Automated Analysis of Compositional Multi-Agent Systems

Alexei Sharpanskykh, Jan Treur

VU University Amsterdam, Department of Artificial Intelligence
De Boelelaan 1081, Amsterdam, 1081HV The Netherlands
http://www.few.vu.nl/~{sharp, treur}

Abstract. An approach for handling the complex dynamics of a multi-agent system is based on distinguishing aggregation levels. The behaviour at a given aggregation level is specified by a set of dynamic properties at that level, expressed in some (temporal) language. Such behavioural specifications may be complex and difficult to analyse. To enable automated analysis of system specifications, a simpler format is required. To this end, a specification at a lower aggregation level can be created, describing basic steps in the processes of a system. This paper presents a method and tool to support the automated creation of such a specification, as a refinement of a given higher level specification. The generated specification has a simple format which can easily be used for analysis. This paper describes an approach for automated verification of logical consequences of specifications using model checking techniques.

Keywords: automated analysis of multi-agent systems, compositional modeling of multi-agent systems, compositional verification of multi-agent systems, model checking, automated transformation of behavioral specifications, temporal modeling, temporal analysis, executable format, transition system

1 Introduction

Often the dynamics of a multi-agent system is described by a behavioural temporal specification, which consists of dynamic properties of elements of the system (i.e., agents, interaction relations, and an environment). Usually, these properties are expressed as formulae in some (temporal) language. Even if the behavioural description of a single element is simple, the overall dynamics of the multi-agent system is often difficult to analyze (e.g., to establish the satisfaction of some crucial general properties of the system given the description of the local dynamics of its elements). With the 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 behaviour 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 for managing complex dynamics is by representing a multi-agent system as a composite component-based system, in which different *aggregation levels* can be distinguished. Then, such systems can be analyzed using generic analysis techniques for component-based systems (e.g., software programs); e.g., (Jonker and Treur, 2002). In the component-based representation of a multi-agent system at the lowest aggregation level a component is an agent or an environmental object (e.g., a database). A component that represents an agent is able to interact with other components and with the environment, and may have internal states that correspond to internal states of the agent. Further, at higher aggregation levels a component has the form of either a group of agents or a multi-agent system as a whole. A grouping of components in higher level components may be based on diverse principles: e.g., similarity of tasks performed by components, intensity of communication between components, behavioural similarity of components. 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 behaviour of a component is described by a set of dynamic properties. To specify dynamics properties of components, the reified predicate logic temporal language (Galton, 2006) is used in this paper. Using the state language of the reified temporal predicate logic one can specify diverse aspects and features of a multi-agent system (e.g., cognitive states, deontic aspects, norms, interaction possibilities). Furthermore, also temporal relations with a numerical representation of time on internal and externally observable states of agents are possible to express using the reified temporal predicate logic. In contrast to different variants of modal logic dedicated to multi-agent systems (Bordini et al., 2004; Fisher, 2005), the reified temporal predicate logic does not contain syntactical elements dedicated to multi-agent systems in particular. Nevertheless, the reified temporal predicate logic provides to the designer the possibility to introduce necessary aspects and constructs as a part of the state ontology for all or particular agents, as has been demonstrated in many applications (cf. Bosse et al, 2009).

The dynamic properties of components of a higher aggregation level may have the form of a few temporal expressions of high complexity. At a lower aggregation level a system is described in terms of more basic steps. This usually takes the form of a specification consisting of a large number of temporal expressions in a simpler format. Furthermore, 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** (Marcio Cysneiros and Yu, 2002) and SADT (Marca, 1988). To formally prove that the identified interlevel relations are indeed correct, model checking techniques (Clarke, Grumberg and Peled, 1999; McMillan, 1993) 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 behavioural 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 necessary to transform an original behavioural specification of the lower aggregation level into a model based on a finite state transition system. To obtain this, as a first step, a behavioural description for the lower aggregation level is replaced by one in executable temporal format. A software environment has been developed to automate this process. 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. In general, a representation in the form of a finite state transition system is required by many existing verification and simulation tools (e.g., model checkers). However, specific input formats of such tools differ. To address this issue, an intermediate general finite state transition system format is introduced, which can be translated easily into the input formats of a particular tool. In this paper it is shown how a specification in this general format is translated into the input format of the SMV model checker. Using model checking techniques (SMV in particular), 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.

Moreover, besides model checking, executable specifications can be analyzed using other available logical techniques and supporting software environments such as causal-temporal modelling approaches (e.g., LEADSTO; Bosse et al., 2007), modal temporal logic (e.g., MetateM; Fisher, 1996), monodic first-order logic (Hodkinson, Wolter and Zakharyaschev, 2000), and the guarded fragment of predicate logic (Andreka, Benthem and Nemeti, 1998).

The paper is organized as follows. In Section 2 the concepts for formal specification of the dynamics of multi-agent system components and the reified predicate logic temporal language used are briefly introduced. After that, in Section 3 an overview of the transformation procedure from a behavioural specification into a specification in an executable format is described and illustrated by means of an example. Sections 4-7 describe in more detail the different steps of the procedure. Transformation of an executable specification into a finite state transition system description is described in Sections 8 and 9. Section 10 describes a case study on verification of interlevel relations in the multi-agent system for co-operative information gathering. Some complexity considerations of the proposed transformation procedure are presented in Section 11. In Section 12 the implementation details of the described transformation procedure are discussed. Section 13 shows how an executable specification can easily be used to perform analysis using one of four different available logical techniques and supporting software environments: LEADSTO, Propositional Modal Temporal Logic and MetateM, Monodic First-Order Temporal Logic, and the Guarded Fragment of Predicate Logic. Section 14 discusses the related literature. The paper ends with a discussion in Section 15.

2 Temporal Specification of Dynamic Properties

In this section first the temporal modelling approach adopted is discussed in Section 2.1. This approach is based on the reified temporal predicate logic. After that modelling dynamics of components, in which a multi-agent system may be clustered, using the adopted approach is considered in Section 2.2.

2.1 The Temporal Modelling Approach Adopted

From the philosophical perspective Galton (2003) considers two main streams in temporal logic: modal logic approaches to temporal logic (developed mainly within Computer Science), and predicate logic approaches to temporal logic (developed mainly within AI). In (Galton, 2006) he addresses different approaches in the latter stream in more detail. Two substreams distinguished are the use of temporal arguments within domain predicates, and the reification approach, where state properties are represented not by statements but by terms in the language, and predicates are used to express temporal structure over these term expressions. In this approach part of the model theory is incorporated in the language. This reification approach to predicate logical temporal modelling is the approach adopted here. A basic predicate used in this approach is the holds_at predicate:

holds_at(p, t)

means that state property p holds at time point t. The model theory notation for this is

γ, t |= p

where γ is a model representing a possible trace of the process (i.e., a sequence of states indexed by the time frame). The notion of a trace in the reification approach has a meaning similar to the notion of a path in LTL (Clark et al., 1999). However, in contrast to LTL, numerical time values may be associated with each state in the reification approach.

One of the features of the language used is that the trace γ indicated above also can be represented (by a constant or a variable) as a first class citizen in the language. So, as a variant of

holds_at(p, t)

the expression

holds_at(p, γ, t)

means that state property p holds in the state of trace γ at time point t, also denoted in an infix notation by

state(γ, t) |= p

This feature gives the possibility to quantify over traces and to compare traces, which can be useful and even necessary when adaptive behaviour is analysed. Quantification over traces has a meaning similar to path quantification in CTL (Clark et al., 1999), which is not possible in LTL. For example, a property such as 'the more exercising, the more skill' compares two traces, one with less and one with more exercising. Another example of such a property is trust monotonicity: 'the better the experiences, the higher the trust'. However, in the current paper these trace-related features of the language are left out of consideration. The subset of the language considered here does not include quantification over traces; when the argument γ occurs in a formula, it will be considered a fixed constant; thus an expression such as holds_at(p, γ, t) or state(γ, t) |= p is equivalent to (and can be replaced by) holds_at(p, t), which is the more standard expression in reified predicate logic approaches to temporal modelling. In the following we shall use t with subscripts and superscripts for variables of the sort TIME; and γ with subscripts and superscripts for variables of the sort TRACE.

2.2 Modelling Dynamics of Components Using the Reified Temporal Predicate Logic

Components can be active (e.g., an agent) or passive (e.g., database). An active component represents an autonomous entity that interacts with the environment and with other components. The 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. State properties of a component are expressed as terms using a standard many-sorted first-order predicate language with a signature, which consists of a number of sorts, sorted constants, variables, functions and predicates. Specifically, to express state properties every component A has assigned an interaction state ontology InteractionOnt(A) for its input and output states. This ontology contains such sorts as STATE (a set of all state names of a system), TRACE (a set of all trace names; a trace or trajectory can be thought of as a timeline with a state for each time point), STATPROP (a set of all state property names), and VALUE (an ordered set of numbers).

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. Within an agent system context, using an ontology InteractionOnt one can define observations of state properties by a component (by function observed: STATPROP $\rightarrow$ STATPROP), communications (by function communicated: STATPROP $\rightarrow$ STATPROP), and actions (by function output: STATPROP $\rightarrow$ STATPROP). To indicate that a component has a state, the function has_state: COMPONENT x STATPROP $\rightarrow$ STATPROP is used. For example, the observation of a component A of the movement of another component B from the position p1 to the position p2 can be specified by has_state(A, observed(moved_to_from(B, p1, p2))).

As in the approach described in Section 2.1 the statement that a state property p holds at a time point t is formalized using the reified temporal predicate logic: holds_at(p, t).

In the formulae of the reified temporal predicate logic the formulae of the state language are used as objects. For every sort S from the state language the following sorts of the reified temporal predicate language exist: the sort S^{VARS}, which contains all variable names of sort S; the sort S^{GTERMS}, which contains names of all ground terms, constructed using sort S; sorts S^{GTERMS} and S^{VARS} are subsorts of sort S^{TERMS}. To provide names of state language formulae φ in the reified logic the operator (*) is used (written as φ*), which maps variable sets, term sets and formula sets of the state language to the elements of sorts of the reified logic S^{GTERMS}, S^{TERMS}, S^{VARS} and STATPROP. The set of function symbols of reified temporal predicate logic includes $\wedge$, $\vee$, $\rightarrow$, $\leftrightarrow$: STATPROP x STATPROP$\rightarrow$ STATPROP; not: STATPROP$\rightarrow$ STATPROP, $\forall$, $\exists$: S^{VARS} x STATPROP$\rightarrow$ STATPROP, which are counterparts of Boolean connectives and quantifiers in the state language. Further we shall use $\wedge$, $\vee$, $\rightarrow$, $\leftrightarrow$ in infix notation and $\forall$, $\exists$ in prefix notation for better readability. The terms of the reified temporal predicate logic are constructed by induction in a standard way from variables, constants and function symbols typed with all before mentioned sorts.

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 LTIME) to time external to states (sort TIME) a function present_time: $\text{LTIME}^{\text{TERMS}} \rightarrow$ STATPROP is used. Here time is assumed to have the properties of correctness and uniqueness:

Uniqueness of time
This expresses that present_time(t) is true for at most one time point t:
$\forall t, t''$ holds_at(present_time(t''), t) $\Rightarrow \forall t', t' \neq t''$ $\neg$holds_at(present_time(t'), t)
Correctness of time
This expresses that present_time(t) is true for the current time point t:
$\forall t$ holds_at(present_time(t), t)

Furthermore, for the purposes of this paper it is assumed that $\text{LTIME}^{\text{GTERMS}}$=TIME and $\text{LVALUE}^{\text{GTERMS}}$=VALUE (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 VALUE and TIME function symbols $-$, $+$, $/$, $\bullet$: TIME x VALUE$\rightarrow$ TIME are introduced. And for sorts $\text{LVALUE}^{\text{TERMS}}$ and $\text{LTIME}^{\text{TERMS}}$ the function symbols $-$, $+$, $/$, $\bullet$ are overloaded: $\text{LTIME}^{\text{TERMS}}$ x $\text{LVALUE}^{\text{TERMS}} \rightarrow$ STATPROP.

Temporal relations between state properties at different points in time are described by dynamic properties, which are expressed by formulae. The set of *atomic formulae* of the reified temporal predicate logic is defined as:

(1) If v_1 is a term of sort STATE, and u_1 is a term of the sort STATPROP, then holds_at(v_1,u_1) is an atomic formula.
(2) If τ_1, τ_2 are terms of any sort of the reified temporal predicate logic, then $\tau_1=\tau_2$ is an atomic formula.
(3) If t_1, t_2 are terms of sort TIME, then $t_1<t_2$ is an atomic formula.

The set of *well-formed formulae* of the reified temporal predicate logic is defined inductively in a standard way using Boolean connectives and quantifiers.

Dynamic properties to model a behavioural 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, which is a necessary assumption for enabling verification of a system using existing model checking techniques and tools. Past, interval and future statements that can be used to formalize input and output temporal patterns are defined as follows:
a) A *past statement* for a trace γ and a time point t over state ontology Ont is a temporal statement $\varphi_p(\gamma,t)$ in the reified temporal predicate logic, 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 γ and a time point t over state ontology Ont is a temporal statement $\varphi_f(\gamma,t)$ in the reified temporal predicate logic, 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 γ and time points t_1 and t_2 over state ontology Ont is a temporal statement $\varphi(\gamma,t_1,t_2)$ in the reified temporal predicate logic that is a past statement for t_2 and a future statement for t_1.

For example, the following dynamic property describes a communication between components A and B: for all time points if component A receives a request for information from component B at its input, then at a later time point it will produce an answer for component B at its output. Formally:

$\forall$t1 [holds_at(has_state(A, communicated(request_from_to_for(B, A, info))), t1)
    $\Rightarrow \exists$t2 t2>t1 & holds_at(has_state(A, obs_focus_from_to_for(A, B, answer))), t2)]

Furthermore, one can also specify temporal statements that describe some temporal patterns of environment processes. For example, the dynamic property expressing that the temperature of location l1 in the environment is rising between time points t1 and t2 can be formalized as

$\forall t, t'$ $\forall c_1,c_2$ [$t_1 \leq t \leq t' \leq t_2$ & holds_at(has_state(l1, temperature(c_1)), t) & holds_at(has_state(l1, temperature(c_2)), t') $\Rightarrow c_1 \leq c_2$]

Environmental states may be (partially) observable by components. Observation of the environmental states by a component can be specified as a three-step process:

(1) A component sends an observation focus for some information to the environment and the environment receives it.

(2) The environment generates information for the observation focus (i.e., the observation result) for the component-requester (e.g., the result that an information chunk from the observation focus is valid).

(3) The provided observation result is received by the component-requester.

These steps are formalized by dynamic properties in an example of a multi-agent system for co-operative information gathering considered in the following Section 3.

3 Overview of the Transformation Process

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

Definition 3.1 (External Behavioural Specification)
An *external behavioural specification* for a multi-agent system component consists of dynamic properties $\varphi(\gamma,t)$ expressed in the reified temporal predicate logic of the form

$$\varphi_p(\gamma,t) \Rightarrow \varphi_f(\gamma, t)$$

where $\varphi_p(\gamma, t)$ is a past statement over the interaction ontology and $\varphi_f(\gamma, t)$ is a future statement. The future statement is represented in the form of a conditional behaviour:

$$\varphi_f(\gamma, t) \Leftrightarrow \forall t_1 > t\ [\varphi_{cond}(\gamma, t, t_1) \Rightarrow \varphi_{bh}(\gamma, t_1)],$$

where $\varphi_{cond}(\gamma, t, t_1)$ is an interval statement over the interaction ontology, which describes a condition for some specified action(s) and/or communication(s), and $\varphi_{bh}(\gamma, t_1)$ is a (conjunction of) future statement(s) for t_1 over the output ontology of the form holds_at(a, γ, t_1+c), for some integer constant c and action or communication a.

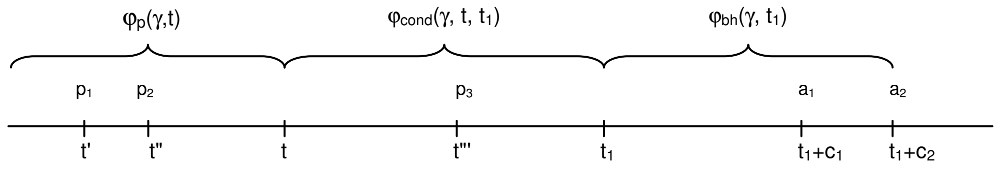

Figure 1. Graphical illustration of the structure of a formula from an external behavioural specification. In the illustration p_1, p_2 and p_3 represent state properties that hold at the time points t', t'' and t''' correspondingly, and a1, a2 are the actions executed at time points t_1+c_1 and t_1+c_2 correspondingly.

When a past formula $\varphi_p(\gamma,t)$ that describes in Figure 1 a temporal pattern over state properties p_1 and p_2 at time points t' and t'', is true for γ at time t, a potential to perform one or more action(s) (a_1 and a_2 in Figure 1) and/or communication(s) exists. This potential is realized at time t_1 when the condition formula $\varphi_{cond}(\gamma,t,t_1)$ specified in Figure 1 over a state property p_3 that holds at t''', becomes true, which leads to the action(s) and/or communication(s) being performed at the time point(s) t_1+c indicated in $\varphi_{bh}(\gamma,t_1)$ (t_1+c and t_1+c_2 in Figure 1).

The term 'external' refers to the fact that such a specification is merely based on the interaction state ontology, no other (e.g., no internal or hidden) state ontology is assumed. An external behavioural specification can include arbitrarily complex temporal relationships. In contrast, an executable specification consists of a set of dynamic properties in a more simple executable temporal language, representing transition-like temporal relations between pairs of states. A dynamic property in executable format relates a state property that holds at some time point to the same or different state property that holds at the same or different time point. Specifications in such a format can be used directly for simulation and other types of automated analysis.

Definition 3.2 (Executable Format)

A temporal formula is in *executable format* if it has one of the following forms, for all properties X and Y with $X \neq Y$, and integer constant c.

(1) $\forall t$ holds_at(X, γ, t) $\Rightarrow$ holds_at(Y, γ, t+c) (states relation property)

(2) $\forall t$ holds_at(X, γ, t) $\Rightarrow$ holds_at(X, γ, t+1) (persistency property)

(3) $\forall t$ holds_at(X, γ, t) $\Rightarrow$ holds_at(Y, γ, t) (state relation property)

The next step is to define when a specification in executable format is a refinement of a given external behavioural specification. First the following definition is needed.

Definition 3.3 (Coinciding Traces)

Two traces γ_1, γ_2 *coincide* on ontology Ont denoted by a predicate symbol

coincide_on: TRACE x TRACE x ONTOLOGY

where ONTOLOGY is a sort that contains all names of ontologies, if and only if

$\forall t\ \forall a \in \text{STATATOM}_{\text{Ont}}$ [holds_at(a, γ_1, t) $\Leftrightarrow$ holds_at(a, γ_2, t)]

Here $\text{STATATOM}_{\text{Ont}} \subseteq \text{STATPROP}_{\text{Ont}}$ is the sort which contains all names of ground atoms expressed in terms of Ont.

The notion of refinement as expressed in the following Definition plays a central role in this paper.

Definition 3.4 (Refinement of an External Dynamic Property)

Let $\varphi(\gamma, t)$ be an externally observable dynamic property. An executable specification $\pi(\gamma, t)$ *refines* $\varphi(\gamma, t)$ iff

(1) for any trace γ: $\forall t\ \ \pi(\gamma, t) \Rightarrow \varphi(\gamma, t)$

(2) for any trace γ_1 exists trace $\gamma_2 \forall t$ [$\varphi(\gamma_1, t) \Rightarrow$ coincide_on(γ_1, γ_2, InteractionOnt(A)) & $\pi(\gamma_2, t)$]

Note that this definition achieves that if $\pi(\gamma, t)$ refines $\varphi(\gamma, t)$ and γ is a trace generated in accordance with $\pi(\gamma, t)$ then by (1) it follows that this trace satisfies $\varphi(\gamma, t)$. This means that simulation traces generated on the basis of specification $\pi(\gamma, t)$ are simulation traces for $\varphi(\gamma, t)$. Moreover, (2) guarantees that every trace for $\varphi(\gamma, t)$ can be obtained in this manner. This shows that analysis by simulation of $\varphi(\gamma, t)$ can be done based on $\pi(\gamma, t)$. For another type of analysis, namely verification of logical consequences, first a theorem is needed. This theorem needs the following Lemma. [1]

Lemma 3.1 (Coinciding Traces)

Let $\varphi(\gamma, t)$ be a dynamic property expressed using the state ontology Ont. Then the following hold:

(1) coincide_on(γ_1, γ_2, Ont) & coincide_on(γ_2, γ_3, Ont) $\Rightarrow$ coincide_on(γ_1, γ_3, Ont)

(2) coincide_on(γ_1, γ_2, Ont) $\Rightarrow$ [$\varphi(\gamma_1, t) \Leftrightarrow \varphi(\gamma_2, t)$].

[1] Proofs of lemmas, propositions and theorems given in this paper are provided in Appendix A

Corollary 3.1 (Equivalence of formulae)

For any past interaction statement $\varphi_p(\gamma, t)$ and future interaction statement $\varphi_f(\gamma, t)$ and traces $\gamma 1$ and $\gamma 2$ the following holds:

$$\text{coincide\_on}(\gamma 1, \gamma 2, \text{InteractionOnt}) \Rightarrow [\ \varphi_p(\gamma 1, t) \Leftrightarrow \varphi_p(\gamma 2, t)\ \&\ \varphi_f(\gamma 1, t) \Leftrightarrow \varphi_f(\gamma 2, t)\ ]$$

Theorem 3.1 (Refinement Implies the Same Consequences)

If the executable specification $\pi(\gamma, t)$ refines the external behavioural specification $\varphi(\gamma, t)$ of a multi-agent system component, and $\psi(\gamma, t)$ is a dynamic interaction property of the multi-agent system component in its environment, expressed using the interaction ontology, then for any trace γ

$$[\ \pi(\gamma, t) \ \Rightarrow\ \psi(\gamma, t)\ ] \Leftrightarrow [\ \varphi(\gamma, t) \ \Rightarrow\ \psi(\gamma, t)\ ]$$

Theorem 3.1 shows that when $\pi(\gamma, t)$ refines $\varphi(\gamma, t)$, verification of logical consequences of $\varphi(\gamma, t)$ can be done by verification of logical consequences of $\pi(\gamma, t)$. Therefore, summarizing, when a refinement of $\varphi(\gamma, t)$ has been obtained, analysis is supported of $\varphi(\gamma, t)$ both by simulation and by verification of logical consequences.

In Section 7 it will be proven that every external behavioural specification can be refined into an executable specification. To obtain this refinement, an automated transformation procedure can be used as described briefly below and in more detail in Sections 4 to 7.

For transformation of an external behavioural specification into executable format, postulated internal states of the system are used. Internal states of a component or system A are described using a postulated internal state ontology InternalOnt(A). In Cognitive Science, which has 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; e.g., (Dennett, 1991; Damasio, 2000). 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,

memory(t, observed(a))

expresses that the component has memory that it observed a state property a at time point t. By identifying specific time points in the memory states an ordering of information about past events is preserved in the memory of a component. Before performing an action or communication it is postulated that a component creates an internal preparation state. For example, preparation_for_output(b) represents a preparation of a component to perform an action or a communication b. Each dynamic property in the internal behavioural specification is specified in *executable* form.

To transform an external behavioural specification of a multi-agent system component into the executable format, a procedure sketched below is used.

The Transformation Procedure: Brief Outline

Let $\varphi(\gamma,t)$ be a non-executable dynamic property from an external behavioural specification for the multi-agent system component, for which an executable representation should be found.

(1) Identify the set $Th_{o\rightarrow m}$ of executable temporal properties, which describe transitions from interaction states to memory states (Section 4) (for a graphical representation of relations between the states considered in this procedure see Figure 2).

(2) Identify the set $Th_{m\rightarrow p}$ of executable temporal properties, which describe transitions from memory states to preparation states for output (Section 5).

(3) Identify the set $Th_{p \to o}$ of executable properties, which describe the transition from preparation states to the corresponding output states (Section 6).
(4) From the sets of executable properties, identified during steps 1-3, construct the specification $\pi(\gamma,t) = Th_{o \to m} \cup Th_{m \to p} \cup Th_{p \to o}$ (considered as conjunction), which describes a refinement of $\varphi(\gamma,t)$ (Section 7).

Using the procedure, a non-executable dynamic property is transformed in a number of executable properties. These properties can be seen as an execution chain, which describes the dynamics of the non-executable property. In this chain each unit generates intermediate states, used to link the following unit. In particular, by executing the step 1 a number of properties are created to generate and maintain memory states. These memory states are used to store information about the past dynamics of components, which is available afterwards at any point in time. At the steps 2 and 3 executable properties are created to generate preparation for output and output states of components. In these properties temporal patterns based on memory states are identified required for generation of particular outputs of components. At the step 4 all created properties are combined in one executable specification.

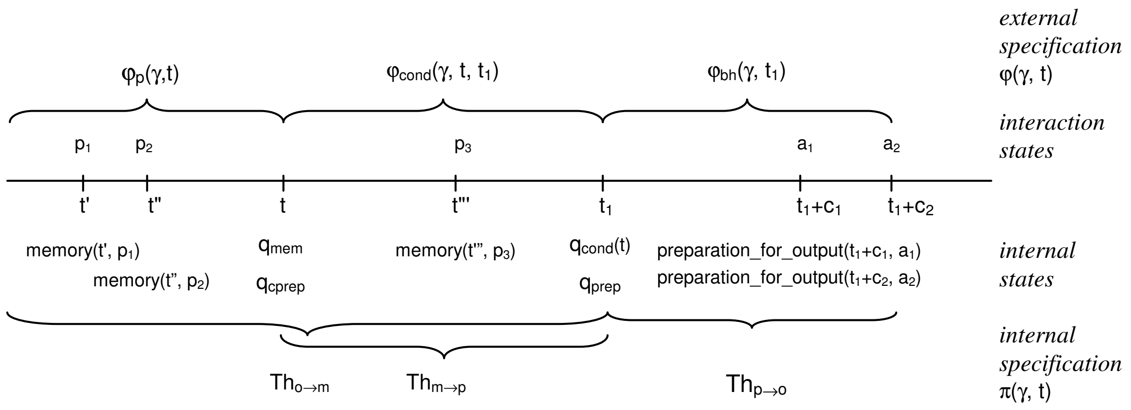

Figure 2. A graphical representation of relations between interaction states described by a non-executable dynamic property and internal states described by rules from the executable theories $Th_{o \to m}$, $Th_{m \to p}$ and $Th_{p \to o}$. Here p_1, p_2 and p_3 represent state properties that hold at time points t', t'' and t'''; the memory states specified over these state properties are memory(t', p_1), memory(t', p_2) and memory(t', p_3); a_1 and a_2 are output states; preparation states for these output states are preparation_for_output(t1+c1, a1) and preparation_for_output(t1+c2, a2); q_{mem} and q_{cprep} are normalized memory and conditional preparation state formulae and $q_{cond}(t)$ and q_{cpret} are normalized condition state formula and preparation state formula used to specify transitions from memory states to preparation states for output.

The details of the proposed procedure are described in the next four sections 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, meaning that the agent may initiate a request for 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 the multi-agent system, a number of dynamic properties were identified and formalized as it is shown below. In the formalization the variables A1 and A2 are defined over the sort $\text{AGENT}^{\text{TERMS}}$, the constant E belongs to the sort $\text{ENVIRONMENTAL\_COMPONENT}^{\text{GTERMS}}$, the variable IC is defined over the sort $\text{INFORMATION\_CHUNK}^{\text{TERMS}}$, the constants IC1, IC2 and IC3 belong to the sort $\text{INFORMATION\_CHUNK}^{\text{GTERMS}}$ and the constant C belongs to the sort $\text{AGENT}^{\text{TERMS}}$.

DP1(A1, A2) (Effectiveness of information request transfer between agents)
If agent A1 communicates a request for an information chunk to agent A2 at any time point t1, then this request will be received by agent A2 at time point t1+c. Formally:

∀IC∀t1 [holds_at(has_state(A1, communicated(request_from_to_for(A1, A2, IC))), t1)
⇒ holds_at(has_state(A2, communicated(request_from_to_for(A1, A2 , IC))), t1+c)

DP2(A1, A2) (Effectiveness of information transfer between agents)
If agent A1 communicates information chunk to agent A2 at any time point t1, then this information will be received by agent A2 at the time point t1+c. Formally:

∀IC∀t1 holds_at(has_state(A1, communicated(message_from_to(A1, A2, IC))), t1)
⇒ holds_at(has_state(A2, communicated(message_from_to(A1, A2, IC))), t1+c)]

DP3(A1, E) (Effectiveness of information transfer between an agent and environment)
If agent A1 communicates an observation request to the environment at any time point t1, then this request will be received by the environment at the time point t1+c. Formally:

∀IC∀t1 [holds_at(has_state(A1, obs_focus_from_to_for(A1, E, IC))), t1)
⇒ holds_at(has_state(E, observed(obs_focus_from_to_for(A1, E, IC))), t1+c)]

DP4(A1, E) (Information provision effectiveness)
If the environment receives an observation request from agent A1 at any time point t1 for a particular chunk of information and this chunk is valid in the environment, then the environment will generate the result for this request that comprise this chunk of information (meaning that the chunk is valid) at the time point t1+c. Formally:

∀IC∀t1 [holds_at(has_state(E, observed(obs_focus_from_to_for(A1, E, IC))), t1) & holds_at(has_state(E, valid_information(IC)), t1)
⇒ holds_at(has_state(E, observed(provide_result_from_to(E, A1, IC))), t1+c)]

DP5(E, A1) (Effectiveness of information transfer between environment and an agent)
If the environment generates a result for an agent's information request at any time point t1, then this result will be received by the agent at the time point t1+c. Formally:

∀IC∀t1 [holds_at(has_state(E, observed(provide_result_from_to(E, A1, IC))), t1)
⇒ holds_at(has_state(A1, observed(provided_result_from_to(E, A1, IC))), t1+c)]

DP6(A1, A2) (Information acquisition reactiveness)
If agent A2 receives a request for an information chunk from agent A1 at any time point t1, then agent A2 will generate a request for this information to the environment at the time point t1+c. Formally:

∀IC∀t1 [holds_at(has_state(A2, communicated(request_from_to_for(A1, A2, IC))), t1)
⇒ holds_at(has_state(A2, obs_focus_from_to_for(A2, E, IC)), t1+c)]

DP7(A1, A2) (Information provision reactiveness)
If exists a time point t2 when agent A2 received a request for a chunk of information from agent A1, then for all time points t1 when the requested information is provided to agent A2, this information will be further provided by agent A2 to agent A1 at the time point t1+c. Formally:

∀IC [∃t2 [t2 ≤t & holds_at(has_state(A2, communicated(request_from_to_for(A1, A2, IC))), t2)]]
⇒ ∀t1 [t<t1 & holds_at(has_state(A2, observed(provided_result_from_to(E, A2, IC))), t1) ⇒
holds_at(has_state(A2, communicated(message_from_to(A2, A1, IC))), t1+c)]]

DP8(A1, A2) (Conclusion proactiveness)
For any time points t1 and t2, if agent A1 receives a result for its observation request from the environment at t1 and it receives information required for the conclusion generation from agent A2 at t2, then agent A1 will generate a conclusion based on the received information to agent C at a time point t4 later than t1 and t2. Formally:

$\forall$IC1, IC2 [$\forall$t1, t2 t1≤t & t2≤t & holds_at(has_state(A1, observed(provided_result_from_to(E, A1, IC1))), t1) &
holds_at(has_state(A1, communicated(message_from_to(A2, A1, IC2))), t2)
$\Rightarrow$ $\exists$IC3, t4>t [holds_at(has_state(A1, communicated(message_from_to(A1, C, IC3))), t4)]]

DP9(A1, E) (Information acquisition proactiveness)
At some time point an observation request is generated by agent A1 to the environment. Formally:

holds_at(has_state(A1, obs_focus_from_to_for(A1, E, IC1)), c)

DP10(A1, A2) (Information request proactiveness)
At some time point a request for an information chunk is communicated by agent A1 to agent A2. Formally:

holds_at(has_state(A1, communicated(request_from_to_for(A1, A2, IC2))), c)

DP11 (Information chunks valid in the environment)
For all time points the chunks of information IC1 and IC2 are valid in the environment.

$\forall$t holds_at(has_state(E, valid_information(IC1)) $\wedge$ valid_information(IC2)), t)

Notice that most of the properties in the behavioural specification above (e.g., DP1, DP2) are already specified in executable format. Therefore, as an illustration the transformation procedure is applied to properties such as DP7 and DP8 which are non-executable (i.e., are not specified in the executable format and establish the relations between multiple states over time). To illustrate the required transformation the dynamic property DP7(A1, A2) with the instantiation of the variables A1 $\leftarrow$ A and A2 $\leftarrow$ B has been chosen. Informally this property expresses that the agent B generates an information chunk (the constant IC of sort $\text{INFORMATION\_CHUNK}^{\text{GTERMS}}$) for the agent A if the agent B observes the IC at its input from the environment and at some point in the past B received a request for the IC from the agent A. According to the definition of an external behavioural specification the considered property can be represented in the form

$$\varphi_p(t) \Rightarrow \varphi_f(t)$$

where $\varphi_p(t)$ is a formula

$\exists$t2≤t holds_at(has_state(B, communicated(request_from_to_for(A, B, IC))), t2)

and $\varphi_f(t)$ is a formula

$\forall$t1>t [holds_at(has_state(B, observed(provided_result_from_to(E, B, IC))), t1) $\Rightarrow$
holds_at(has_state(B, communicated(message_from_to(B, A, IC))), t1+c)]

with $\varphi_{cond}(t, t1)$ is

holds_at(has_state(B, observed(provided_result_from_to(E, B, IC))), t1)

and $\varphi_{bh}(t1)$ is

holds_at(has_state(B, communicated(message_from_to(B, A, IC))), t1+c)],

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

4 From Interaction States to Memory States

In this section the part of the executable specification describing the basic steps from interaction states to memory states is addressed. Here the past part $\varphi_p(\gamma, t)$ of the behavioural specification is taken and encoded in a memory state. Memory states are represented by memory formulae in the following form.

Definition 4.1 (Memory formula)
The formula $\varphi_{mem}(\gamma, t)$ obtained by replacing all occurrences in $\varphi_p(\gamma, t)$ of subformulae of the form holds_at(p, γ, t') by holds_at(memory(t', p), γ, t) is called the memory formula for $\varphi_p(\gamma, t)$.

Thus, a memory formula defines a sequence of past events (i.e., a history of observations of an external world and actions) for the present time point t. For the considered example $\varphi_{mem}(t)$ is obtained from $\varphi_p(t)$ as $\exists t2 \leq t$ holds_at(memory(t2, has_state(B, communicated(request_from_to_for(A, B, IC)))), t).

The memory formula is no state formula yet. To obtain a memory state formula, normalization of the memory formula for $\varphi_p(\gamma, t)$ is performed by using Lemma 4.1 below. This Lemma will also be used to obtain other types of state formulae in Sections 5 and 6.

Lemma 4.1 (Normalization to State Formula)
Let t be a given time point. If a formula $\delta(\gamma, t)$ only contains temporal relations such as $t' < t''$ and $t' \leq t''$, and atoms of the form holds_at(p, γ, t) for some state formula p, and the given time point t, then some state formula q(t) can be constructed such that $\delta(\gamma, t)$ is equivalent to the formula $\delta^*(\gamma, t)$ of the form holds_at(q(t), γ, t).

To prove the lemma, $\delta^*(\gamma, t)$ is constructed using the following procedure:

(1) In the formula $\delta(\gamma, t)$ replace all temporal relations such as $t' < t''$ and $t' \leq t''$ by holds_at($t' < t''$, γ, t) and holds_at($t' \leq t''$, γ, t) respectively.

(2) Proceed by induction on the composition of the formula $\delta(\gamma, t)$. In particular, conjunction is treated as follows:

By induction hypothesis:

$\delta1(\gamma, t) \Leftrightarrow$ holds_at(p1, γ, t) (which is $\delta1^*(\gamma, t)$)

$\delta2(\gamma, t) \Leftrightarrow$ holds_at(p2, γ, t) (which is $\delta2^*(\gamma, t)$)

Then

$\delta(\gamma, t) \Leftrightarrow$ holds_at(p1, γ, t) & holds_at(p2, γ, t) $\Leftrightarrow$ holds_at(p1 $\wedge$ p2 , γ, t) (which becomes $\delta^*(\gamma, t)$)

Such a procedure can be transformed in an obvious fashion to a recursive algorithm for normalization to a state formula.

Definition 4.2 (Normalized Memory State Formula)
The state formula constructed by Lemma 4.1 for a memory formula $\varphi_{mem}(\gamma, t)$ is called *the (normalized) memory state formula for* $\varphi_{mem}(\gamma, t)$ and denoted by $q_{mem}(t)$. Moreover, q_{mem} is the state formula $\forall u'$ [present_time(u') $\rightarrow q_{mem}(u')$].

The normalized memory state formula for $\varphi_{mem}(\gamma, t)$ uniquely describes the present state at the time point t by a certain history of events. For the considered example $q_{mem}(t)$ for $\varphi_{mem}(t)$ is specified as:

$\exists u2 \leq t$ memory(u2, has_state(B, communicated(request_from_to_for(A, B, IC))))

Lemma 4.2 (Memory Formula and Memory State Formula)
If time has the properties correctness and uniqueness, then the memory formula is equivalent to the (normalized) memory state formula:

$\varphi_{mem}(\gamma, t) \Leftrightarrow$ holds_at($q_{mem}(t)$, γ, t) & holds_at($q_{mem}(t)$, γ, t) $\Leftrightarrow$ holds_at(q_{mem}, γ, t)

Additionally, memory state persistency properties are composed for all memory atoms. It is assumed that a component does not forget information and its memory states related to particular time points in the past persist forever. Rules that describe creation and persistence of memory atoms are given in the executable theory from observation states to memory states $Th_{o\rightarrow m}$ described in Definition 4.3.

Definition 4.3 (Executable Theory from Interaction to Memory $Th_{o\rightarrow m}$)

For a given $\varphi(\gamma, t)$ the executable theory from observation states to memory states $\mathrm{Th}_{o\to m}$ consists of the following formulae.
For any atom p occurring in $\varphi_p(\gamma, t)$, expressed in the InteractionOnt(A):

$\forall t'$ holds_at(p, γ, t') $\Rightarrow$ holds_at(memory(t', p), γ, t')
$\forall t''$ holds_at(memory(t', p), γ, t'') $\Rightarrow$ holds_at(memory(t', p), γ, t''+1)
holds_at(present_time(0), γ, 0)
$\forall t$ holds_at(present_time(t), γ, t) $\Rightarrow$ holds_at(present_time(t+1), γ, t+1)

The last two rules are assumed to be included into the two theories $\mathrm{Th}_{m\to p}$ and $\mathrm{Th}_{p\to o}$ defined in subsequent sections as well.

For the example the rules for creation and persistence of memory atoms are specified as follows:

$\forall t'$ holds_at(has_state(B, communicated(request_from_to_for(A, B, IC))), t') $\Rightarrow$
holds_at(memory(t', has_state(B, communicated(request_from_to_for(A, B, IC))), t')
$\forall t''$ holds_at(memory(t', has_state(B, communicated(request_from_to_for(A, B, IC)))), t'') $\Rightarrow$
holds_at(memory(t', has_state(B, communicated(request_from_to_for(A, B, IC)))), t''+1)

The following Proposition expresses in what sense the executable theory guarantees that memory states are created that are faithful.

Proposition 4.1 (Relating Past Formula and Memory State)
Let $\varphi_p(\gamma, t)$ be a past statement for a given t, $\varphi_{mem}(\gamma, t)$ the memory formula for $\varphi_p(\gamma, t)$, $q_{mem}(t)$ the normalized memory state formula for $\varphi_{mem}(\gamma, t)$, and $\mathrm{Th}_{o\to m}$ the executable theory from the interaction states for $\varphi_p(\gamma, t)$ to the memory states. Then, in theory $\mathrm{Th}_{o\to m}$ the past statement is equivalent to the (normalized) memory state formula:

$$\mathrm{Th}_{o\to m} \models [\varphi_p(\gamma, t) \Leftrightarrow \varphi_{mem}(\gamma, t)]$$

and

$$\mathrm{Th}_{o\to m} \models [\ \varphi_p(\gamma, t) \Leftrightarrow \text{holds\_at}(q_{mem}(t), \gamma, t) \quad \& \quad \text{holds\_at}(q_{mem}(t), \gamma, t) \Leftrightarrow \text{holds\_at}(q_{mem}, \gamma, t)\ ].$$

5 From Memory States to Preparation States

This section describes the executable theory for the basic steps from memory states to preparation states. First the $\varphi_{cond}(\gamma, t, t_1)$ part of the future formula in the behavioural specification is taken and encoded in memory state in a similar manner as the past formula was handled in Section 4.

Definition 5.1 (Condition Memory Formula)
Obtain the *condition memory state formula* $\varphi_{cmem}(\gamma, t, t_1)$ by replacing all occurrences in $\varphi_{cond}(\gamma, t, t_1)$ of holds_at(p, γ, t') by holds_at(memory(t', p), γ, t_1) .

The condition memory formula $\varphi_{cmem}(\gamma, t, t_1)$ describes a history of events, between the time point t, when $\varphi_p(\gamma, t)$ is true and the time point t_1, when the formula $\varphi_{cond}(\gamma, t, t_1)$ becomes true. For the considered example $\varphi_{cmem}(t, t_1)$ is obtained from $\varphi_{cond}(t, t_1)$ as:

holds_at(memory(t', has_state(B, observed(provided_result_from_to(E, B, IC)))), t_1)

Definition 5.2 (Normalized Condition State Formula)
The state formula constructed by Lemma 4.1 for the condition memory formula $\varphi_{cmem}(\gamma, t, t_1)$ is called *the (normalized) condition state formula for* $\varphi_{cmem}(\gamma, t, t_1)$ and denoted by $q_{cond}(t, t_1)$. Moreover, $q_{cond}(t)$ is the state formula $\forall u'$ [present_time(u') $\rightarrow q_{cond}(t, u')$].

Lemma 5.1 (Condition Memory Formula and Condition State Formula)
If time has the properties correctness and uniqueness, then the condition memory formula is equivalent to the (normalized) condition state formula:

$$\varphi_{cmem}(\gamma, t, t_1) \Leftrightarrow \text{holds\_at}(q_{cond}(t, t_1), \gamma, t_1) \;\&\; \text{holds\_at}(q_{cond}(t, t_1), \gamma, t_1) \Leftrightarrow \text{holds\_at}(q_{cond}(t), \gamma, t_1)$$

For the considered example $q_{cond}(t, t1)$ for $\varphi_{cmem}(\gamma, t)$ is obtained as

memory(t', has_state(B, observed(provided_result_from_to(E, B, IC))))

and $q_{cond}(t)$ as

∀u' [present_time(u') → memory(u', has_state(B, observed(provided_result_from_to(E, B, IC))))].

Next the $\varphi_{bh}(\gamma, t_1)$ part of the future formula is considered.

Definition 5.3 (Preparation Formula)
Obtain the *preparation formula* $\varphi_{prep}(\gamma, t_1)$ by replacing in $\varphi_{bh}(\gamma, t_1)$ any occurrence of $\text{holds\_at}(a, \gamma, t_1{+}c)$ for some number c and output a by $\text{holds\_at}(\text{preparation\_for\_output}(t_1{+}c, a), \gamma, t_1)$.

The preparation state is created at the same time point t_1, when the condition $\varphi_{cond}(\gamma, t, t_1)$ for an output is true.

Definition 5.4 (Normalized Preparation State Formula)
The state formula constructed by Lemma 4.1 for the preparation formula $\varphi_{prep}(\gamma, t_1)$ is called *the (normalized) preparation state formula for* $\varphi_{prep}(\gamma, t_1)$ and denoted by $q_{prep}(t_1)$. Moreover, q_{prep} is the state formula ∀u' [present_time(u')] → q_{prep}(u')]

For the considered example $q_{prep}(t_1)$ is composed as

preparation_for_output(t1+c, has_state(B, communicated(message_from_to(B, A, IC))))

Lemma 5.2 (Preparation Formula and Preparation State Formula)
If time has the properties correctness and uniqueness, then the preparation formula is equivalent to the (normalized) preparation state formula:

$$\varphi_{prep}(\gamma, t_1) \Leftrightarrow \text{holds\_at}(q_{prep}(t_1), \gamma, t_1) \;\&\; \text{holds\_at}(q_{prep}(t_1), \gamma, t_1) \Leftrightarrow \text{holds\_at}(q_{prep}, \gamma, t_1)$$

Definition 5.5 (Conditional Preparation Formula)
Let $q_{cond}(t, t_1)$ be the normalized condition state formula for $\varphi_{cmem}(\gamma, t, t1)$ and $q_{prep}(t_1)$ the normalized preparation state formula for $\varphi_{prep}(\gamma, t_1)$. The formula $\varphi_{cprep}(\gamma, t)$ of the form

$$\text{holds\_at}(\forall u_1{>}t\, [q_{cond}(t, u_1) \rightarrow q_{prep}(u_1)], \gamma, t)$$

is called the *conditional preparation formula* for $\varphi_f(\gamma, t)$.

Definition 5.6 (Normalized Conditional Preparation State Formula)
The state formula

$$\forall u_1{>}t\, [\, q_{cond}(t, u_1) \rightarrow q_{prep}(u_1)\,]$$

is called *the normalized conditional preparation state formula for* $\varphi_{cprep}(\gamma, t)$ and denoted by $q_{cprep}(t)$. Moreover, q_{cprep} is the formula

$$\forall u' \text{ [present\_time}(u') \rightarrow q_{cprep}(u') \text{]}$$

Lemma 5.3 (Conditional Preparation and Conditional Preparation State Formula)
If time has the properties correctness and uniqueness, then the conditional preparation formula is equivalent to the (normalized) conditional preparation state formula:

$$\varphi_{cprep}(\gamma, t) \Leftrightarrow \text{holds\_at}(q_{cprep}(t), \gamma, t) \quad \& \quad \text{holds\_at}(q_{cprep}(t), \gamma, t) \Leftrightarrow \text{holds\_at}(q_{cprep}, \gamma, t)$$

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* $\text{Th}_{m\rightarrow p}$.

Definition 5.7 (Executable Theory From Memory to Preparation $\text{Th}_{m\rightarrow p}$)
For any state atom p occurring in $\varphi_{cond}(\gamma, t, t_1)$, expressed in the InteractionOnt(A)[2]:

$\forall t'$ holds_at(p, γ, t') $\Rightarrow$ holds_at(memory(t', p) $\wedge$ stimulus_reaction(p), γ, t')

$\forall t''$, t' holds_at(memory(t', p), γ, t'') $\Rightarrow$ holds_at(memory(t', p), γ, t''+1)

$\forall t'$ holds_at(q_{mem}, γ, t') $\Rightarrow$ holds_at(q_{cprep}, γ, t')

$\forall t'$, t holds_at($q_{cprep} \wedge q_{cond}(t) \wedge \wedge_p$ stimulus_reaction(p), γ, t') $\Rightarrow$ holds_at(q_{prep}, γ, t')

$\forall t'$ holds_at(stimulus_reaction(p) $\wedge \neg$ preparation_for_output(t'+c, a), γ, t') $\Rightarrow$ holds_at(stimulus_reaction(p), γ, t'+1)

$\forall t'$ holds_at(preparation_for_output(t'+c, a) $\wedge \neg$a, γ, t') $\Rightarrow$ holds_at(preparation_for_output(t'+c, a), γ, t'+1)

$\forall t'$ holds_at(present_time(t') $\wedge$ [present_time(u') $\rightarrow$ preparation_for_output(u'+c, a)], γ, t') $\Rightarrow$
holds_at(preparation_for_output(t'+c, a), γ, t')

where a is an action or a communication for which holds_at(a, γ, t'+c) occurs in $\varphi_f(\gamma, t)$.

Note that the last rule in the theory can be derived from other rules of the theory and lemmas, and was introduced only for convenience purposes to support the following proofs.

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

For the considered example:

$\forall t'$ holds_at(has_state(B, observed(provided_result_from_to(E,B,IC))), t') $\Rightarrow$
holds_at(memory(t', has_state(B, observed(provided_result_from_to(E,B,IC)))) $\wedge$
stimulus_reaction(has_state(B, observed(provided_result_from_to(E,B,IC)))), t')

$\forall t''$ holds_at(memory(t', has_state(B, observed(provided_result_from_to(E,B,IC)))), t'')
holds_at(memory(t', has_state(B, observed(provided_result_from_to(E,B,IC)))), t''+1)

$\forall t'$ holds_at($\forall u''$ [present_time(u'')$\rightarrow \exists u2$ [memory(u2, has_state(B, observed(provided_result_from_to(E,B,IC))))]], t')$\Rightarrow$
holds_at($\forall u'''$[present_time(u''')$\rightarrow$ [$\forall u1>u'''$ [memory(u1, has_state(B, observed(provided_result_from_to(E, B, IC)))) $\rightarrow$
preparation_for_output(u1+c, has_state(B, communicated(message_from_to(B, A, IC)))))

$\forall t'$,t holds_at([$\forall u'''$ [present_time(u''')$\rightarrow$ [$\forall u1>u'''$ [memory(u1, has_state(B, observed(provided_result_from_to(E, B, IC)))) $\rightarrow$
preparation_for_output(u1+c, has_state(B, communicated(message_from_to(B, A, IC)))) $\wedge$ $\forall u''$
[present_time(u'')$\rightarrow$ memory(u'', has_state(B, observed(provided_result_from_to(E, B, IC))))] $\wedge$
stimulus_reaction(has_state(B, observed(provided_result_from_to(E,B,IC))))], t') $\Rightarrow$
holds_at($\forall u1$ [present_time(u1) $\rightarrow$
preparation_for_output(u1+c, has_state(B, communicated(message_from_to(B, A, IC))))], t')

[2] If a future formula does not contain a condition, then stimulus_reaction atoms are generated from the corresponding past formula

∀t' holds_at([stimulus_reaction(has_state(B, observed(provided_result_from_to(E,B,IC)))) ∧
not(preparation_for_output(t'+c, has_state(B, communicated(message_from_to(B, A, IC)))))], t') ⇒
holds_at(stimulus_reaction(has_state(B, observed(provided_result_from_to(E,B,IC)))), t'+1)

∀t' holds_at(preparation_for_output(t'+c, has_state(B, communicated(message_from_to(B, A, IC)))) ∧
not(has_state(B, communicated(message_from_to(B, A, IC)))), t') ⇒
holds_at(preparation_for_output(t'+c, has_state(B, communicated(message_from_to(B, A, IC)))), t'+1).

Proposition 5.1 (Relating Preparation Formula and Preparation State Formula)
Let $\varphi_f(\gamma, t)$ be a future statement for t of the form $\forall t_1>t\ [\varphi_{cond}(\gamma, t, t_1) \Rightarrow \varphi_{bh}(\gamma, t_1)]$, where $\varphi_{cond}(\gamma, t, t_1)$ is an interval statement, which describes a condition for one or more actions and/or communications and $\varphi_{bh}(\gamma, t_1)$ is a (conjunction of) future statement(s) for t_1, which describes action(s) and/or communications that are to be performed; let $\varphi_{prep}(\gamma, t_1)$ be the preparation formula, $\varphi_{cprep}(\gamma, t)$ be the conditional preparation formula for $\varphi_f(\gamma, t)$, $q_{cprep}(t)$ be the normalized conditional preparation state formula for $\varphi_{cprep}(\gamma, t)$, and $Th_{m\rightarrow p}$ the executable theory for $\varphi(\gamma, t)$ from memory states to preparation states. Then, in theory $Th_{m\rightarrow p}$ condition preparation formula is equivalent to the (normalized) condition preparation state formula:

$$Th_{m\rightarrow p} \models \forall t_1>t\ [\varphi_{cond}(\gamma, t, t_1) \Rightarrow \varphi_{prep}(\gamma, t_1)] \Leftrightarrow \varphi_{cprep}(\gamma, t)]$$

and

$$Th_{m\rightarrow p} \models [\forall t_1>t\ [\varphi_{cond}(\gamma, t, t_1) \Rightarrow \varphi_{prep}(\gamma, t_1)] \Leftrightarrow \text{holds\_at}(q_{cprep}(t), \gamma, t) \quad \&$$
$$\text{holds\_at}(q_{cprep}(t), \gamma, t) \Leftrightarrow \text{holds\_at}(q_{cprep}, \gamma, t)\ ].$$

6 From Preparation States to Output States

The preparation state preparation_for_output(t_1+c, a) is followed by the output state, created at 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)* $Th_{p\rightarrow o}$.

Definition 6.1 (Executable Theory from Preparation to Output $Th_{p\rightarrow o}$**)**
For a given $\varphi_f(\gamma, t)$ the executable theory from the preparation to the output state(s) consists of the formula

∀t' holds_at(preparation_for_output(t'+c, a), γ, t') ⇒ holds_at(a, γ, t'+c)

where c is a number and a an action or a communication for which holds_at(a, γ, t'+c) occurs in $\varphi_f(\gamma, t)$.

For the considered example the following rule is generated:

∀t' holds_at(preparation_for_output(t'+c, has_state(B, communicated(message_from_to(B, A, IC)))), t') ⇒
holds_at(has_state(B, communicated(message_from_to(B, A, IC))), t'+c)

Proposition 6.1 (Relating Preparation Formula and Behaviour Formula)
Let $\varphi_{bh}(\gamma, t_1)$ be a (conjunction of) future statement(s) for t_1, which describes action(s) and/or communications that are to be performed, $\varphi_{prep}(\gamma, t_1)$ be the preparation formula and $Th_{p\rightarrow o}$ the executable theory from preparation states to output states. Then,

$$Th_{p\rightarrow o} \models [\ \varphi_{prep}(\gamma, t_1) \Rightarrow \varphi_{bh}(\gamma, t_1)\ ]$$

7 Combining the Executable Theories to Obtain the Refinement

In this section the sets of executable properties, identified in Sections 4 to 6 are combined to construct the specification $\pi(\gamma,t) = Th_{o\rightarrow m} \cup Th_{m\rightarrow p} \cup Th_{p\rightarrow o}$ (considered as conjunction), which describes a refinement of $\varphi(\gamma,t)$. The following theorem proves the existence of such a refinement for every external behavioural specification.

Theorem 7.1 (Existence of Executable Refinement)

Every external behavioural specification can be refined into an executable specification. To obtain such a refinement, the automated transformation procedure can be used as described in Sections 4 to 6.

Proof.

According to Definition 3.4 an executable specification $\pi(\gamma, t)$ refines an externally observable dynamic property $\varphi(\gamma, t)$ iff

(1) for any trace γ $\forall t\ \ \pi(\gamma, t) \Rightarrow \varphi(\gamma, t)$

(2) for any trace γ_1 exists trace γ_2 such that $\forall t$ [$\varphi(\gamma_1, t) \Rightarrow$ [coincide_on(γ_1, γ_2, InteractionOnt(A)) & $\pi(\gamma_2, t)$]

The first condition can be reformulated as

$$Th_{o\to m} \cup Th_{m\to p} \cup Th_{p\to o} \models \varphi(\gamma,t)$$

Here $\varphi(\gamma,t)$ is of the form $[\varphi_p(\gamma, t) \Rightarrow \varphi_f(\gamma, t)]$ with $\varphi_f(\gamma, t)$ future statement for t of the form $\forall t_1{>}t\ [\varphi_{cond}(\gamma, t, t_1) \Rightarrow \varphi_{bh}(\gamma, t_1)]$. As a basis we use Propositions 4.1, 5.1 and 6.1. These propositions relate the past formula to the memory state formula, the preparation formula to the preparation state formula, and the preparation formula to the behaviour formula, respectively:

(1) $Th_{o\to m} \models [\varphi_p(\gamma, t) \Leftrightarrow$ holds_at(q_{mem}, γ, t)]

(2) $Th_{m\to p} \models [\,[\forall t_1{>}t\ [\varphi_{cond}(\gamma, t, t_1) \Rightarrow \varphi_{prep}(\gamma, t_1)] \Leftrightarrow$ holds_at(q_{cprep}, γ, t)]

(3) $Th_{p\to o} \models [\ \varphi_{prep}(\gamma, t_1) \Rightarrow \varphi_{bh}(\gamma, t_1)\ ]$

Moreover, as this executable rule is included in $Th_{m\to p}$ (see Definition 5.7), it holds:

(4) $Th_{m\to p} \models$ [$\forall t'$ holds_at(q_{mem}, γ, t') $\Rightarrow$ holds_at(q_{cprep}, γ, t')]

Based on these four lines, it can be seen that $\pi(\gamma,t)$ indeed satisfies the first criterion for refinement, by the following steps in the theory $\pi(\gamma,t) = Th_{o\to m} \cup Th_{m\to p} \cup Th_{p\to o}$:

- when $\varphi_p(\gamma, t)$ holds, also holds_at(q_{mem}, γ, t) holds (1)
- when holds_at(q_{mem}, γ, t) holds, also holds_at(q_{cprep}, γ, t) holds (4)
- when holds_at(q_{cprep} ,γ, t) holds, also $\forall t_1{>}t\ [\varphi_{cond}(\gamma, t, t_1) \Rightarrow \varphi_{prep}(\gamma, t_1)]$ holds (2)
- when $\forall t_1{>}t\ [\varphi_{cond}(\gamma, t, t_1) \Rightarrow \varphi_{prep}(\gamma, t_1)]$ holds, also $\forall t_1{>}t\ [\varphi_{cond}(\gamma, t, t_1) \Rightarrow \varphi_{bh}(\gamma, t_1)]$ holds (3)
- hence $\varphi_p(\gamma, t) \Rightarrow \forall t_1{>}t\ [\varphi_{cond}(\gamma, t, t_1) \Rightarrow \varphi_{bh}(\gamma, t_1)]$ holds, which is $\varphi(\gamma, t)$.

For the second criterion of refinement, first of all notice that:

- the consequents of the executable rules in $\pi(\gamma,t)$ always are atoms, never negations
- these consequent atoms are always internal atoms, except the consequent of the executable rule in $Th_{p\to o}$, which is the interaction atom holds_at(a, γ, t'+c) .

For any trace γ, let

diag(γ, Ont) = { holds_at(a, γ, t_1) | $\gamma(t_1) \models$ a & a literal in Ont }

Given these observations, let any trace γ be given such that $\varphi(\gamma, t)$ holds. Construct the trace γ' such that γ' is equal to γ for the interaction ontology, and complies to the executable rules for the internal atoms, as follows:

$\gamma'(t_1) \models b \Leftrightarrow \gamma(t_1) \models b$	for any interaction literal b
$\gamma'(t_1) \models a$	if a is an internal atom and diag(γ, InteractionOnt) $\cup\ \pi(\gamma,t) \models$ [holds_at(a, γ, t_1)]
$\gamma'(t_1) \models \neg a$	if a is an internal atom and not diag(γ, InteractionOnt) $\cup\ \pi(\gamma,t) \models$ [holds_at(a, γ, t_1)]

By this construction all executable rules hold for γ', except possibly the rule from $Th_{p\rightarrow o}$. This last rule is the remaining issue to be addressed. Suppose this rule does not hold for γ'. Then a t_1 exists such that the antecedent holds, but not the consequent:

holds_at(preparation_for_output(t_1+c, a), γ', t_1) &

not holds_at(a, γ', t_1+c)

From the construction of the trace γ' and Definition 5.3 it follows that the preparation atom preparation_for_output(t_1+c, a) is based on the occurrence of holds_at(a, γ, t_1+c) in $\varphi_{bh}(\gamma, t_1)$. Moreover, the preparation atom is derivable from $\pi(\gamma, t)$ so it originates from a condition formula and a memory state formula that both hold for γ'. By Propositions 4.1 and 5.1 it holds that

$\varphi_p(\gamma', t)$

$\varphi_{cond}(\gamma', t, t_1)$

Since these are formula based on InteractionOnt, and γ and γ' coincide on InteractionOnt, by Lemma 3.1(2) also

$\varphi_p(\gamma, t)$

$\varphi_{cond}(\gamma, t, t_1)$

hold. As $\varphi(\gamma, t)$ holds, this implies that $\varphi_{bh}(\gamma, t_1)$ holds. Moreover, it was found that holds_at(a, γ, t_1+c) occurs in $\varphi_{bh}(\gamma, t_1)$. Therefore, holds_at(a, γ, t_1+c) holds, and again, since γ and γ' coincide on InteractionOnt, this implies that holds_at(a, γ', t_1+c) holds, which is a contradiction. This shows that the second criterion of refinement is fulfilled, which completes the proof of Theorem 7.1. ■

8 Transformation into a General Description for a Finite State Transition System

In this and the following Section 9 transformation is described of an executable specification obtained by the procedure from Section 3 into a finite state transition system description: First, an executable specification is transformed into the general description for a finite state transition system as described in this section. Then, the obtained description is transformed further into SMV format as described in Section 9.

For the purposes of practical analysis (e.g., by performing simulations and verification of logical consequences) a specification based on executable temporal logical properties generated by the procedure described in the previous sections 4-7 is translated into a finite state transition system model. The translation is based on the fact that a computation (in our case the execution of temporal logical properties) is essentially an (infinite) sequence of states (Vardi, 1996). Therefore, similarly to the approach from (Vardi, 1996), given an executable temporal specification one can construct a finite state transition system that generates the set of traces (by all possible executions of transition rules) equivalent to the set produced by all possible execution of temporal logical properties from the specification.

In computer science a finite state transition system is often described by a tuple $\langle Q, Q_0, \Sigma, \rightarrow\rangle$, where Q is a finite set of states of an agent, $Q_0 \subseteq Q$ is a set of initial states, Σ is a set of labels or events, which trigger the transition and $\rightarrow\ \subseteq Q \times \Sigma \times Q$ is a set of transitions. Such a representation often assumes an explicit denotation for every state in a transition system, which can be very numerous. However, a more compact representation, close to the production systems style, in the form of a set of transition rules with variables is possible (Arnold, 1994).

Definition 8.1 (General Representation of a Finite State Transition System)
Let Ont be a state ontology consisting of sorts, constants, functions and predicates. Let At(Ont) be the set of (many-sorted predicate logic) atoms over Ont (possibly with variables). A general representation for a finite state transition system over Ont consists of transition rules of the form [P $\twoheadrightarrow$ N], where P is a proposition based on atoms from At(Ont), and N is a conjunction of atoms from At(Ont). The meaning is that when a certain instance of P by a certain variable assignment is true in a state, then the instance of N by the same variable assignment will be true in the next state; here $\twoheadrightarrow$ is a symbol for the transition between the two states.

Note that this (predicate-logic-based) transition format can be translated into one without variables (propositional-logic-based) by replacing every transition rule by all of its instances.

By considering all the possible executions of transition rules from such a representation, all the possible states (without explicit names) and transition paths of the considered transition system can be generated. Such a general representation for a finite state transition system has as an advantage that it does not depend on any particular implementation (e.g., verification or simulation tools). However, as this generic format describes states and transitions between them, it can be relatively easy translated into specialized languages of existing tools, based on the finite state transition system representation. Thus, having a general finite state transition system specification, one can perform different forms of analysis using existing tools. In particular, in this paper it is shown how a general finite state transition system specification can be transformed into the format of the SMV model checker. SMV is used in this paper for verification of higher level properties of a multi-agent system.

In the following the translation procedure from an executable specification into the general representation for a finite state transition system will be described. As it was shown in the previous section an executable specification consists of the rules of three executable theories: $Th_{o\rightarrow m}$, $Th_{m\rightarrow p}$ and $Th_{p\rightarrow o.}$ To translate an executable specification into the finite state transition system format, for each rule from the executable specification the corresponding transition rule should be created.

Let us first consider the formulae from the theory $Th_{o\rightarrow m}$. To relate states of a transition system to the timeline used in these rules the unary predicate present_time is used. The atom present_time(t) being true in a given state indicates that t is the time in this state. Furthermore, the assumption from $Th_{o\rightarrow m}$ that an observation state and a corresponding memory state are created at the same time point should be preserved. Thus, the time increment rules are defined as:

present_time(0) $\wedge \neg$p $\twoheadrightarrow$ present_time(1)
present_time(t) $\wedge \neg q_{mem} \wedge \neg$p $\twoheadrightarrow$ present_time(t+1)

Now, when a relation between states and time points is established, the rules defined in the $Th_{o\rightarrow m}$ can be easily translated into the transition system format as it is shown in Table 1.

Table 1. Translation of the formulae from the executable theory $Th_{o\rightarrow m}$ into the corresponding finite state transition rules

Rule from the executable theory $Th_{o\rightarrow m}$	**Corresponding transition rules**
Memory state creation rule $\forall$t' holds_at(p, γ, t') $\Rightarrow$ holds_at(memory(t', p), γ, t')	present_time(t) $\wedge$ p $\twoheadrightarrow$ memory(t, p)

Memory persistence rule $\forall$t'' holds_at(memory(t', p), γ, t'') $\Rightarrow$ holds_at(memory(t', p), γ, t''+1)	memory(t, p) $\twoheadrightarrow$ memory(t, p)

Next, let us translate the properties from $Th_{m\rightarrow p}$. The time increment rules are created similarly to the $Th_{o\rightarrow m}$ case based on the assumption from $Th_{m\rightarrow p}$ that a preparation state is generated at the same time point, when the condition for an output is true.

present_time(t) $\wedge$ q_{cprep} $\wedge$ $\neg q_{cond}$(t) $\wedge$ $\neg$p $\twoheadrightarrow$ present_time(t+1)

present_time(t) $\wedge$ q_{prep} $\twoheadrightarrow$ present_time(t+1)

Then, the rules defined in the $Th_{m\rightarrow p}$ are translated into the transition system format in a straightforward manner as it is shown in Table 2.

Table 2. Translation of the rules from the executable theory $Th_{m\rightarrow p}$ into the corresponding finite state transition rules

Rule from the executable theory $Th_{m\rightarrow p}$	**Corresponding transition rules**
Memory state creation rule $\forall$t' holds_at(p, γ, t') $\Rightarrow$ holds_at(memory(t', p) $\wedge$ stimulus_reaction(p), γ, t')	present_time(t) $\wedge$ p $\twoheadrightarrow$ [memory(t, p) $\wedge$ stimulus_reaction(p)]
Memory persistence rule $\forall$t'' holds_at(memory(t', p), γ, t'') $\Rightarrow$ holds_at(memory(t', p), γ, t''+1)	memory(t, p) $\twoheadrightarrow$ memory(t, p)
Conditional preparation generation rule $\forall$t' holds_at(q_{mem}, γ, t') $\Rightarrow$ holds_at(q_{cprep}, γ, t')	q_{mem} $\twoheadrightarrow$ q_{cprep}
Preparation state creation rule $\forall$t', t holds_at(q_{cprep} $\wedge$ q_{cond}(t) $\wedge$ $\bigwedge_p$ stimulus_reaction(p), γ, t') $\Rightarrow$ holds_at(q_{prep}, γ, t')	present_time(t') $\wedge$ q_{cprep} $\wedge$ q_{cond}(t) $\wedge$ $\bigwedge_p$ stimulus_reaction(p) $\twoheadrightarrow$ q_{prep}
Preparation state persistence rule $\forall$t' holds_at(preparation_for_output(t'+c, a) $\wedge$ $\neg$ a, γ, t') $\Rightarrow$ holds_at(preparation_for_output(t'+c, a), γ, t'+1)	preparation_for_output(t+c, a) $\wedge$ $\neg$ a $\twoheadrightarrow$ preparation_for_output(t+c, a)
Stimulus reaction state persistence rule $\forall$t' [holds_at(stimulus_reaction(p) $\wedge$ $\neg$preparation_for_output(t'+c, a), γ, t')] $\Rightarrow$ holds_at(stimulus_reaction(p), γ, t'+1)	present_time(t') $\wedge$ stimulus_reaction(p) $\wedge$ $\neg$preparation_for_output(t'+c, a) $\twoheadrightarrow$ stimulus_reaction(p)

The executable theory from preparation to output $Th_{p\rightarrow o}$ contains only one formula that relates a preparation state at the time point t' to an output state at the time point t'+c; its translation is given in Table 3.

Table 3. Translation of the rule from the executable theory $Th_{p\rightarrow o}$ into the corresponding finite state transition rule

Rule from the executable theory $Th_{p\rightarrow o}$	**Corresponding transition rules**
Output generation rule $\forall$t' holds_at(preparation_for_output(t'+c, a), γ, t') $\Rightarrow$ holds_at(a, γ, t'+c)	preparation_for_output(t+c, a) $\wedge$ present_time(t+c-1) $\twoheadrightarrow$ a

By means of the described translation rules the executable properties from the specification for the example considered in this paper are translated into the transition rules as it is shown below:

```
present_time(t) ∧ communicated(request_from_to_for(A1,A2,IC)) ↠
        memory(t,communicated(request_from_to_for(A1,A2,IC)))
present_time(t) ∧ observed(provided_result_from_to(E,A2,IC)) ↠
        memory(t,observed(provided_result_from_to(E,A2,IC)) ∧
        stimulus_reaction(observed(provided_result_from_to(E,A2,IC)))
memory(t,communicated(request_from_to_for(A1,A2,IC))) ↠
          memory(t,communicated(request_from_to_for(A1,A2,IC)))
memory(t,observed(provided_result_from_to(E,A2,IC))  ↠
        memory(t,observed(provided_result_from_to(E,A2,IC))
present_time(t) ∧
   ∃u2≤t memory(u2,communicated(request_from_to_for(A1, A2, IC))) ↠
      conditional_preparation_for_output(communicated(send_from_to(A2,A1,IC)))
present_time(t) ∧
conditional_preparation_for_output(communicated( send_from_to(A2,A1,IC))) ∧
memory(t,observed(provided_result_from_to(E,A2, IC))) ∧
stimulus_reaction(observed(provided_result_from_to(E,A2,IC)))  ↠
        preparation_for_output(t+c,communicated(send_from_to(A2,A1,IC)))
present_time(t) ∧
stimulus_reaction(observed(provided_result_from_to(E,A2, IC))) ∧
not(preparation_for_output(t+c,communicated(send_from_to(A2,A1,IC)))) ↠
           stimulus_reaction(observed(provided_result_from_to(E,A2,IC)))
preparation_for_output( t+c,communicated(send_from_to(A2,A1,IC)))∧
 not( communicated(send_from_to(A2,A1,IC))) ↠
        preparation_for_output(t+c,communicated(send_from_to(A2,A1,IC)))
preparation_for_output(t+c,communicated(send_from_to(A2,A1,IC))) ∧
present_time(t+c-1) ↠
         communicated(send_from_to(A2,A1,IC)).
```

The obtained general representation for a finite state transition system will be further used as a model for the model checker SMV (McMillan, 1993). By means of the SMV will be performed the automatic verification of relationships between dynamic properties of components of different aggregation levels. For this purpose a procedure was developed for translating the general description of a transition system into the input format of the SMV model checking tool. The description of this procedure is given in the next section.

9 Transformation into SMV Format

For automatic verification of interlevel relationships between dynamic properties of different aggregation levels by means of model checking techniques, the representation of a finite state transition system, corresponding to a behavioural specification of the lower aggregation level should be translated into the input format of one of the existing model checkers. The model checker SMV has been chosen as a verification tool for two reasons. First, the input language of SMV is syntactically and semantically close to the general description of a finite state transition system, which facilitates automatic translation into the SMV input format. Second, SMV uses efficient symbolic algorithms to traverse a model and the expressive temporal logic CTL for specifying properties to check.

A transition system specification in SMV consists of a number of sections. In the section labeled VAR the names and types of the variables used in the model are defined. The type associated with a variable is either Boolean, scalar, or an array. In the second section labeled ASSIGN the initial values of variables are defined (i.e., the values that the variables have in the initial state) and the transition rules between states are specified. To initialize a variable var with a value val in SMV the construct init(var):=val is used. The transition rules are specified by case-expressions that define the change of values of the variables of the transition system as follows:

```
next (var) := case
                    boolean_expression: val;
            esac
```

All case-expressions are evaluated in every state. When boolean_expression on the left-hand side of ":" of some transition rule is evaluated to true in some state, then the corresponding variable var will receive the value val in the next state.

Let us describe in a condensed form[3] the main steps of the transformation procedure, which is automatically performed by the dedicated software that has been developed.

First, using the standard rules (Fitting, 1996) $q_{mem}(t)$ and $q_{cond}(t, t_1)$ expressions for each dynamic property DP_n are transformed into prenex normal form. Then, for each dynamic property the steps 1-3 described below are applied first to $q_{mem}(t)$ and then to $q_{cond}(t, t_1)$. After that conditional preparation generation rules are added by performing the step 4. Finally, the preparation and output state creation rules are generated for each dynamic property by performing the step 5.

Step 1. For each occurrence of an existential quantifier of the form ∃t1 P(t1), where t1 is a time variable name and P(t1) is some function of the form memory(observed(t1, obs_event)), ¬memory(observed(t1, obs_event)), memory(t1, act_event), or ¬memory(t1, act_event), where obs_event and act_event are some atoms and for each occurrence of a universal quantifier of the form ∀t1 P(t1), create an atom (a label) t1 and add to the specification the corresponding initialization rules.

Step 2. For each occurrence of the expression Q t1, t2 R t1 memory(observed(t1, obs_event)), where Q is either an existential or a universal quantifier, R is the comparison relation for the linear ordered time line: R∈{<, ≤}; t1 and t2 are time variables, add to the specification the following rule:

```
next(t1):= case
            t2 & obs_event: 1; //memory state creation
            !t2: 0;
            1: t1;             //persistence of memory
esac;
```

Similar rules should be added for the expressions Q t1, t2 R t1 memory(t1, act_event), Q t1, t2 R t1 ¬memory(observed(t1, obs_event)) and Q t1, t2 R t1 ¬memory(t1, act_event).

Step 3. For each expression of the form ∃t1, t2 ∀t3 [t3 R t2 AND t1 R t3 AND memory(observed(t1, obs_event1)) AND memory(observed(t2, obs_event2)) & P3(t3)]:

(a) if P3(t) is of the form memory(observed(t3, obs_event))

i. For t3 < t2 and t1< t3 add to the specification the following rules:

```
t3t1_eq: boolean ;
init(t3t1_eq):=0;
next(t3t1_eq):= case
            t1: 1;
             1: 0;
esac;
next(t1):= case
            !obs_event2 & !t2 & t3t1_eq & !obs_event3: 0;
            1: t1;
esac;
```

[3] All the technical details of the described procedure are given in Appendix B: http://www.few.vu.nl/~sharp/appendixes_interlevel_relations.pdf

```
next(t3):= case
           !t1: 0;
           !obs_event2 & !t2 & !obs_event3: 0;
           !obs_event2 & !t2 & obs_event3: 1;
           1: t3;
esac;
```

ii. For t3 < t2 and t1≤ t3 add to the specification the following rules:

```
t3t1_eq: boolean ;
init(t3t1_eq):=0;
next(t3t1_eq):= case
            t1: 1;
             1: 0;
esac;
next(t1):= case
            !t2 & t3t1_eq & !obs_event3: 0;
            1: t1;
esac;
next(t3):= case
           !t1: 0;
           !t2 & !obs_event3: 0;
           !t2 & obs_event3: 1;
           1: t3;
esac;
```

iii. For t3 ≤ t2 and t1< t3 add to the specification the following rules:

```
next(t1):= case
            !obs_event2 & !t2 & !obs_event3: 0;
            1: t1;
esac;
next(t3):= case
           !t1: 0;
           !obs_event2 & !t2 & !obs_event3: 0;
           !obs_event2 & !t2 & obs_event3: 1;
           1: t3;
esac;
```

iiii. For t3 ≤ t2 and t1≤ t3 add to the specification the following rules:

```
next(t1):= case
            !t2 & !obs_event3: 0;
            1: t1;
esac;
next(t3):= case
           !t1: 0;
           !t2 & !obs_event3: 0;
           !t2 & obs_event3: 1;
           1: t3;
esac;
```

Similarly for the case, when P3(t) is of the form memory(t3, act_event), ¬memory(observed(t3, obs_event)) and ¬memory(t3, act_event)

Step 4. Add conditional preparation generation rules to the specification:

```
next(fmemN):= case    // N is a number of a dynamic property in the input specification
            ∧ti: 1;   // conjunction of all labels, created based on φp(γ, t)
            i
            1: 0;
esac;
```

Step 5. For each action and communication a function act_event in a formula $q_{bt}(t)$ add to the specification the following rules:

```
next(fprep_act):= case
            fmemN & ∧tj: 1; //conjunction of all labels, created based on φcond(γ, t, t1)
                     j
                      1: 0;
esac;

next(act_event):= case
                fprep_act: 1;
                       1: 0;
esac;
```

10 Case Study

To verify interlevel relations in the context of the multi-agent system of the considered example a number of properties for the higher aggregation level component (the whole multi-agent system) were identified, which further were related to the properties of the lower level components specified in Section 3.

GP1 (Information acquisition initiation effectiveness): At some points in time A and B will start information acquisition to E.

∃t1, t2 [holds_at(has_state(A, obs_focus_from_to_for(A, E, IC1)), t1) &
holds_at(has_state(B, obs_focus_from_to_for(B, E, IC2)), t2)]

GP2(A1) (Information source effectiveness for agent A): If at some point in time A starts information acquisition to E, then E will generate all the correct relevant information for agent A.

∀t1 t1<t & ∀IC [holds_at(has_state(A, obs_focus_from_to_for(A, E, IC)), t1)]
⇒ ∃t2 t2>t & [holds_at(has_state(E, observed(provide_result_from_to(E, A1, IC))), t2)]

GP3 (Concluding effectiveness): If at some point in time E generates all the correct relevant information, then C will receive a correct conclusion.

∀t1, t2 [t1<t & t2<t [holds_at(has_state(E, observed(provide_result_from_to(E, A, IC1))), t1) & holds_at(has_state(E, observed(provide_result_from_to(E, B, IC2))), t2)]]
⇒ ∃t3>t [holds_at(has_state(C, communicated(send_from_to(A, C, IC3))), t3)

A number of interlevel relations identified manually are specified below (formal expressions for lower level properties DP1-DP11 are given in Section 3):

DP9 & DP10 & DP1(A, B) & DP6(A, B) ⇒ GP1 (1)

DP3 (A, E) & DP4(A, E) & DP11 ⇒ GP2(A) (2)

DP3 (B, E) & DP4(B, E) & DP11 ⇒ GP2(B) (3)

DP5 (E, A) & DP5 (E, B) & DP10 & DP1(A, B) & DP7(A, B) & DP2(B, A) &

& DP8(A, B) & DP2(A, C) & DP11⇒ GP3 (4)

From the higher level properties GP1, GP2(A), GP2(B) and GP3 the global system successfulness property can be inferred, which is also a liveness property of the system.

GP (Successfulness): There exists a point in time when agent C will receive a correct conclusion.

∃t ∃IC holds_at(has_state(C, communicated(send_from_to(A, C, IC))), t)

GP1 & GP2(A) & GP2(B) & GP3 ⇒ GP

Now, the identified interlevel relations between dynamic properties of different aggregation levels (1), (2), (3) and (4) can be formally justified (or refuted). For this purpose first for every relationship using the developed software for the steps 1-4 of the transformation procedure (Section 3) the external behavioural specification of the multi-agent system that consists of the lower level properties of the antecedent of the relationship is automatically transformed into an executable behavioural specification. Then, using the other software tool for the steps 5 and 6 of the procedure, every executable behavioural specification is converted into the description of a finite state transition system. In order to perform verification by means of SMV model checker, every general description of the finite state transition system has been automatically translated into the SMV model specification format. Using the state transition system representation, verification of the entailment relations, represented as CTL formulas, can be performed.

For example, for the relation (4) the dynamic property GP3 is expressed in CTL as:

AG (E_ observed_provide_result_from_to_E_A_info & E_ observed_provide_result_from_to_E_B_info
→ ***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 showed that all the relationships between properties (1), (2), (3) and (4) indeed hold with respect to the corresponding models of the multi-agent system.

At the same time, if one excludes property DP8 from the antecedent of the relation (4), model checking proves that the formulated in such way relationship fails. From the counter-example produced by the model checker, it is visible that although agent A has all necessary information to draw a conclusion, it will never send the conclusion to agent C. Thus, by performing such verification, it is possible to reveal mistakes in manually identified relations between properties and improve them.

11 Notes on the Complexity of the Transformation Procedure

The complexity of the representation of the obtained executable model is linear in size of the non-executable behavioural 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);
(4) for every non-executable dynamic property auxiliary variables fmem and fprep (i.e., the variables that indicate truth values of $\varphi_{mem}(\gamma, t)$ and $\varphi_{prep}(\gamma, t_1)$ respectively) and corresponding update rules are introduced;
(5) for every output atom (i.e., action or communication) specified in $\varphi_{bh}(\gamma, t_1)$ a preparation state persistence rule and an output state creation rule are introduced;
(6) for reactivation of agent preparation states the auxiliary variables and the update rules corresponding to communicated and observed functions from $\varphi_{prep}(\gamma, t_1)$ are introduced.

Formally, the number of rules of the executable specification IS, generated from the non-executable specification ES is evaluated as $|IS| = v+nv+3*inp+2*nex+2*outp$, where v is the number of variables in the ES; nv is the number of nested quantifiers in the ES; inp (outp) is the number of functions from the input (output) ontology that occur in the ES; nex is the number of non-executable properties in ES. The number of variables in the IS equals $rules_{IS} - inp$.

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 (McMillan, 1993). In general, the complexity of model checking grows exponentially with increase of the number of state variables and of the branching factor of the corresponding state space. However, the auxiliary variables and rules in an executable specification, generated by the transformation procedure, do not create any new branches in the state space corresponding to the specification, but rather are related linearly (i.e., in sequence) to other states of the space. In particular, an observation state and the corresponding memory state are related by an unconditional transition relation; after a memory (preparation) formula becomes true in some state, the state is created in which the corresponding auxiliary variable fmem (fprep) is assigned the value true; when fprep is true in a state, then the state is unconditionally generated, in which the corresponding action variable is true; the states in which reactivation variables are true, are also related to other states in sequence.

12 Some Implementation Details

To automate the proposed procedure the software tool was developed in Java™. A model that describes dynamics of a system using an interaction ontology should be provided as an ASCII text file with name input.txt in the directory with the translation tool. All dynamic properties should be specified in the format [past formula] implies [future formula], where [future formula] is of the form [conditional formula] implies [action formula]. If the future formula contains a trivial condition (which is always true), then this condition can be omitted, and the corresponding dynamic property should be specified as [past formula] implies [action formula].

In order to enter a new dynamic property into the input file, first the past formula should be entered, then <new line symbol> and the future formula should be entered. If the specified property is not the last one in the specification, then <new line symbol> followed by a combination of symbols "---" and one more <new line symbol> has to be added. More specific technical details for specifying dynamic properties are given below.

- All time variables should be named as t[index], where [index] is a natural number.
- Time variables of both past and future formulae should be related to t, which is a standard variable and should not be additionally introduced (the present time point, with respect to which the formula is being evaluated).
- Names of state atoms should not contain blanks; no white spaces are allowed in predicate expressions.
- There are a number of standard predicates defined:
  - world(t,a): denotes an event a in the external world at a time point t
  - observed(t,a): denotes an observation a of an agent at a time point t
  - communicated(t, a): denotes a communication act a of an agent performed at a time point t
  - output(t,a): denotes an output a at a time point t
- Formulae are built using the following logical connectives and quantifiers:
  - AND: denotes the logical "and"
  - THEN: denotes the logical implication
  - not_a: denotes the negation of an atom a (note that negations can be also applied to the predicates, e.g., not_world(t,a), not_observation(t,not_a))
  - '[','] ': denote brackets for formulas (note that brackets should be always separated by a single blank from the literals, which stand before and after them)
  - At1: denotes a universally quantified variable t1
  - Et1: denotes an existentially quantified variable t1
  - , : denotes a coma (make notice that there should be no blanks between a coma and literals before and after it).
- Other logical connectives can be expressed by means of already mentioned ones.

For example,

Past formula: Et3 t3<t observation(t3,a)

Future formula: At1 t1>=t observation(t1,b) THEN action(t1+2,c)

The transformation algorithm searches in the input file for the standard predicate names and the predefined structures, then performs string transformations that correspond precisely to the described steps of the translation procedure, and adds executable rules to the output specification file. In particular, for every observed atom from past and condition state formulae corresponding memory state generation and memory state persistence rules are formed. During the transformation of dynamic properties into corresponding rules of the executable theory $Th_{o\rightarrow m}$ in the expressions for q_{mem} and q_{cprep}, time variables t[index] are replaced by local time variables u[index], where [index] is a natural number. Additionally, for every observed atom from condition state formulae, a rule for generating stimulus_reaction atom and a stimulus reaction state persistence rule are created. Furthermore, for every output atom the preparation state and the output state generation rules are created. When transformation is finished, the output.txt file with the resulted executable specification is generated.

The transformation tool works on any platform running JRE 1.4 or higher. The processor capacity and the amount of RAM do not bear considerable influence on the time to generate an executable specification. In particular, on a computer system with the Intel Pentium III 850 MHz and 128 Mb RAM, the executable specification for the example considered in this paper was generated in 0.53 seconds.

13 Translating the Executable Format into Various Formats

Although the executable format described in Section 3 is very general, it has much in common with a number of particular executable languages and logics. In this Section we shall consider a number of them: the LEADSTO language, propositional modal temporal logic, monodic first-order temporal logic, and the loosely guarded fragment of first-order predicate logic. These logics have as advantages good computational properties or decidability. For all of these languages and logics dedicated techniques and tools for performing different types of analysis (e.g., by simulation or by verification) are available. The executable format described in this paper is very close to these languages, but nevertheless is generic in the sense that it does not commit to one of them. Therefore, it is easy to make use of these techniques and tools, by simple translations of executable specifications into the considered languages and logics. First, some general translation principles will be described, applicable to all considered languages and logics. Then, more specific translation techniques for the particular languages and logics will be presented and illustrated by examples.

Since most of the considered languages do not allow function symbols, first all functions $f: y \times z \rightarrow v$ in executable specifications and in state properties in particular are replaced by predicates combined_of(f, v, y, z). Furthermore, the holds-relation (|=) in the reified temporal predicate logic expressions is used as a predicate holds_at(X, γ, t) without function symbols in the arguments, which denotes that the state property X holds at time point t in the trace γ. Moreover, when occurring, a universal quantification (over a finite domain) in a state property is replaced by a conjunction of propositions and similarly an existential quantification is replaced by a disjunction of propositions. More specifically,

$\forall x$: SORT P(x) is replaced by $\bigwedge_{a \in SORT} P(a)$

$\exists x$: SORT P(x) is replaced by $\bigvee_{a \in SORT} P(a)$.

13.1 Translation Into LEADSTO Format

The LEADSTO language (Bosse *et al.*, 2007) is an executable fragment of order-sorted logic. It models direct temporal or causal dependencies between two state properties in states at different points in time as follows. Let α and β be state properties of the form 'conjunction of atoms or negations of atoms', and e, f, g, h real or integer numbers (constants of sort VALUE). A LEADSTO expression $\alpha \twoheadrightarrow_{e, f, g, h} \beta$, holds for a trace γ if:

$\forall t1\ [\forall t\ [t1-g \le t < t1 \Rightarrow \alpha \text{ holds in } \gamma \text{ at time } t\ ] \Rightarrow \exists d\ [e \le d \le f\ \&\ \forall t'\ [t1+d \le t' < t1+d+h \Rightarrow \beta \text{ holds in } \gamma \text{ at time } t'\ ]$

The types of executable formulae introduced in Section 3 can easily be translated into the LEADSTO format as shown in Table 4.

Table 4. Translation of executable formulae into LEADSTO format

Executable formulae	Corresponding LEADSTO translation
$\forall t$ holds_at(X,γ,t) $\Rightarrow$ holds_at(Y, γ,t+c)	X $\twoheadrightarrow_{c\text{-}1,\, c\text{-}1,\, 1,\, 1}$ Y
$\forall t$ holds_at(X,γ,t) $\Rightarrow$ holds_at(X,γ,t+1)	X $\twoheadrightarrow_{0,\, 0,\, 1,\, 1}$ X
$\forall t$ holds_at(X,γ,t) $\Rightarrow$ holds_at(Y,γ,t)	X $\twoheadrightarrow_{-1,\, -1,\, 1,\, 1}$ Y

As an illustration, consider the following two examples of translation of properties from executable theories into the LEADSTO format.

1. The transition property from the a preparation to an action state

$\forall t'$ holds_at(preparation_for_output(t'+c, a), γ, t') $\Rightarrow$ holds_at(a, γ, t'+c)

is translated into:

preparation_for_output(y1, a) & combined_of(plus, y1, t', c) $\twoheadrightarrow_{c\text{-}1,\, c\text{-}1,\, 1,\, 1}$ a

2. The persistence property for the stimulus_reaction atom

$\forall t'$ [holds_at(stimulus_reaction(p) $\wedge \neg$ preparation_for_output(t'+c, a), γ, t')] $\Rightarrow$ holds_at(stimulus_reaction(p), γ, t'**+**1)

is translated into the LEADSTO expression:

stimulus_reaction(p) & $\neg$preparation_for_output(y1, a) & combined_of(**plus**, y1, t', c)]
$\twoheadrightarrow_{0,\, 0,\, 1,\, 1}$ stimulus_reaction(p)

A specification in LEADSTO format has as an advantage that it can be easily depicted graphically, in a causal graph or system dynamics style. Furthermore, based on specifications in LEADSTO format, using the dedicated software environment, simulations of different scenarios can be performed and predicate logical dynamic properties can be automatically checked with respect to the generated simulation traces.

13.2 Translation Into Propositional Modal Temporal Logic

Propositional modal temporal logic (Benthem, 1995; Fisher, 1996, 2005) has been extensively used in the area of computer science to formalize the temporal development of a system. This logic can be seen as an extension of classical propositional logic by temporal operators, for a linear discrete time frame (e.g., '○', meaning "at the next moment in time", '□' meaning "at every future moment", '◊' meaning "at some future moment"). The executable formulae of three types are translated into the propositional modal temporal logic as shown in Table 5.

Table 5. Translation of executable formulae into propositional modal temporal logic

Executable formulae	Corresponding propositional modal temporal logic translation
$\forall t$ holds_at(X,γ,t) $\Rightarrow$ holds_at(Y, γ,t+c)	$\Box\ (X \Rightarrow\ \circ_c\ Y)$*
$\forall t$ holds_at(X,γ,t) $\Rightarrow$ holds_at(X,γ,t+1)	$\Box\ (X \Rightarrow\ \circ\ X)$
$\forall t$ holds_at(X,γ,t) $\Rightarrow$ holds_at(Y,γ,t)	$\Box\ (X \Rightarrow Y)$

* $\circ_c$ is the contracted form that denotes c executable rules in form $X' \Rightarrow\ \circ\ Y'$, which describe c intermediate transitions between the state in which X holds and the state in which Y becomes true; notice that this requires that c-1 intermediate state properties are added to the state ontology to represent these intermediate states.

As a first example, the executable property

$\forall t'$ holds_at(preparation_for_output(t'+c, a), γ, t') $\Rightarrow$ holds_at(a, γ, t'+c)

with domain D_{ACTION} ={a1, a2} is translated into two propositional modal temporal logic formulae:

$\square$ (preparation_for_output(a1) $\Rightarrow \circ_c$ a1)

$\square$ (preparation_for_output(a2) $\Rightarrow \circ_c$ a2)

Note that some state properties contain variables (e.g., in memory functions) over sort LTIME, whereas in modal temporal logic time is not explicitly available. To allow translation of such properties into propositional modal temporal logic, the predicate present_time is added to the state ontology and the domain for sort LTIME is explicitly defined. For example, the memory state generation property

$\forall t'$ holds_at(p, γ, t') $\Rightarrow$ holds_at(memory(t', p) $\wedge$ stimulus_reaction(p), γ, t')

with the domains D_{LTIME} = {1,2,3}, D_{EVENT} = {p1, p2} is translated into a propositional modal temporal logic formula as follows:

present_time(1) & p1 $\Rightarrow$ memory(1, p1) $\wedge$ stimulus_reaction(p1)
present_time(2) & p1 $\Rightarrow$ memory(2, p1) $\wedge$ stimulus_reaction(p1)
present_time(3) & p1 $\Rightarrow$ memory(3, p1) $\wedge$ stimulus_reaction(p1)

Similar for the domain instance p2.

Although the obtained specification may look quite cumbersome, nevertheless, by applying automated verification techniques based on efficient temporal resolution methods, such specifications can be effectively processed and analyzed. Furthermore, executable properties translated into propositional modal temporal logic can be naturally represented in MetateM, a modelling language based on the direct execution of modal temporal logic statements. By means of the dedicated software tools simulation and analysis of MetateM specifications can be performed. However, the expressivity of propositional modal temporal logic is still limited. For practical purposes more compact and expressive representations are needed, such as, for example, suggested by first-order variants of temporal logic.

13.3 Translation into Monodic First-Order Temporal Logic

The first-order temporal logic (FOTL) is an extension of classical first-order logic with modal operators for a linear discrete time frame; e.g., (Hodkinson *et al.*, 2000; Hustadt *et al*, 2005). The monodic fragment of FOTL consists of all formulae, in which quantifiers over the domain variables are applied to formulae with at most one free temporal variable. The three types of formulae defined in the executable format comply with this requirement. In general, translation into this first order temporal logic is similar to one given in Table 5. Furthermore, since the monodic fragment does not include function symbols, the functions used for building state properties in the reified temporal predicate logic are to be replaced by the corresponding predicates, as shown earlier. Consider the following two examples.

1. The transition property from the a preparation to an action state

$\forall t'$ holds_at(preparation_for_output(t'+c, a), γ, t') $\Rightarrow$ holds_at(a, γ, t'+c)

is translated into

$\square$ [preparation_for_output(y1, a) & combined_of(**plus**, y1, t', c) $\Rightarrow$ $\circ_c$ a]

2. The persistence property for the stimulus_reaction atom

$\forall t'$ holds_at(stimulus_reaction(p) $\wedge \neg$ preparation_for_output(t'+c, a), γ, t') $\Rightarrow$
holds_at(stimulus_reaction(p), γ, t'+1)

is translated into

$\square$ [stimulus_reaction(p) & $\neg$preparation_for_output(y1, a) &
combined_of(**plus**, y1, t', c) $\Rightarrow$ $\circ$ stimulus_reaction(p)]

Specifications in monodic first-order temporal logic can be automatically verified using the dedicated theorem prover TeMP (Hustadt *et al*, 2005) that implements the resolution-based calculus for monodic first-order temporal logic.

13.4 Translation Into the Loosely Guarded Fragment of Predicate Logic

The loosely guarded fragment (Andreka, Benthem and Nemeti, 1998) is decidable and has good computational properties. Formulae in the loosely guarded fragment are specified in the form:

$$\exists y\ ((\alpha_1 \wedge \ldots \wedge \alpha_m) \wedge \psi(x, y)) \quad \text{or} \quad \forall y\ ((\alpha_1 \wedge \ldots \wedge \alpha_m) \rightarrow \psi(x, y))$$

where x and y are tuples of variables, $\alpha_1 \ldots \alpha_m$ are atoms that relativize a quantifier (the guard of the quantifier), and $\psi(x, y)$ is an inductively defined formula in the guarded fragment, such that each free variable of the formula is in the set of free variables of the guard. The formulae defined in the executable format are also formulae of the loosely guarded fragment of the first-order predicate logic with atomic guards specified by predicates holds_at(X, γ, t). For example, the transition property from the a preparation to an action state

$\forall$t' holds_at(preparation_for_output(t'+c, a), γ, t') $\Rightarrow$ holds_at(a, γ, t'+c)

is translated into the loosely guarded fragment as follows

$\forall$t' [[holds(y3, γ, t') & combined_of(preparation_for_output, y3, y1, a) &

combined_of(**plus**, y1, t', c)] $\rightarrow$ holds_at(a, γ, y1)]

Specifications in terms of the loosely guarded fragment can be effectively analyzed by resolution techniques implemented by theorem provers such as Bliksem (Nivelle, 1999).

14 Related Work

The proposed approach has similarities with compositional reasoning and verification techniques (Hooman, 1994; Jonker and Treur, 2002; de Roever *et al.*, 2001) in the way how it handles complex dynamics of a system. Compositional reasoning approaches developed in the area of software engineering are based on one common idea that the analysis of global properties of a software system can be reduced to the analysis of local properties of system components. More specifically, the problem of satisfaction of global properties of a complex software system can be reduced to two (easier) problems: (i) identifying and justifying relations between global properties of the system and local properties of its components (parts); (ii) verifying local properties of system components with respect to components specifications.

Recently different variations of the "assume-guarantee" method for compositional reasoning has gained popularity. According to this method, a system is divided into a (limited) number of components (processes), and then for each component, properties that describe (or prescribe) its externally observable behaviour are specified. A component property is expressed by a pair (A, G) consisting of a commitment (or a guarantee) G that the component will satisfy provided the environment of the component satisfies the assumption property A. Note that environmental assumptions for each component include also (results of) commitments of other components of the system, with which the component interacts. Furthermore, properties of components are expressed in terms of externally observable interface states of components; this allows abstraction from internal dynamics of components. For the specification of properties different types of (temporal) logics are used.

A specification of properties in the "assume-guarantee" method can be related to agent specifications considered in this paper. In particular, if a component is represented by an agent, then environmental assumptions for the component can be represented by input (observation) state properties of the agent, and the commitments can be represented by agent output (action, communication) state properties. Then, a component property corresponds to an implication relation between agent input and output state properties. Furthermore, both input and output state properties are expressed based on the interface ontology of the agent (i.e., abstracted from the internal dynamics of the agent and agent implementation details). Input state properties express observations and communications of/to environment, other agents, and even of/to the agent itself (which is different from the "assume-guarantee method").

Local properties of components in the "assume-guarantee" specifications are often verified on the internal specifications of components by algorithmic methods (e.g., model checking), however, (de)compositions relations between components are justified by compositional deductive technology (e.g., PVS theorem prover), in most cases semi-automatically. In the proposed approach the verification of relations between properties of different aggregation levels is performed completely automatically. In contrast to compositional reasoning methods, the proposed method provides means for automatic translation of the original specification of a complex system into the executable format that can be easily processed by model checking tools.

In (Pill et al., 2006) formal methods for the analysis of hardware specifications expressed in the language PSL (an extension of the standard temporal logics LTL and CTL), are described. By means of the suggested property assurance technique supported by a tool, different global system properties (e.g., consistency) can be verified on specifications and in such a way the correctness of specifications can be established. The verification is based on bounded model checking techniques. Besides the specification language, an essential difference between this analysis method and the approach proposed in this paper is that the latter provides means for the multi-level (or compositional) representation and verification of properties in specifications. This allows system modelling at a necessary level of abstraction and the reduction of the complexity of verification of system dynamics.

Similar differences can be identified in comparison with the approach proposed in (Fuxman, 2004). This approach allows semi-automatic formalization of informal graphical specifications of multi-agent systems with the subsequent verification of dynamic properties using model checking techniques. Formalized specifications comprise descriptions of classes that describe components of a multi-agent system and relations between them, constraints over these components, assertions and possibilities. Although the first-order temporal logic that is used for formalizing these specifications is expressive enough to define complex temporal relations, it is does not provide the complete expressivity allowed by the reified temporal logic used in this paper (e.g., arithmetical operations, references to multiple traces in the same formula). Furthermore, although such specifications can be built and analyzed in parts, the idea of compositional verification, central in our approach, is not elaborated in this approach.

Component structures and multi-aggregation-level representation of multi-agent systems are conceptually similar to organizational structures and organizational view on multi-agent systems, which also addresses the issue of handling complex dynamics; e.g., (Ferber and Gutknecht, 1998). However, organizational structures of multi-agent systems are defined often more strictly than groups of components in the proposed approach. In particular, in (Fisher et al., 2003; Ferber and Gutknecht, 1998) groups form a well-defined communication space for agents. Such a space ensures that messages are exchanged between and understood by the group members only. Furthermore, in (Ferber and Gutknecht, 1998) only agents that play a special role are allowed to transfer information in and outside a group. Such requirements are not always necessary in component-based systems, however may be ensured by specifying appropriate properties for components of different levels. Moreover, often a more fine-grained definition of roles as functions of agents is provided in multi-agent organizations (e.g., Jonker et al., 2007; Hannoun et al., 1998; Zambonelli et al., 2003). In a component-based system a way to divide different functions of a component and to treat them separately is by refining a component in subcomponents, which may be placed in different groups. Furthermore, sometimes norms can be enforced in multi-agent organizations at the level of groups, with which the group members should comply (Dignum et al., 2005). In the approach proposed in this paper the properties of higher level components are entailed logically from properties of lower level components. In such a way, group properties are not enforced in a top-down manner as norms, but rather emerge from the properties of the lower level components.

Representation formats similar to the one of an executable specification used in this paper are known in Computer Science and AI. In particular, representation of the dynamics of a system by logical implications from properties of the past to properties of the future is used in high-level modelling approaches developed in the area of Requirements Engineering (Kotonya and Sommerville, 1998). Such specifications often are used at a higher aggregation level for the more global properties of a process as a whole, abstracting from the basic steps or mechanisms that realise the process, but also can be used at lower aggregation levels.

In Dynamics Systems Theory (Van Gelder and Port, 1995) a dynamical system is often considered as a state-determined system, which dynamics is described by logical implications from properties of a present state to properties of future states. Whereas a specification 'from past to future' expresses a property of the process from a more global aggregation level, a specification 'from present to future' expresses properties of the same process at a lower, more basic aggregation level. This can be considered a refinement of the former specification. This illustrates Ashby (1960)'s assumption that the ontology for world states can be chosen or extended in such a manner that it is possible to obtain a specification in the more simple and more limited 'present to future' format which determines the behaviour of the system.

Recently, model checking techniques used for analysis in this paper have been applied more and more for verification of different aspects of multi-agent systems: for checking agent specifications in the form of programs (Bordini, 2004), for verifying communication protocols, and specification expressed in epistemic temporal logics (Wozna, Lomuscio and Penczek, 2005). In situations, in which dynamics of a multi-agent system are specified using temporal modal logics, temporal proof techniques can be of use. As an example, an approach for automated verification of temporal logical specifications in modal temporal logic by means of clausal resolution is suggested in (Fisher, 2005).

15 Discussion

The approach to analyzing behaviour of a multi-agent system proposed in this paper is based on distinguishing dynamic properties of different aggregation levels. The behaviour at a given aggregation level can be specified in some temporal logical language by a set of dynamic properties. As the behaviour of a system can be complex, specifications at higher aggregation levels in principle may involve complex temporal expressions. However, performing analysis (e.g., by simulations and determining logical consequences) of complex temporal formulae is not easy in general. Software tools to support analysis need system specifications in a simple format describing the system's basic steps at a lower aggregation level. For that reason, to make analysis possible, often specifications at a lower aggregation level have to be created, which may be a tedious task. For example, to express one complex temporal relation, usually a large number of simpler specifications are needed.

To support analysis on the basis of a higher level specification, an automated procedure has been developed, which allows transformation of a behavioural specification of a certain aggregation level into an executable temporal specification at a lower aggregation level, as a refinement of the given higher level specification. Specification of multi-agent system behaviour at a higher aggregation level is much easier. The reified temporal predicate logic provides an intuitive way of creating a specification of system dynamics, which by the proposed transformation process still can be automatically translated into a lower level specification, as shown here. To verify relations between the behavioural specifications of a lower and a higher aggregation levels model checking techniques are used. To enable model checking, an executable temporal specification of a lower aggregation level is translated subsequently into a description of a finite state transition system. Using this, the translation into the input format for the SMV model checker is performed. Furthermore, an executable specification in the format introduced in this paper can be easily translated into other existing executable languages and logics. This has as an advantage that analysis techniques and tools developed for other formalisms may be used for translated executable specifications. Furthermore, using the developed procedure also specifications of multi-agent systems in other existing executable languages and logics can be transformed into the input format of the

SMV model checker. For this no execution of the steps 1-4 of the procedure is needed, since the specification is already executable. Instead, the translation rules from Section 13 should be applied to obtain an executable specification in the format considered in this paper. Note that if the original specification is not executable, then to apply the procedure, a translation should be possible between the language of the specification and the reified predicate logic temporal language. In particular, in (Bosse et al., 2009) it has been shown how properties in the reified predicate logic temporal language can be related to temporal languages that are often used for verification, such as PTL and LTL. Then, the steps 5 and 6 are to be performed without any modification. The obtained SMV specification can be used to perform verification of interlevel relations as has been shown in this paper.

Finally, one other observation can be made. The transformation can also be seen and used as a way to eliminate past aspects from temporal formulae. It is sometimes a point of discussion in how far the possibility to incorporate references to the past adds expressivity to a temporal language; e.g., (Hodkinson and Reynolds, 2005). The transformation given here shows on the one hand that the past elements can be eliminated, so one can say that no essential expressivity is added by using past elements. On the other hand, however, this elimination is not for free: the state ontology has to be extended seriously to achieve it, in line with Ashby (1960)'s remarks discussed in Section 14.

REFERENCES

Andreka, H., Benthem, J. van, and Nemeti, I. (1998) Modal languages and bounded fragments of predicate logic. *Journal of Philosophical Logic* 27, pp. 217-274.

Arnold, A. (1994). *Finite transition systems*. Semantics of communicating systems. Prentice-Hall, 1994.

Ashby, R. (1952/1960). *Design for a Brain*. Chapman & Hall, London. First edition 1952, second edition 1960.

Benthem, J. van (1995). *Temporal Logic*. In: D. M. Gabbay, C. J. Hogger, and J. A. Robinson, Handbook of Logic in Artificial Intelligence and Logic Programming, Volume 4, Oxford: Clarendon Press, pp. 241-350.

Bordini, R.H., Fisher, M., Visser, W., Wooldridge, M. (2004). Verifiable multi-agent programs. In: Dastani, M., Dix, J., and El Fallah-Seghrouchni, A., eds., *Programming Multi-Agent Systems*, Proceedings of the First International Workshop (ProMAS-03). LNAI 3067, Springer-Verlag, Berlin, 72-89, (2004).

Bosse, T., Jonker, C. M., van der Meij, L., and Treur, J. (2007) A Language and Environment for Analysis of Dynamics by Simulation. *International Journal of Artificial Intelligence Tools*, vol. 16, 2007, pp. 435-464.

Bosse, T., Jonker, C.M., Meij, L. van der, Sharpanskykh, A., and Treur, J., Specification and Verification of Dynamics in Agent Models. International Journal of Cooperative Information Systems, 18 (1), 2009, pp 167-193.

Clarke, E.M., Grumberg, O., and Peled, D. (1999) A. *Model Checking*, MIT Press, Cambridge Massachusetts, London England, 1999.

Damasio, A. (2000). The Feeling of What Happens: Body, Emotion and the Making of Consciousness. MIT Press, 2000.

De Roever, W.-P., de Boer, F., Hannemann, U., Hooman, J., Lakhnech, Y., Poel, M., Zwiers, J. (2001). Concurrency Verification: Introduction to Compositional and Noncompositional Methods, Cambridge University Press, 2001.

Dignum, V., Vazquez-Salceda, J., Dignum, F. (2005). OMNI: Introducing Social Structure, Norms and Ontologies into Agent Organizations, In: Rafael H. Bordini, Mehdi Dastani, Jürgen Dix, et al. (eds.): Programming Multi-Agent Systems: Second International Workshop ProMAS 2004, New York, NY, July 20, 2004. Editors: LNAI 3346, Springer, 2005.

Dennett, D.C. (1991). *Consciousness Explained*, Penguin Press, 1991.

Ferber, J., Gutknecht, O. (1998). A Meta-Model for the Analysis and Design of Organizations in Multi-Agent Systems, In *Proceedings of the 3rd International Conference on Multi-Agent Systems (ICMAS)*, 128-135, (1998).

Fisher, M. (2005). Temporal Development Methods for Agent-Based Systems, *Journal of Autonomous Agents and Multi-Agent Systems*, vol. 10, 2005, pp. 41-66.

Fisher, M. (1996). A Temporal Semantics for Concurrent METATEM, *Journal of Symbolic Computation (Special Issue on Executable Temporal Logics)* 22(5):627-648, November/December 1996, Academic Press.

Fisher, M., Ghidini, C., Hirsch, B. (2003). Organising Computation through Dynamic Grouping. Objects, Agents, and Features, pp. 117-136.

Fitting, M. (1996). *First-order Logic and Automated Theorem Proving*, 2nd edition, Springer-Verlag, 1996.

Fuxman, A., Liu, L., Pistore, M., Roveri M., and Mylopoulos, J. (2004) Specifying and Analyzing Early Requirements in Tropos. In *Requirements Engineering Journal*, 9(2): 132-150, 2004.

Galton, A. (2003). Temporal Logic. *Stanford Encyclopedia of Philosophy*, URL: http://plato.stanford.edu/entries/logic-temporal/#2.

Galton, A. (2006). Operators vs Arguments: The Ins and Outs of Reification. *Synthese*, vol. 150, 2006, pp. 415-441.

Gelder, T.J. van, and Port, R.F., (1995). It's About Time: An Overview of the Dynamical Approach to Cognition. In: Port, R.F., Gelder, T. van (eds.) (1995). *Mind as Motion: Explorations in the Dynamics of Cognition*. MIT Press, Cambridge, Mass., pp. 1-43.

Hannoun, M., Sichman, J.S., Boissier, O., Sayettat., C.: Dependence Relations between Roles in a Multi-Agent System: Towards the Detection of Inconsistencies in Organization. In: Multi-Agent Systems and Agent-Based Simulation (1998)

Hodkinson, I., and Reynolds, M. (2005). Separation - Past, Present and Future. In: We Will Show Them: Essays in Honour of Dov Gabbay, Vol 2. S. Artemov, H. Barringer, A. S. d'Avila Garcez, L. C. Lamb, and J. Woods (eds.), College Publications, 2005, pp. 117-142.

Hodkinson, I., Wolter, F., and Zakharyaschev, M. (2000). Decidable fragments of first-order temporal logics. *Annals of Pure and Applied Logic* 106, pp. 85–134.

Hooman, J. (1994) Compositional Verification of a Distributed Real-Time Arbitration Protocol, *Real-Time Systems*, 6, 173-206, 1994.

Hustadt, U., Konev, B., Riazanov, A., and Voronkov, A. (2004). TeMP: A Temporal Monodic Prover. In: Basin, D. A., and Rusinowitch, M. (eds), *Proceedings of the Second International Joint Conference on Automated Reasoning IJCAR 2004*, LNAI 3097, Springer, pp. 326-330.

Jonker, C.M., and J. Treur. (2002). Compositional Verification of Multi-Agent Systems: a Formal Analysis of Pro-activeness and Reactiveness, *International Journal of Cooperative Information Systems*, vol. 11, 2002, 51-92.

Jonker C.M., Sharpanskykh, A., Treur, J., Yolum, P. (2007). A Framework for Formal Modeling and Analysis of Organizations, Applied Intelligence, 27(1), pp. 49-66

Kotonya, G. and Sommerville, I. (1998) *Requirements Engineering Processes and Techniques*, John Wiley, 1998.

Manzano, M. (1996). *Extensions of First Order Logic*, Cambridge University Press, 1996.

Marca, D.A. (1988). *SADT: Structured Analysis and Design Techniques*, McGraw-Hill, 1988.

McMillan, K. (1993) *Symbolic Model Checking*, Kluwer Academic Publishers, 1993.

Marcio Cysneiros, L., and E. Yu. (2002). Requirements Engineering for Large-Scale Multi-agent Systems. In: *Proceedings of the 1st International Central and Eastern European Conference on Multi-Agent Systems (CEEMAS*), 2002, pp. 39-56.

Nivelle, H. de. (1999). The Bliksem Theorem Prover, Version 1.12. Max-Planck-Institut, Saarbruecken, Germany, 1999. (*http://www.mpi-sb.mpg.de/~bliksem/manual.ps*)

Pill, I., Semprini, S., Cavada, R., Roveri, M., Bloem, R. and Cimatti, A. (2006) Formal Analysis Of Hardware Requirements. In Proceedings of the 43rd annual conference on Design automation (DAC '06), 2006.

Sharpanskykh, A., and Treur, J. (2005). Syntax and Semantics of the Temporal Trace Language, Technical Report No. TR-1801AI, Artificial Intelligence Department, Vrije Universiteit Amsterdam. *http://www.few.vu.nl/~sharp/tr1801ai.pdf*

Vardi, M.Y. (1996) An automata-theoretic approach to linear temporal logic. In: *Proceedings of the VIII Banff Higher Order Workshop*, in: Lecture Notes in Computer Science, vol. 1043, Springer-Verlag, 1996, pp. 238–266.

Wozna, B., Lomuscio, A., Penczek, W. (2005). Bounded Model Checking for Knowledge and Real Time. In: *Proceedings of the 4th International Joint Conference on Autonomous Agents and Multi Agent Systems (AAMAS'05)*, ACM Press, (2005).

Zambonelli, F., Jennings, N. R., Wooldridge, M. (2003). Developing multiagent systems: the Gaia Methodology. ACM Trans on Software Engineering and Methodology, Vol. 12 (3), pp. 317-370.

Appendix. http://www.few.vu.nl/~sharp/appendixes_interlevel_relations.pdf

Appendix A: Formal Foundations

Lemma 3.1

Let $\varphi(\gamma, t)$ be a dynamic property expressed using the state ontology Ont. Then the following holds:

(1) $\text{coincide\_on}(\gamma_1, \gamma_2, \text{Ont})\ \&\ \text{coincide\_on}(\gamma_2, \gamma_3, \text{Ont}) \Rightarrow \text{coincide\_on}(\gamma_1, \gamma_3, \text{Ont})$

(2) $\text{coincide\_on}(\gamma_1, \gamma_2, \text{Ont}) \Rightarrow [\ \varphi(\gamma_1, t) \Leftrightarrow \varphi(\gamma_2, t)\ ]$.

Proof sketch.

The transitivity property (1) follows directly from the definition of coinciding traces for $\text{coincide\_on}(\gamma_1, \gamma_2, \text{Ont})$ and $\text{coincide\_on}(\gamma_2, \gamma_3, \text{Ont})$:

$\forall a \in \text{STATATOM}_{\text{Ont}}\ \ \forall t'\ \ [\text{holds\_at}(a, \gamma_1, t') \Leftrightarrow \text{holds\_at}(a, \gamma_3, t')\ ] \Rightarrow \text{coincide\_on}(\gamma_1, \gamma_3, \text{Ont})$

From

$\forall\ \gamma_1, \gamma_2\ \ [\ \text{coincide\_on}(\gamma_1, \gamma_2, \text{Ont}) \Rightarrow \forall t'\ [\varphi_p(\gamma_1, t') \Leftrightarrow \varphi_p(\gamma_2, t')\ \ \&\ \ \varphi_f(\gamma_1, t') \Leftrightarrow \varphi_f(\gamma_2, t')\ ]]$

follows that $\varphi(\gamma_1, t) \Leftrightarrow \varphi(\gamma_2, t)$.

Theorem 3.1

If the executable specification $\pi_A(\gamma, t)$ refines the external behavioural specification $\varphi_A(\gamma, t)$ of component A, and $\psi(\gamma, t)$ is a dynamic interaction property of component A in its environment, expressed using the interaction ontology InteractionOnt(A), then for any trace γ

$$[\ \pi_A(\gamma, t) \Rightarrow \psi(\gamma, t)\ ] \Leftrightarrow [\ \varphi_A(\gamma, t) \Rightarrow \psi(\gamma, t)\ ]$$

Proof sketch.

$\Leftarrow$ is direct:

from $\pi_i(\gamma, t) \Rightarrow \varphi_i(\gamma, t)$ and $\wedge\varphi_i(\gamma, t) \Rightarrow \psi(\gamma, t)$ it follows $\wedge\pi_i(\gamma, t) \Rightarrow \psi(\gamma, t)$.

$\Rightarrow$ runs as follows:

Suppose $\varphi_i(\gamma, t)$ holds for all i, then since $\pi_1(\gamma)$ refines $\varphi_1(\gamma, t)$, then according to the definition of refinement of an externally observable property exists such a γ_1 that $\pi_1(\gamma_1)$ and $\text{coincide\_on}(\gamma, \gamma_1, \text{InteractionOnt (A)})$.

Due to Lemma 3.1, this γ_1 still satisfies all $\varphi_i(\gamma_1, t)$ (i.e., $\varphi_i(\gamma_1, t)$ holds for all i).

Proceed with γ_1 to obtain a γ_2 and further for all i to reach a trace γ_n, for which

$\pi_i(\gamma_n)$ holds for all i,

and

$\text{coincide\_on}(\gamma, \gamma_n, \text{InteractionOnt(A)})$,

and

$\varphi_i(\gamma_n)$ holds for all i.

From

for any trace $\gamma\ \forall i\ [\pi_i(\gamma) \Rightarrow \varphi_i(\gamma)]$,

and

for any trace $\gamma\ \ [\ \wedge\pi_i(\gamma) \Rightarrow \psi(\gamma, t)\ ]$

it follows that for any trace $\gamma\ \wedge\varphi_i(\gamma) \Rightarrow \psi(\gamma)$.

So it has been proven that for any trace $\gamma\ \wedge\varphi_i(\gamma) \Rightarrow \psi(\gamma)$. ■

Lemma 4.1 (Normalization lemma)

Let t be a given time point. If a formula $\delta(\gamma, t)$ only contains temporal relations such as $t' < t''$ and $t' \leq t''$, and atoms of the form holds_at(p, γ, t) for some name of a state formula p, then some state formula q(t) can be constructed such that $\delta(\gamma, t)$ is equivalent to the formula $\delta^*(\gamma, t)$ of the form holds_at(q(t), γ, t).

Proof sketch for Lemma 4.1.

First in the formula $\delta(\gamma, t)$ replace all temporal relations such as $t' < t''$ and $t' \leq t''$ by holds_at($t' < t''$, γ, t) and holds_at($t' \leq t''$, γ, t) respectively. Then proceed by induction on the composition of the formula $\delta(\gamma, t)$. Treat the logical connectives &, |, $\neg$, $\Rightarrow$, $\forall$s, $\exists$s.

1) conjunction: $\delta(\gamma, t)$ is $\delta 1(\gamma, t)$ & $\delta 2(\gamma, t)$

By induction hypothesis

$\delta 1(\gamma, t) \Leftrightarrow$ holds_at(p1, γ, t) (which is $\delta 1^*(\gamma, t)$)

$\delta 2(\gamma, t) \Leftrightarrow$ holds_at(p2, γ, t) (which is $\delta 2^*(\gamma, t)$)

Then

$\delta(\gamma, t) \Leftrightarrow$ holds_at(p1, γ, t) & holds_at(p2, γ, t) $\Leftrightarrow$ holds_at(p1 $\wedge$ p2 , γ, t) (which becomes $\delta^*(\gamma, t)$)

2) disjunction: $\delta(\gamma, t)$ is $\delta 1(\gamma, t)$ | $\delta 2(\gamma, t)$

Again by induction hypothesis

$\delta 1(\gamma, t) \Leftrightarrow$ holds_at(p1, γ, t) (which is $\delta 1^*(\gamma, t)$)

$\delta 2(\gamma, t) \Leftrightarrow$ holds_at(p2, γ, t) (which is $\delta 2^*(\gamma, t)$)

Then

$\delta(\gamma, t) \Leftrightarrow$ holds_at(p1, γ, t) | holds_at(p2, γ, t) $\Leftrightarrow$ holds_at(p1 $\vee$ p2, γ, t) (which becomes $\delta^*(\gamma, t)$)

3) negation: $\delta(\gamma, t)$ is $\neg\delta 1(\gamma, t)$

$\delta 1(\gamma, t) \Leftrightarrow$ holds_at(p1, γ, t)

$\delta(\gamma, t) \Leftrightarrow \neg$holds_at(p1, γ, t)

$\delta(\gamma, t) \Leftrightarrow$ holds_at(not(p1), , γ, t) (which is $\delta^*(\gamma, t)$)

4) implication: $\delta(\gamma, t)$ is $\delta 1(\gamma, t) \Rightarrow \delta 2(\gamma, t)$

Again by induction hypothesis

$\delta 1(\gamma, t) \Leftrightarrow$ holds_at(p1, γ, t) (which is $\delta 1^*(\gamma, t)$)

$\delta 2(\gamma, t) \Leftrightarrow$ holds_at(p2, γ, t) (which is $\delta 2^*(\gamma, t)$)

Then

$\delta(\gamma, t) \Leftrightarrow$ [holds_at(p1, γ, t) $\Rightarrow$ holds_at(p2, γ, t)] $\Leftrightarrow$ holds_at(p1 $\rightarrow$ p2, γ, t) (which becomes $\delta^*(\gamma, t)$)

5) universal quantifier:

$\delta(\gamma, t) \Leftrightarrow \forall t'$ holds_at(p1(t'), γ, t)

$\delta(\gamma, t) \Leftrightarrow$ holds_at($\forall u'$ p1(u'), γ, t) (which is $\delta^*(\gamma, t)$)

6) existential quantifier:

$\delta(\gamma, t) \Leftrightarrow \exists t'$ holds_at(p1(t') , γ, t)

$\delta(\gamma, t) \Leftrightarrow$ holds_at($\exists u'$ p1(u') , γ, t) (which becomes $\delta^*(\gamma, t)$)

Lemma 4.2

If time has properties of correctness and uniqueness, then

$\varphi_{mem}(\gamma, t) \Leftrightarrow$ holds_at(q_{mem}(t) , γ, t) $\Leftrightarrow$ holds_at(q_{mem}, γ, t) (1)

Proof.

The proof follows directly from Lemma 4.1, definitions of correctness and uniqueness of time and the definition of the formula q_{mem}.

Proposition 4.1

Let $\varphi_p(\gamma, t)$ be a past statement for a given t, $\varphi_{mem}(\gamma, t)$ the memory formula for $\varphi_p(\gamma, t)$, $q_{mem}(t)$ the normalized memory state formula for $\varphi_{mem}(\gamma, t)$, and $Th_{o\to m}$ the executable theory from the interaction states for $\varphi_p(\gamma, t)$ to the memory states. Then,

$Th_{o\to m} \models [\varphi_p(\gamma, t) \Leftrightarrow \varphi_{mem}(\gamma, t)]$

and

$Th_{o\to m} \models [\varphi_p(\gamma, t) \Leftrightarrow holds\_at(q_{mem}(t), \gamma, t)\ \&\ holds\_at(q_{mem}(t), \gamma, t) \Leftrightarrow holds\_at(q_{mem}, \gamma, t)]$.

Proof.

From the definitions of $q_{mem}(t)$ and of $Th_{o\to m}$ follows

$$Th_{o\to m} \models [\varphi_p(\gamma, t) \Leftrightarrow \varphi_{mem}(\gamma, t)]$$

Further by Lemma 4.2

$$Th_{o\to m} \models [\varphi_p(\gamma, t) \Leftrightarrow holds\_at(q_{mem}(t), \gamma, t)]\ \blacksquare$$

Lemma 5.1
If time has properties of correctness and uniqueness, then

$\varphi_{cmem}(\gamma, t, t_1) \Leftrightarrow holds\_at(q_{cond}(t, t_1), \gamma, t_1)\ \&\ holds\_at(q_{cond}(t, t_1), \gamma, t_1) \Leftrightarrow holds\_at(q_{cond}(t), \gamma, t_1)$ (2)

Proof.

The lemma can be proven in the same manner as Lemma 4.2.

Lemma 5.2
If time has properties of correctness and uniqueness, then

$\varphi_{prep}(\gamma, t_1) \Leftrightarrow holds\_at(q_{prep}(t_1), \gamma, t_1)\ \&\ holds\_at(q_{prep}(t_1), \gamma, t_1) \Leftrightarrow holds\_at(q_{prep}, \gamma, t_1)$ (3)

Proof.

The lemma can be proven in the same manner as Lemma 4.2.

Lemma 5.3
If time has properties of correctness and uniqueness, then

$\varphi_{cprep}(\gamma, t) \Leftrightarrow holds\_at(q_{cprep}(t), \gamma, t)\ \&\ holds\_at(q_{cprep}(t), \gamma, t) \Leftrightarrow holds\_at(q_{cprep}, \gamma, t)$ (4)

Proof.

The lemma can be proven in the same manner as Lemma 4.2.

Proposition 5.1
Let $\varphi_f(\gamma, t)$ be a future statement for t of the form $\forall t_1 > t\ [\varphi_{cond}(\gamma, t, t_1) \Rightarrow \varphi_{bh}(\gamma, t_1)]$, where $\varphi_{cond}(\gamma, t, t_1)$ is an interval statement, which describes a condition for one or more actions and/or communications and $\varphi_{bh}(\gamma, t_1)$ is a (conjunction of) future statement(s) for t_1, which describes action(s) and/or communications that are to be performed; let $\varphi_{prep}(\gamma, t_1)$ be the preparation formula, $\varphi_{cprep}(\gamma, t)$ be the conditional preparation formula for $\varphi_f(\gamma, t)$, $q_{cprep}(t)$ be the normalized conditional preparation state formula for $\varphi_{cprep}(\gamma, t)$, and $Th_{m\to p}$ the executable theory for $\varphi(\gamma, t)$ from memory states to preparation states. Then,

$$Th_{m\to p} \models \forall t_1 > t\ [\varphi_{cond}(\gamma, t, t_1) \Rightarrow \varphi_{prep}(\gamma, t_1)] \Leftrightarrow \varphi_{cprep}(\gamma, t)]$$

and

$$Th_{m\to p} \models [\forall t_1 > t\ [\varphi_{cond}(\gamma, t, t_1) \Rightarrow \varphi_{prep}(\gamma, t_1)] \Leftrightarrow holds\_at(q_{cprep}(t), \gamma, t)\ \&$$
$$holds\_at(q_{cprep}(t), \gamma, t) \Leftrightarrow holds\_at(q_{cprep}, \gamma, t)].$$

Proof.

From the definition of $Th_{m\to p}$, Lemmas 5.1 and 5.2, Definition 5.6 it follows that

$$Th_{m\to p} \models \forall t_1 > t\ [\varphi_{cond}(\gamma, t, t_1) \Rightarrow \varphi_{prep}(\gamma, t_1)] \Leftrightarrow \varphi_{cprep}(\gamma, t)]$$

Then, by Lemma 5.3

$$Th_{m\to p} \models [\forall t_1 > t\ [\varphi_{cond}(\gamma, t, t_1) \Rightarrow \varphi_{prep}(\gamma, t_1)] \Leftrightarrow holds\_at(q_{cprep}(t), \gamma, t)$$

and

$$\text{holds\_at}(q_{cprep}(t)\ ,\ \gamma,\ t) \Leftrightarrow \text{holds\_at}(q_{cprep}\ ,\ \gamma,\ t)$$

■

Proposition 6.1

Let $\varphi_{bh}(\gamma,\ t_1)$ be a (conjunction of) future statement(s) for t_1, which describes action(s) and/or communications that are to be performed, $\varphi_{prep}(\gamma,\ t_1)$ be the preparation formula and $Th_{p\to o}$ the executable theory from preparation states to output states. Then,

$$Th_{p\to o}\ \models\ [\ \varphi_{prep}(\gamma,\ t_1)\ \Rightarrow\ \varphi_{bh}(\gamma,\ t_1)\ ]$$

Proof.

Follows directly from the definition of $Th_{p\to o}$ ■

Figures.

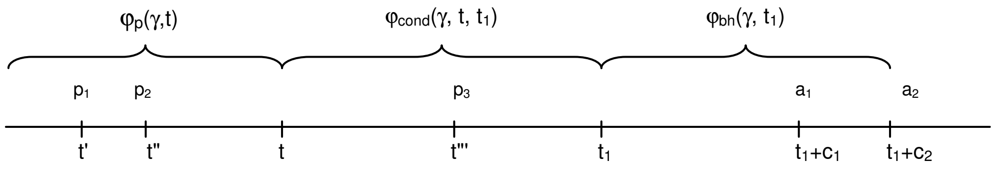

Figure 1. Graphical illustration of the structure of a formula from an external behavioural specification. In the illustration p_1, p_2 and p_3 represent state properties that hold at the time points t', t" and t"' correspondingly, and a1, a2 are the actions executed at time points t_1+c_1 and t_1+c_2 correspondingly.

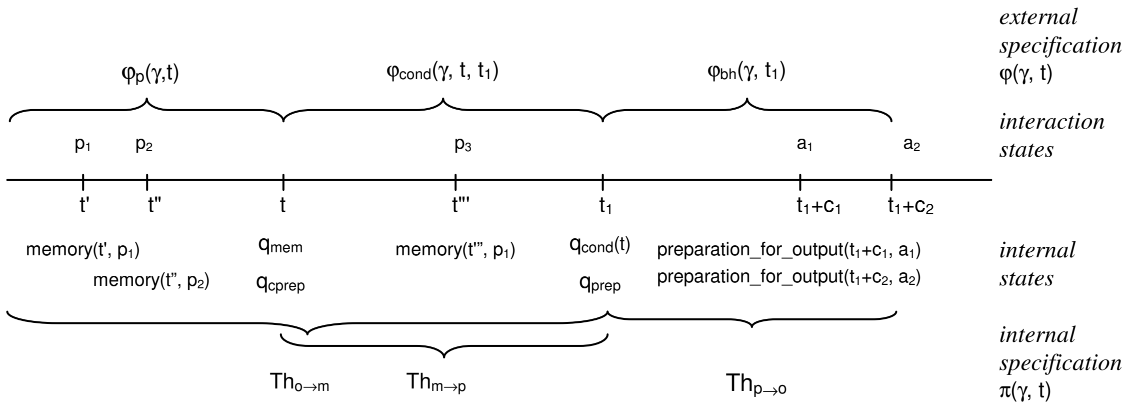

Figure 2. A graphical representation of relations between interaction states described by a non-executable dynamic property and internal states described by rules from the executable theories $Th_{o\to m}$, $Th_{m\to p}$ and $Th_{p\to o}$. Here p_1, p_2 and p_3 represent state properties that hold at time points t', t" and t"'; the memory states specified over these state properties are memory(t', p_1), memory(t', p_2) and memory(t', p_3); a_1 and a_2 are output states; preparation states for these output states are preparation_for_output(t1+c1, a1) and preparation_for_output(t1+c2, a2); q_{mem} and q_{cprep} are normalized memory and conditional preparation state formulae and $q_{cond}(t)$ and q_{cpret} are normalized condition state formula and preparation state formula used to specify transitions from memory states to preparation states for output.