indicate truth values of  $\phi_{mem}(\gamma, t)$  and  $\phi_{prep}(\gamma, t_1)$  respectively) and corresponding update rules are introduced; (5) for every action and communication specified in  $\phi_{bh}(\gamma, t_1)$  a variable and an appropriate update rule are introduced; (6) for reactivation of agent preparation states the auxiliary variables and the update rules corresponding to communicated and observed functions from  $\phi_{prep}(\gamma, t_1)$  are introduced. For verifying an executable model in the SMV OBDD-based symbolic model checking algorithms are used; the study of complexity of such algorithms is given in [9].

## REFERENCES

- [1] E.M. Clarke, O. Grumberg, D.A. Peled, *Model Checking*, MIT Press, Cambridge Massachusetts, London England, 1999.
- [2] M. Fisher, Temporal Development Methods for Agent-Based Systems, *Journal of Autonomous Agents and Multi-Agent Systems*, **10(1)**,41-66, (2005).
- [3] J. Hooman, Compositional Verification of a Distributed Real-Time Arbitration Protocol, *Real-Time Systems*, **6**, 173-206, (1994).
- [4] C.M. Jonker and J. Treur, Compositional Verification of Multi-Agent Systems: a Formal Analysis of Pro-activeness and Reactiveness, *International Journal of Cooperative Information Systems*, **11**, 51-92, (2002).
- [5] R. Kowalski and M.A. Sergot, A logic-based calculus of events, *New Generation Computing*, **4**, 67-95, (1986).
- [6] M. Manzano, *Extensions of First Order Logic*, Cambridge University Press, 1996.
- [7] D.A. Marca, *SADT: Structured Analysis and Design Techniques*, McGraw-Hill, 1988.
- [8] L. Marcio Cysneiros and E. Yu, Requirements Engineering for Large-Scale Multi-agent Systems. In: *Proceedings of the 1st International Central and Eastern European Conference on Multi-Agent Systems (CEEMAS)*, 39-56, (2002).
- [9] K. McMillan, *Symbolic Model Checking*, Kluwer Academic Publishers, 1993.
- [10] R. Reiter, *Knowledge in Action: Logical Foundations for Specifying and Implementing Dynamical Systems*, Cambridge MA: MIT Press, 2001.
- [11] A. Sharpanskykh and J. Treur, *Verifying Interlevel Relations within Multi-Agent Systems: formal theoretical basis*, Technical Report No. TR-1701AI, Artificial Intelligence Department, Vrije Universiteit Amsterdam, (2005). <http://hdl.handle.net/1871/9777>