Verifying Interlevel Relations within Organizational Models

Technical Report: TR-061909AI

Catholijn M. Jonker[1], Alexei Sharpanskykh[2],
Jan Treur[2], and pInar Yolum[3]

[1] NICI, Radboud University Nijmegen
Montessorilaan 3, 6525 HR Nijmegen, The Netherlands
C.Jonker@nici.ru.nl
[2] Department of Artificial Intelligence, Vrije Universiteit Amsterdam,
De Boelelaan 1081a, NL-1081 HV Amsterdam, The Netherlands
{sharp,treur}@few.vu.nl
[3] Department of Computer Engineering, Bogazici University,
TR-34342 Bebek, Istanbul, Turkey
pyolum@cmpe.boun.edu.tr

In this paper the formal theoretical basis used for transformation of a non-executable external behavioral specification for an organizational model into an executable format, required for enabling verification techniques, is explained in detail.

An external behavioral specification for components of the organizational model is specified using the Temporal Trace Language (TTL), which syntax and semantics are explained in Section 1.

In the general case, at any aggregation level a behavioral specification for an organizational model component consists of dynamic properties expressed by complex temporal relations in TTL, which therefore does not allow direct application of automatic verification procedures, more specifically, model checking techniques, used in this paper. In order to apply model checking techniques it is needed to transform an original behavioral specification of a certain aggregation level into a model based on a finite state transition system. In order to obtain this, as a first step a behavioral description for the lower aggregation level is replaced by one in executable (temporal) format. As a solution, an automated substitution of the behavioral specification for the component by an executable specification (expressed in an executable temporal language) is put forward. The justification is based on the theorem that a behavioral specification entails a certain dynamic property if and only if the generated executable specification entails the same property. The proof for this theorem and other formal theoretical results are given in Section 2.

Moreover, for the purposes of practical verification by means of model checking techniques, an automated translation from a behavioral specification based on executable temporal logical properties into a finite state transition system description has been developed. The details of a translation procedure are explained in Section 3.

Furthermore, the procedure for translating from the state transition system description into the model specification format for the SMV model checker that is used for verification is described in Section 4.

In Sections 5 the application of the proposed approach is illustrated by a paradigmatic example.

1. TTL Syntax and Semantics

The language TTL, short for Temporal Trace Language, is a variant of order-sorted predicate logic. Whereas the standard multi-sorted predicate logic is a language to reason about static properties only, TTL is an extension of such languages with facilities for reasoning about the dynamic properties of arbitrary systems expressed by static languages.

1.1 The State Language

For expressing state properties of a system ontologies are used. In logical terms, an ontology is a signature that specifies the vocabulary of a language to represent and reason about a system. In order to represent different ontological entities of a system a number of different syntactical sorts are used. Thus, the state properties are specified using a multi-sorted first-order predicate language L_{STATE} with the vocabulary, specified by a signature:

Definition 1.1 (State Language Signature)

A signature ℓ_{SL} is a tuple, ℓ_{SL}=<***S***; ***C***; ***P***; ***f***>, where ***S*** is a set of sort names; ***C*** is a set of constants of each sort; ***P*** is a set of predicate symbols; ***f*** is a set of functional symbols.

Let ℓ_{SL}=<***S***; ***C***; ***P***; ***f***> be a signature for L_{STATE}. We assume that the set of variables is given and that with each variable x from this set a sort S is associated, written as x:S. Then the terms and formulae of the language L_{STATE} are defined as follows.

Definition 1.2 (Terms of the state language)

The terms of any sort S are inductively defined by:

1. If x:S is a variable of the state language, then x is a term of L_{STATE}.
2. If $c \in \boldsymbol{C}$ is a constant symbol, then c is a term of L_{STATE}.
3. If $f \in \boldsymbol{f}$ is an n-place function symbol $S_1 \times \ldots \times S_n \rightarrow S$ and $\tau_1,\ldots, \tau_n$ are terms of L_{STATE}, such that $\tau_i \in S_i^{TERMS}$ (a set of all terms constructed using the sort S_i), then $f(\tau_1,\ldots, \tau_n) \in S^{TERMS}$ is a term of $L_{STATE.}$

Definition 1.3 (Formulae of the state language)

The formulae of L_{STATE} are inductively defined as follows:

1. If P: $S_1 \times S_2 \times \ldots \times S_n$ is a predicate symbol from $\boldsymbol{P}$ and $\tau_1, \tau_2\ldots, \tau_n$ are terms of L_{STATE}, such that $\tau_i \in S_i^{TERMS}$, then $P(\tau_1, \tau_2,\ldots, \tau_n)$ is a formula of $L_{STATE.}$
2. If τ_1 and τ_2 are terms of the same sort S, then $\tau_1 = \tau_2$ is a formula of $L_{STATE.}$
3. If φ and ψ are the formulae of L_S, then $\sim\varphi$, $(\varphi|\psi)$, $(\varphi\&\psi)$, $(\varphi\Rightarrow\psi)$ and $(\varphi\Leftrightarrow\psi)$ are formulae of L_{STATE}.
4. If φ a formula of L_S containing x as a free variable, then $(\forall x\ \varphi)$ and $(\exists x\ \varphi)$ are formulae of L_{STATE}.

1.2. The Language of TTL

In the language TTL formulae of the state language L_{STATE} are used as objects. To provide names of object language formulae φ in the TTL the operator (*) is used (written as φ*). Then, by means of the defined in the TTL *holds* predicate these objects are evaluated in states defined within TTL. The language TTL (L_{TTL}) has a signature ℓ_{TTL} that facilitates the specification of and reasoning about the dynamics of systems:

Definition 1.4 (TTL Signature)

A signature consists of the symbols of the following classes:

(1) a number of sorts: TIME (a set of all time points), STATE (a set of all state names), TRACE (a set of all trace names; a trace can be considered as a timeline with for each time point a state), STATPROP (a set of all names for state properties expressed using the state language); and VALUE (an ordered set of numbers). Furthermore, for every sort S from L_{STATE} three TTL sorts exist: the sort S^{VARS}, which contains all variable names of sort S, and the sort S^{GTERMS}, which contains names of all ground terms, constructed using sort S. Sorts S^{GTERMS} and S^{VARS} are subsorts of a sort S^{TERMS}.

(2) countably infinite number of individual variables of each sort. We shall use t with subscripts and superscripts for variables of the sort TIME; γ with subscripts and superscripts for variables of the sort TRACE; s with subscripts and superscripts for variables of the sort STATE; v with subscripts and superscripts for variables of the sort VALUE.

(3) a set of function symbols $\boldsymbol{\Phi}$, among which:
   a) a function symbol *state* of type TRACE x TIME $\rightarrow$ STATE.
   b) functional symbols $\wedge, \vee, \rightarrow, \leftrightarrow$: STATPROP x STATPROP $\rightarrow$ STATPROP; not: STATPROP $\rightarrow$ STATPROP.
   c) functional symbols $\forall$: S^{VARS} x STATPROP $\rightarrow$ STATPROP, and $\exists$: S^{VARS} x STATPROP $\rightarrow$ STATPROP for every sort S.
   d) functional symbols –, +, /, •: TIME x VALUE $\rightarrow$ TIME.
   e) functional symbols –, +, /, •: VALUE x VALUE $\rightarrow$ VALUE.

(4) a set of predicate symbols **P,** among which:
   a) a predicate symbol *holds* (|=) of type STATE x STATPROP.
   b) =: an identity relation on arbitrary sorts
   c) <: TIME x TIME is the earlier than relation on time
   d) <: VALUE x VALUE is the less than relation on the sort VALUE

When it is necessary to indicate an aspect state of a system component (i.e., input, output or internal), the sorts ASPECT_COMPONENT (a set of the component aspects of a system); COMPONENT (a set of all component names of a system); and COMPONENT_STATE_ASPECT (a set of all names of aspects of all component states) are included in the TTL. Using these sorts a functional symbol *comp_aspect* can be defined as: ASPECT_COMPONENT x COMPONENT $\rightarrow$ COMPONENT_STATE_ASPECT. Then, a function *state* is specified as: TRACE x TIME x COMPONENT_STATE_ASPECT $\rightarrow$ STATE.

Notice that also within states statements about time can be made, for this purposes the sort LTIME is used in the state language. Further we shall use u with subscripts and superscripts to denote constants of sort LTIME^{VARS}.

State language formulae are incorporated into the TTL by mappings of variable sets, terms sets and formulae sets into the names of sorts S^{GTERMS}, S^{TERMS}, S^{VARS} and STATPROP using the operator (*).

Definition 1.5 (Operator *)

The operator (*) is defined inductively on the structure of formulae from L_{STATE} by the following mappings:

1. Each constant symbol $c \in S$ in $\boldsymbol{C}$ is mapped to the constant name c' of sort S^{GTERMS}.
2. Each variable x:S of the state language is mapped to the constant name $x' \in S^{VARS}$

3. Each function symbol f: $S_1 \times S_2 \times \ldots \times S_n \rightarrow S_{n+1}$ in ℓ_{SL} is mapped to the function name f': $S_1^{TERMS} \times S_2^{TERMS} \times \ldots \times S_n^{TERMS} \rightarrow S_{n+1}^{TERMS}$
4. Each predicate symbol P: $S_1 \times S_2 \times \ldots \times S_n$ is mapped to the function name P': $S_1^{TERMS} \times S_2^{TERMS} \times \ldots \times S_n^{TERMS} \rightarrow$ STATPROP
5. The mappings for state formulae are defined as follows:
    a. $(\sim\varphi)^* = not(\varphi^*)$
    b. $(\varphi \,\&\, \psi)^* = \varphi^* \wedge \psi^*$
    c. $(\varphi \mid \psi)^* = \varphi^* \vee \psi^*$
    d. $(\varphi \Rightarrow \psi)^* = \varphi^* \rightarrow \psi^*$
    e. $(\varphi \Leftrightarrow \psi)^* = \varphi^* \leftrightarrow \psi^*$
    f. $(\forall x\, \varphi(x))^* = \forall x'\, \varphi^*(x')$, where x' is any constant of S^{VARS}

Make notice that the sorts S^{GTERMS} and S^{VARS} contain only the elements, corresponding to mappings in Definition 1.5.

Furthermore, it is assumed that the state language and the TTL signature define disjoint sets of expressions. Therefore, further in TTL formulae we will use the same notations for the elements of the object language (i.e, constants, variables, functions, predicates) and for their names in the TTL without introducing any ambiguity.

Definition 1.6 (TTL Terms)

a. Any variable x of TTL sort S is a term of L_{TTL} of sort S.
b. If f is an n-place function symbol of the TTL language $S_1 \times \ldots \times S_n \rightarrow S$ and $\tau_1,\ldots, \tau_n$ are terms of TTL sorts $S_1,\ldots., S_n$, then $f(\tau_1,\ldots, \tau_n)$ is a term of L_{TTL} of sort S.

Definition 1.7 (TTL Formulae)

TTL- formulae are defined inductively as follows:

A. The set of **atomic TTL-formulae** 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(v_1, u_1) is an atomic TTL formula (sometimes is used in infix notation like $v_1 \models u_1$).

(2) if τ_1, τ_2 are terms of any sort, then $=(\tau_1, \tau_2)$ is an atomic TTL formula. (further we shall use this predicate in infix form $\tau_1 = \tau_2$)

(3) if t_1, t_2 are terms of sort TIME, then $<(t_1, t_2)$ is an atomic TTL formula. (further we shall use this predicate in form $t_1 < t_2$, furthermore we shall use $t_1 \le t_2$ for $t_1<t_2 \wedge t_1=t_2$)

(4) If v1, v2 are terms of sort VALUE, then v1 < v2 is an atomic TTL formula.

B. The set of **well-formed TTL-formulae** is defined as

(1) Any atomic TTL-formula is a well-formed TTL-formula

(2) If φ and ψ are well-formed TTL-formulae, then so are $\sim\varphi$, $(\varphi|\psi)$, $(\varphi\&\psi)$, $(\varphi\Rightarrow\psi)$ and $(\varphi\Leftrightarrow\psi)$.

(3) If φ is a well-formed TTL-formula containing TTL variable x of sort S, where S is one of TTL sorts, then $(\forall x\, \varphi)$ and $(\exists x\, \varphi)$ are well-formed TTL-formulae.

1.2. The Semantics of TTL

An interpretation of a TTL formula is defined by the standard interpretation of order sorted predicate logic formulae.

Definition 1.8 (Interpretation)

An **interpretation** of a TTL formula is defined by a mapping *I* that:

(1) associates each sort symbol S to a certain set (subdomain) D_S, and if $S \subseteq S'$ then $D_S \subseteq Ds'$;

(2) associates each constant c of sort S to some element of D_S

(3) associates each function symbol f of sort $<X_1, \ldots, X_i> \rightarrow X_{i+1}$ to a mapping $I(X_1) \times \ldots \times I(X_i) \rightarrow I(X_{i+1})$

(4) associates each predicate symbol P of sort $<X_1, \ldots, X_i>$ to a relation on $I(X_1) \times \ldots \times I(X_i)$

Definition 1.9 (TTL Model)

A **model M** for the language TTL is a pair M=<I, V>, where:

- *I* is an interpretation function, and

- V is a variable assignment function, mapping each variable x: S to an element of D_S.

We write V [x/v] for the assignment function that maps variables y other than x to V(y) and maps x to v. Analogously, we write M[x/v] = <I, V [x/v]>.

Definition 1.10 (Interpretation of TTL terms)

Let M=<I,V> be a model for TTL. Then the meaning of a term $\tau \in$ TTL, denoted by τ^M, is inductively defined by:

1. $(x)^M = V(x)$, where x is a variable over one of the TTL sorts.
2. $(c)^M = I(c)$, where c is a constant of one of the TTL sorts.
3. $f(\tau_1,\ldots,\tau_k)^M = I(f)(\tau_1^M,\ldots, \tau_k^M)$, where f is a TTL function of type $S_1 \times \ldots \times S_n \rightarrow S$ and $\tau_1,\ldots, \tau_n$ are terms of TTL sorts $S_1,\ldots., S_n$

Definition 1.11 (Truth definition for TTL)

Let M=<I, V> be a model for TTL. Then the truth definition of TTL is inductively defined by:

1. $\models_M P(\tau_1,\ldots,\tau_k)$ iff $I(P) (\tau_1^M,\ldots, \tau_k^M) = \mathit{true}$
2. $\models_M \neg\varphi$ iff $\not\models_M \varphi$
3. $\models_M \varphi \wedge \psi$ iff $\models_M \varphi$ and iff $\models_M \psi$
4. $\models_M \forall x (\varphi(x))$ iff $\models_{M[x/v]} \varphi(x)$ for all $v \in D_S$, where x is a variable of sort S.

The semantics of connectives and quantifiers is defined in the standard way.

1.3 Axioms of TTL

(1) Equality of traces:
$\forall\gamma_1, \gamma_2 [\forall t [state(\gamma_1, t) = state(\gamma_2, t)] \Rightarrow \gamma_1 = \gamma_2]$

(2) Equality of states:
$\forall s_1, s_2 [\forall a{:}STATPROP [truth\_value(s_1, a) = truth\_value(s_2, a)] \Rightarrow s_1 = s_2]$

(3) Truth value in a state:
$holds(s, p) \Leftrightarrow truth\_value(s, p) = \mathit{true}$

(4) State consistency axiom:
$\forall \gamma, t, p (holds(state(\gamma, t), p) \Rightarrow \neg holds(state(\gamma, t), not(p)))$

(5) State property semantics:
   a. $holds(s, (p_1 \wedge p_2)) \Leftrightarrow holds(s, p_1)$ & $holds(s, p_2)$
   b. $holds(s, (p_1 \vee p_2)) \Leftrightarrow holds(s, p_1)$ | $holds(s, p_2)$
   c. $holds(s, not(p_1)) \Leftrightarrow \neg holds(s, p_1)$

For any constant variable name x from the sort S^{VARS}:

   d. $holds(s, (\exists(x, F))) \Leftrightarrow \exists x'{:}S^{GTERMS}$ holds(s, G), with G, F terms of sort STATPROP, where G is obtained from F by substituting all occurrences of x by x'
   e. $holds(s, (\forall(x, F))) \Leftrightarrow \forall x'{:}\ S^{GTERMS}$ holds(s, G), with G, F terms of sort STATPROP, where G is obtained from F by substituting all occurrences of x by x'

(6) Partial order axioms for the sort TIME:
   a. $\forall t\ t \leq t$ (Reflexivity)
   b. $\forall t_1,t_2 [t_1 \leq t_2 \wedge t_2 \leq t_1] \Rightarrow t_1 = t_2$ (Anti-Symmetry)
   c. $\forall t_1, t_2, t_3 [t_1 \leq t_2 \wedge t_2 \leq t_3] \Rightarrow t_1 \leq t_3$ (Transitivity)

(7) Axioms for the sort VALUE:
   a. $\forall v\ v \leq v$ (Reflexivity)
   b. $\forall v_1,v_2 [v_1 \leq v_2 \wedge v_2 \leq v_1] \Rightarrow v_1 = v_2$ (Anti-Symmetry)
   c. $\forall v_1, v_2, v_3 [v_1 \leq v_2 \wedge v_2 \leq v_3] \Rightarrow v_1 \leq v_3$ (Transitivity)
   d. Standard arithmetic axioms

(8) Axioms, which relate the sorts TIME and VALUE:
   a. $(t + v_1) + v_2 = t + (v_1 + v_2)$
   b. $(t \bullet v_1) \bullet v_2 = t \bullet (v_1 \bullet v_2)$

(9) Finite variability property (optional):

$\forall\gamma\ \forall t\ \exists\delta{>}0\ \exists t_1, t_2\ \ t_1 \le t \le t_2\ \&\ t_2 - t_1 \ge \delta\ \&\ \forall t'\ t_1 \le t' \le t_2\ \text{state}(\gamma, t') = \text{state}(\gamma, t)$

2. FORMAL JUSTIFICATION FOR THE TRANSFORMATION PROCEDURE

Lemma 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 $t' \le t''$, and atoms of the form $\text{state}(\gamma, t) \models p$ 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 $\text{state}(\gamma, t) \models q(t)$.

Proof sketch for Lemma 1.

First in the formula $\delta(\gamma, t)$ replace all temporal relations such as $t' < t''$ and $t' \le t''$ by $\text{state}(\gamma, t) \models t' < t''$ and $\text{state}(\gamma, t) \models t' \le t''$ respectively. Then proceed by induction on the composition of the formula $\delta(\gamma, t)$. Treat the logical connectives &, |, ¬, ⇒, ∀s, ∃s.

1) conjunction: $\delta(\gamma, t)$ is $\delta1(\gamma, t)\ \&\ \delta2(\gamma, t)$

By induction hypothesis

$\delta1(\gamma, t) \Leftrightarrow \text{state}(\gamma, t) \models p1$ (which is $\delta1^*(\gamma, t)$)

$\delta2(\gamma, t) \Leftrightarrow \text{state}(\gamma, t) \models p2$ (which is $\delta2^*(\gamma, t)$)

Then

$\delta(\gamma, t) \Leftrightarrow \text{state}(\gamma, t) \models p1\ \&\ \text{state}(\gamma, t) \models p2 \Leftrightarrow \text{state}(\gamma, t) \models [\, p1 \wedge p2\,]$ (which becomes $\delta^*(\gamma, t)$)

2) disjunction: $\delta(\gamma, t)$ is $\delta1(\gamma, t)\ |\ \delta2(\gamma, t)$

Again by induction hypothesis

$\delta1(\gamma, t) \Leftrightarrow \text{state}(\gamma, t) \models p1$ (which is $\delta1^*(\gamma, t)$)

$\delta2(\gamma, t) \Leftrightarrow \text{state}(\gamma, t) \models p2$ (which is $\delta2^*(\gamma, t)$)

Then

$\delta(\gamma, t) \Leftrightarrow \text{state}(\gamma, t) \models p1\ |\ \text{state}(\gamma, t) \models p2 \Leftrightarrow \text{state}(\gamma, t) \models [\, p1 \vee p2\,]$ (which becomes $\delta^*(\gamma, t)$)

3) negation: $\delta(\gamma, t)$ is $\neg\delta1(\gamma, t)$

$\delta1(\gamma, t) \Leftrightarrow \text{state}(\gamma, t) \models p1$

$\delta(\gamma, t) \Leftrightarrow \neg\text{state}(\gamma, t) \models p1$

$\delta(\gamma, t) \Leftrightarrow \text{state}(\gamma, t) \models \text{not}(p1)$ (which is $\delta^*(\gamma, t)$)

4) implication: $\delta(\gamma, t)$ is $\delta1(\gamma, t) \Rightarrow \delta2(\gamma, t)$

Again by induction hypothesis

$\delta1(\gamma, t) \Leftrightarrow \text{state}(\gamma, t) \models p1$ (which is $\delta1^*(\gamma, t)$)

$\delta2(\gamma, t) \Leftrightarrow \text{state}(\gamma, t) \models p2$ (which is $\delta2^*(\gamma, t)$)

Then

$\delta(\gamma, t) \Leftrightarrow [\text{state}(\gamma, t) \models p1 \Rightarrow \text{state}(\gamma, t) \models p2] \Leftrightarrow \text{state}(\gamma, t) \models [p1 \rightarrow p2]$ (which becomes $\delta^*(\gamma, t)$)

5) universal quantifier:

$\delta(\gamma, t) \Leftrightarrow \forall t'\ \text{state}(\gamma, t) \models p1(t')$

$\delta(\gamma, t) \Leftrightarrow \text{state}(\gamma, t) \models \forall u'\ p1(u')$ (which is $\delta^*(\gamma, t)$)

6) existential quantifier:

$\delta(\gamma, t) \Leftrightarrow \exists t'\ \text{state}(\gamma, t) \models p1(t')$

$\delta(\gamma, t) \Leftrightarrow \text{state}(\gamma, t) \models \exists u'\ p1(u')$ (which becomes $\delta^*(\gamma, t)$)

Definition 2.1 (Uniqueness and correctness of time)

To relate time within a state property to time external to states a functional symbol present_time: $\text{LTIME}^{\text{TERMS}} \rightarrow \text{STATPROP}$ 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:

$\forall t, t''\ \text{state}(\gamma, t) \models \text{present\_time}(t'') \Rightarrow \forall t', t' \neq t''\ \neg\text{state}(\gamma, t) \models \text{present\_time}(t')$

Correctness of time

This expresses that present_time(t) is true for the current time point t:

$\forall t\ state(\gamma, t) \models present\_time(t)$

Definition 2.2 (Memory formula)
The formula $\varphi_{mem}(\gamma, t)$ obtained by replacing all occurrences in $\varphi_p(\gamma, t)$ of subformulae of the form $state(\gamma, t') \models p$ by $state(\gamma, t) \models memory(t', p)$ is called the *memory formula for* $\varphi_p(\gamma, t)$.

Definition 2.3 (Normalized memory state formula)
The state formula constructed by Lemma 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 $\forall u' [present\_time(u') \rightarrow q_{mem}(u')]$.

Lemma 2
If time has properties of correctness and uniqueness, then

$$\varphi_{mem}(\gamma, t) \Leftrightarrow state(\gamma, t) \models q_{mem}(t) \Leftrightarrow state(\gamma, t) \models q_{mem} \quad (1)$$

Proof.

The proof for Lemma 2 follows directly from the Lemma 1, definitions of correctness and uniqueness of time and the definition of the formula q_{mem}. Lemmas 3, 4 and 5 can be proven in the same manner.

Definition 2.4 (Executable theory from interaction to memory)
For a given $\varphi(\gamma, t)$ the executable theory from observation states to memory states $Th_{o\rightarrow m}$ consists of the formulae:
For any atom p occurring in $\varphi_p(\gamma, t)$, expressed in the InteractionOnt(A) for a component A:

$\forall t'\ state(\gamma, t') \models p \Rightarrow state(\gamma, t') \models memory(t', p)$,

$\forall t''\ state(\gamma, t'') \models memory(t', p) \Rightarrow state(\gamma, t''+1) \models memory(t', p)$,

$state(\gamma, 0) \models present\_time(0)$,

$\forall t\ state(\gamma, t) \models present\_time(t) \Rightarrow state(\gamma, t+1) \models present\_time(t+1)$,

The last two rules are assumed to be included into two following theories $Th_{m\rightarrow p}$ and $Th_{p\rightarrow o}$ as well.

Proposition 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 $Th_{o\rightarrow m}$ the executable theory from the interaction states for $\varphi_p(\gamma, t)$ to the memory states. Then,

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

and

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

Proof.

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

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

Further by Lemma 2

$Th_{o\rightarrow m} \models [\varphi_p(\gamma, t) \Leftrightarrow state(\gamma, t) \models q_{mem}(t)]$ ■

Definition 2.5 (Normalized condition state formula)
The state formula constructed by Lemma 1 for $\varphi_{cmem}(\gamma, t, t_1)$ 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 3
If time has properties of correctness and uniqueness, then

$$\varphi_{cmem}(\gamma, t, t_1) \Leftrightarrow state(\gamma, t_1) \models q_{cond}(t, t_1) \ \&\ state(\gamma, t_1) \models q_{cond}(t, t_1) \Leftrightarrow state(\gamma, t_1) \models q_{cond}(t) \quad (2)$$

Proof.

The lemma can be proven in the same manner as Lemma 2.

Definition 2.6 (Normalized preparation state formula)
The state formula constructed by Lemma 1 for $\varphi_{prep}(\gamma, t_1)$ 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 $\forall u' [present\_time(u')] \rightarrow q_{prep}(u')]$

Lemma 4

If time has properties of correctness and uniqueness, then

$\varphi_{prep}(\gamma, t_1) \Leftrightarrow state(\gamma, t_1) \models q_{prep}(t_1)$ & $state(\gamma, t_1) \models q_{prep}(t_1) \Leftrightarrow state(\gamma, t_1) \models q_{prep}$ (3)

Proof.

The lemma can be proven in the same manner as Lemma 2.

Definition 2.7 (Conditional preparation formula and normalized conditional preparation state 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 $state(\gamma, t) \models \forall u_1 > t\ [q_{cond}(t, u_1) \rightarrow q_{prep}(u_1)]$ is called the *conditional preparation formula* for $\varphi_f(\gamma, t)$.

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

Lemma 5
If time has properties of correctness and uniqueness, then

$\varphi_{cprep}(\gamma, t) \Leftrightarrow state(\gamma, t) \models q_{cprep}(t)$ & $state(\gamma, t) \models q_{cprep}(t) \Leftrightarrow state(\gamma, t) \models q_{cprep}$ (4)

Proof.

The lemma can be proven in the same manner as Lemma 2.

Definition 2.8 (Executable theory from memory to preparation)
For any state atom p occurring in $\varphi_{cond}(\gamma, t, t_1)$, expressed in the InteractionOnt(A) for component A[1]:

$\forall t'\ state(\gamma, t') \models p \Rightarrow state(\gamma, t') \models [\ memory(t', p) \wedge stimulus\_reaction(p)\ ]$

$\forall t'', t'\ state(\gamma, t'') \models memory(t', p) \Rightarrow state(\gamma, t''+1) \models memory(t', p)$

$\forall t'\ state(\gamma, t') \models q_{mem} \Rightarrow state(\gamma, t') \models q_{cprep}$,

$\forall t', t\ state(\gamma, t') \models [q_{cprep} \wedge q_{cond}(t) \wedge \bigcap_{p} stimulus\_reaction(p)\ ] \Rightarrow state(\gamma, t') \models q_{prep}$,

$\forall t'\ state(\gamma, t') \models [\ stimulus\_reaction(p) \wedge \neg\ preparation\_for(output(t'+c, a))\ ] \Rightarrow state(\gamma, t'+1) \models stimulus\_reaction(p)$,

$\forall t'\ state(\gamma, t') \models [\ preparation\_for(output(t'+c, a)) \wedge \neg\ output(a)\ ] \Rightarrow state(\gamma, t'+c) \models preparation\_for(output(t'+c, a))$,

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

Proposition 2
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 $\varphi_{cprep}(\gamma, t)$ 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,

$Th_{m \rightarrow p} \models [\varphi_f(\gamma, t) \Leftrightarrow \varphi_{cprep}(\gamma, t)]$

and

$Th_{m \rightarrow p} \models [\ \varphi_f(\gamma, t) \Leftrightarrow state(\gamma, t) \models q_{cprep}(t)$ & $state(\gamma, t) \models q_{cprep}(t) \Leftrightarrow state(\gamma, t) \models q_{cprep}]$.

Proof.

From the definition of $Th_{m \rightarrow p}$, Lemmas 3 and 4 follows that

$Th_{m \rightarrow p} \models [\ \varphi_{cond}(\gamma, t, t_1) \Leftrightarrow q_{cond}(t, t_1)\ ]$ (5)

and

$Th_{m \rightarrow p} \models [\ \varphi_{bh}(\gamma, t_1) \Leftrightarrow q_{prep}(\gamma, t_1)]$ (6)

From (5), (6), definitions of the conditional preparation formula and the normalized conditional preparation state formulae, and the conditions of the proposition it follows

$Th_{m \rightarrow p} \models [\ \varphi_f(\gamma, t) \Leftrightarrow \forall u_1 > t\ [q_{cond}(t, u_1) \rightarrow q_{prep}(\gamma, u_1)]$ & $\forall u_1 > t\ [q_{cond}(t, u_1) \rightarrow q_{prep}(\gamma, u_1)] \Leftrightarrow \forall t_1 > t\ \varphi_{cprep}(\gamma, t, t_1) \Leftrightarrow \varphi_{cprep}(\gamma, t)\ ]$

And from Lemma 5 follows

$Th_{m \rightarrow p} \models [\ \varphi_f(\gamma, t) \Leftrightarrow \forall t_1 > t\ state(\gamma, t) \models q_{cprep}(t, t_1) \Leftrightarrow state(\gamma, t) \models q_{cprep}(t)$ & $state(\gamma, t) \models q_{cprep}(t) \Leftrightarrow state(\gamma, t) \models q_{cprep}\ ]$ ■

Proposition 3

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

Let $\varphi_p(\gamma, t)$ be a past statement for t and $\varphi_f(\gamma, t)$ be a future statement for t. Let $\varphi_{mem}(\gamma, t)$ be the memory formula for $\varphi_p(\gamma, t)$ and $\varphi_{cprep}(\gamma, t)$ the conditional preparation formula for $\varphi_f(\gamma, t)$. Then

$$[\varphi_p(\gamma, t) \Rightarrow \varphi_f(\gamma, t)] \Leftrightarrow [\varphi_{mem}(\gamma, t) \Rightarrow \varphi_{cprep}(\gamma, t)]$$

Proof.

From the Proposition 1 and the Proposition 2 follows

$$\varphi_p(\gamma, t) \Leftrightarrow \varphi_{mem}(\gamma, t) \text{ and } \varphi_f(\gamma, t) \Leftrightarrow \forall t_1 > t\ \varphi_{cprep}(\gamma, t, t_1)$$

Then,

$$[\varphi_p(\gamma, t) \Rightarrow \varphi_f(\gamma, t)] \Leftrightarrow [\varphi_{mem}(\gamma, t) \Rightarrow \forall t_1 > t\ \varphi_{cprep}(\gamma, t, t_1)]$$

So it has been proven that $[\varphi_p(\gamma, t) \Rightarrow \varphi_f(\gamma, t)] \Leftrightarrow [\varphi_{mem}(\gamma, t) \Rightarrow \varphi_{cprep}(\gamma, t)]$ ■

Definition 2.9 (Executable theory from preparation to output)

For a given $\varphi_f(\gamma, t)$ the executable theory from the preparation to the output state(s) $Th_{p\to o}$ consists of the formula

$\forall t'$ state(γ, t') |= preparation_for(output(t'+c, a)) $\Rightarrow$ state(γ, t'+c) |= output(a),

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

Definition 2.10 (Executable specification)

An *executable specification* $\pi(\gamma, t)$ for the component A is defined by a union of the dynamic properties from the executable theories $Th_{o\to m}$, $Th_{m\to p}$ and $Th_{p\to o}$.

Definition 2.11 (Coinciding traces)

Two traces γ_1, γ_2 *coincide* on ontology Ont (denoted by a predicate symbol coincide_on: TRACE x TRACE x ONTOLOGY (ONTOLOGY is a sort that contains all names of ontologies)) iff

$\forall t\ \forall a \in STATATOM_{Ont}$ state(γ_1, t) |= a $\Leftrightarrow$ state(γ_2, t) |= a,

where $STATATOM_{Ont} \subseteq STATPROP_{Ont}$ is the sort, which contains all names of ground atoms expressed in terms of Ont.

Definition 2.12 (Refinement of an externally observable property)

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

(1) $\forall \gamma, t\ \ \pi(\gamma, t) \Rightarrow \varphi(\gamma, t)$

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

Note that for any past interaction statement $\varphi_p(\gamma, t)$ and future interaction statement $\varphi_f(\gamma, t)$ the following holds:

$\forall \gamma 1, \gamma 2$ [coincide_on($\gamma 1, \gamma 2$, InteractionOnt) $\Rightarrow$ $[\varphi_p(\gamma 1, t) \Leftrightarrow \varphi_p(\gamma 2, t)$ & $\varphi_f(\gamma 1, t) \Leftrightarrow \varphi_f(\gamma 2, t)\ ]]$

Lemma 6

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

(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) $\Rightarrow$ [$\varphi(\gamma_1, t) \Leftrightarrow \varphi(\gamma_2, t)$].

Proof sketch.

The transitivity property (1) follows directly from the definition of coinciding traces for coincide_on(γ_1, γ_2, Ont) and coincide_on(γ_2, γ_3, Ont):

$\forall a \in STATATOM_{Ont}\ \ \forall t'$ [state(γ_1, t') |= a $\Leftrightarrow$ state(γ_3, t') |= a] $\Rightarrow$ coincide_on(γ_1, γ_3, Ont)

From

$\forall\ \gamma_1, \gamma_2$ [coincide_on(γ_1, γ_2, 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

If the executable specification $\pi_A(\gamma, t)$ refines the external behavioral 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

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

Proof sketch for Theorem.

$\Leftarrow$ 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 coincide_on(γ, γ_1, InteractionOnt (A)).

Due to Lemma 6, 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

coincide_on(γ, γ_n, InteractionOnt(A)),

and

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

From

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

and

$$\forall\gamma\ \ [\ \wedge\pi_i(\gamma)\ \Rightarrow\ \psi(\gamma, t)\ ]$$

it follows that $\forall\gamma\ \wedge\varphi_i(\gamma)\ \Rightarrow\ \psi(\gamma)$.

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

3. TRANSFORMATION INTO THE FINITE STATE TRANSITION SYSTEM FORMAT

According to the Definition 2.10 the executable specification of a component's behavior consists of the union of three theories $Th_{o\to m}$, $Th_{m\to p}$ and $Th_{p\to o}$, which in turn contain a number of executable dynamic properties. These dynamic properties can be translated into transition rules for a finite state transition system, based on which the same traces are generated as by executing the dynamic properties. For this purpose we use the predicate present_time(t) introduced earlier, which is only true in a state for the current time point t. Further the executable properties from the executable specification, translated into the transition rules are given.

Time increment rules:

present_time(0) $\wedge\ \neg$p $\rightarrow\!\!\!\!\rightarrow$ present_time(1)

present_time(t) $\wedge\ \neg q_{mem} \wedge\ \neg$p $\rightarrow\!\!\!\!\rightarrow$ present_time(t+1)

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

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

Memory state creation rule:

For any state atom p occurring in $\varphi_{cond}(\gamma, t, t_1)$, expressed in the InteractionOnt(A) for component A:

present_time(t) $\wedge$ p $\rightarrow$ [memory(t, p) $\wedge$ stimulus_reaction(p)]

For all other state atoms p

present_time(t) $\wedge$ p $\rightarrow\!\!\!\!\rightarrow$ memory(t, p)

Memory persistence rule:

memory(t, p) $\rightarrow\!\!\!\!\rightarrow$ memory(t, p)

Conditional preparation generation rule:

$q_{mem} \rightarrow\!\!\!\!\rightarrow$ conditional_preparation_for(output(a)),

where a an action or a communication for which state(γ, t'+c) |= output(a) occurs in $\varphi_f(\gamma, t)$.

Preparation state creation rule:

present_time(t') $\wedge$ conditional_preparation_for(output(a)) $\wedge\ q_{cond}$(t) $\wedge\ \bigcap_{p}$ stimulus_reaction(p) $\rightarrow\!\!\!\!\rightarrow$ preparation_for(output(t'+c, a))

for every subformula of the form

present_time(t') $\rightarrow$ preparation_for(output(t'+c, a))

that occurs in q_{cprep}.

Preparation state persistence rule:

preparation_for(output(t+c, a)) $\wedge$ $\neg$output(a) $\rightarrow$ preparation_for(output(t+c, a))

Stimulus reaction state persistence rule:

present_time(t') $\wedge$ stimulus_reaction(p) $\wedge$ $\neg$ preparation_for(output(t'+c, a)) $\rightarrow$ stimulus_reaction(p)

Output state creation rule:

preparation_for(output(t+c, a)) $\wedge$ present_time(t+c-1) $\rightarrow$ output(a), where a is an action or a communication.

4. TRANSFORMATION INTO THE SMV MODEL SPECIFICATION FORMAT

For automatic verification of relationships between dynamic properties of components of different aggregation levels by means of model checking techniques, a corresponding to the behavioral specification of the lower aggregation level representation of a finite state transition system 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 similar 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.

Let us describe the transformation procedure, which is automatically performed by the dedicated software that has been developed.

First, using the standard rules [1] $q_{mem}(t)$ and $q_{cond}(t, t_1)$ expressions for each dynamic property DP_n are transformed into the prenex normal form. Then, for each dynamic property the described below steps 1-3 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. Rewrite in the external behavioral specification all occurrences of the function memory(communicated(t1, a)) by memory(observed(t1, a)) and of the function $\neg$memory(communicated(t1, a)) by $\neg$memory(observed(t1, a)) for some given atom a. For each occurrence of an existential quantifier of the form $\exists$t1 P(t1), where t1 is a time variable name and P(t1) is some function of the form memory(observed(t1, obs_event)), $\neg$memory(observed(t1, obs_event)), memory(output(t1, act_event)), and $\neg$memory(output(t1, act_event)), where obs_event and act_event are some atoms and for each occurrence of a universal quantifier of the form $\forall$t1 P(t1), create an atom (a label) t1 and add to the specification the following:

For memory(observed(t1, obs_event)), $\neg$memory(observed(t1, obs_event)):

```
t1: boolean ;
init(t1):=0;
obs_event: boolean;
init(obs_event):=0;
```

For memory(output(t1, obs_event)) and $\neg$memory(output(t1, obs_event)):

```
t1: boolean ;
init(t1):=0;
act_event: boolean;
init(act_event):=0;
```

Step 2. For each existentially quantified time variable and universally quantified time variable that is not in the scope of any existential quantifier with a time variable:

(a) 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 rules:

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

(b) For each occurrence of the expression Q t1, t2 R t1 memory(output(t1, act_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 rules:

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

(c) For each occurrence of the expression Q t1, t2 R t1 ¬memory(observed(t1, obs_event)), add to the specification the following rules:

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

(d) For each occurrence of the expression Q t1, t2 R t1 ¬memory(output(t1, act_event)), add to the specification the following rules:

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

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;
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;
```

(b) if P3(t) is of the form memory(output(t3, act_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
            !act_event2 & !t2 & t3t1_eq & !act_event3: 0;
            1: t1;
esac;
next(t3):= case
           !t1: 0;
           !act_event2 & !t2 & !act_event3: 0;
           !act_event2 & !t2 & act_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 & !act_event3: 0;
            1: t1;
esac;
next(t3):= case
           !t1: 0;
           !t2 & !act_event3: 0;
           !t2 & act_event3: 1;
           1: t3;
esac;
```

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

```
next(t1):= case
            !act_event2 & !t2 & !act_event3: 0;
            1: t1;
esac;
next(t3):= case
           !t1: 0;
           !act_event2 & !t2 & !act_event3: 0;
           !act_event2 & !t2 & act_event3: 1;
           1: t3;
esac;
```

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

```
next(t1):= case
            !t2 & !act_event3: 0;
            1: t1;
esac;
next(t3):= case
           !t1: 0;
           !t2 & !act_event3: 0;
           !t2 & act_event3: 1;
           1: t3;
esac;
```

(c) 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:

```
neg_t3t1_eq: boolean ;
init(neg_t3t1_eq):=0;
next(neg_t3t1_eq):= case
            t1: 1;
             1: 0;
esac;
next(t1):= case
         !obs_event2 & !t2 & neg_t3t1_eq & 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;
```

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

```
neg_t3t1_eq: boolean ;
init(neg_t3t1_eq):=0;
next(neg_t3t1_eq):= case
            t1: 1;
             1: 0;
esac;
next(t1):= case
            !t2 & neg_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;
```

(d) If $P3(t)$ is of the form ¬memory(output(t3, act_event))

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

```
neg_t3t1_eq: boolean ;
init(neg_t3t1_eq):=0;
next(neg_t3t1_eq):= case
            t1: 1;
             1: 0;
esac;
next(t1):= case
         !act_event2 & !t2 & neg_t3t1_eq & act_event3: 0;
         1: t1;
esac;
next(t3):= case
           !t1: 0;
           !act_event2 & !t2 & act_event3: 0;
           !act_event2 & !t2 & !act_event3: 1;
           1: t3;
esac;
```

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

```
neg_t3t1_eq: boolean ;
init(neg_t3t1_eq):=0;
next(neg_t3t1_eq):= case
            t1: 1;
             1: 0;
esac;
next(t1):= case
            !t2 & neg_t3t1_eq & act_event3: 0;
            1: t1;
esac;
next(t3):= case
           !t1: 0;
           !t2 & act_event3: 0;
           !t2 &!act_event3: 1;
           1: t3;
esac;
```

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

```
next(t1):= case
            !act_event2 & !t2 & act_event3: 0;
            1: t1;
esac;
next(t3):= case
           !t1: 0;
           !act_event2 & !t2 & act_event3: 0;
           !act_event2 & !t2 & !act_event3: 1;
           1: t3;
esac;
```

iiii. For $t3 \leq t2$ and $t1 \leq t3$ add to the specification the following rules:

```
next(t1):= case
            !t2 & act_event3: 0;
            1: t1;
esac;
next(t3):= case
           !t1: 0;
           !t2 & act_event3: 0;
           !t2 &!act_event3: 1;
           1: t3;
esac;
```

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 output(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;
```

5. Case study

To illustrate the proposed approach to verify interlevel relations, consider a running example based on a case study from the area of logistics. This case study was done within the project DEAL (Distributed Engine for Advanced Logistics). For the project description, we refer to *http://www.almende.com/deal/*. A template organizational model was created, based on the informal description of the structure and functioning of the large Dutch logistics company. Only relevant to the actual delivery process actors (roles) and their properties are specified in this model. To secure anonymity of the company, the real names of the organizational units were substituted by general ones.

At the highest aggregation level (level 0) the whole organization is represented as one role. At aggregation level 1, the organization consists of two interacting roles: TC and CI (see Fig.1; explanation for this and the following abbreviations and functional descriptions are given in Table 1). Note, that the organizational model is depicted in a modular way; i.e., components of every aggregation level can be visualized and analyzed both separately and in relation to each other. Consequently, scalability of graphical representation of an organizational model is achieved.

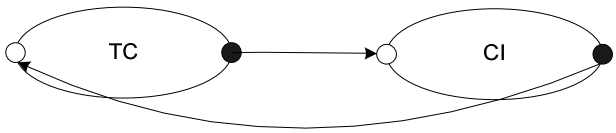

Fig. 1. Representation of the organization at abstraction level 1, which consists of role Transport Company (TC) and role Customer Interaction (CI)

At aggregation level 2 role TC can be refined into three interacting roles: ST, CR, and OP (see Fig.2). All interactions with a customer are conducted within CI role. At aggregation level 2 it consists of two roles: TCR and C (see Fig. 2). Role TCR produces at its output messages from CR and ST departments of the transport company, i.e., CR and ST roles stand as company representatives in certain interactions with a customer. Therefore, the input state of role TCR has influence on the output state of role CR and vice versa. The same holds for role ST.

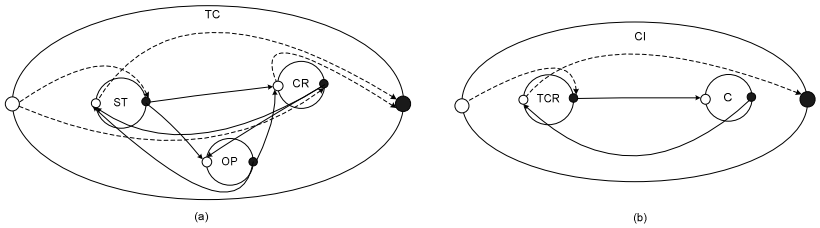

Fig. 2. Representation of (a) the Transport Company (TC) and (b) the Customer Interaction role (CI) at abstraction level 2

The structure of the operational department that is responsible for the direct fulfillment of the order from a customer is depicted at aggregation level 3 in Figure 3. It consists of interacting roles LM, FM, SP and D. Roles LM and SP are able to receive (or transmit) information from (or to) roles outside of role OP by means of interlevel links. Furthermore, in this model only role D interacts with the conceptualized environment.

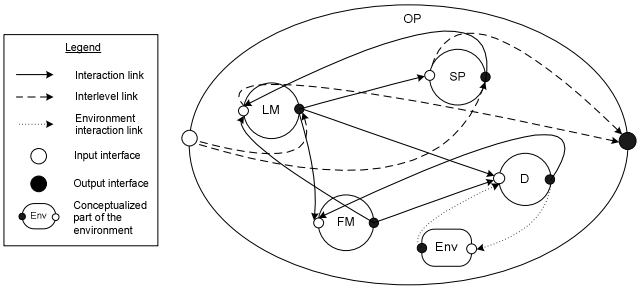

Fig. 3. Representation of the operational department at abstraction level 3

Table 1. Role names, abbreviations, and descriptions for the organizational model in the case study

Role name	Abbreviation	Description
Transport Company	TC	Provides logistic services to customers
Customer Interaction	CI	Identifies interaction rules between a customer and the transport company
Strategy and Tactical Department	ST	Performs analysis and planning of company activities; considers complaints from customers; analyses the satisfaction level of a customer by means of surveys and questionnaires
Custom Relations Department	CR	Handles requests from customers
Operational Department	OP	Responsible for direct fulfillment of the order from a customer
Transport Company Representative	TCR	Mediator role between a customer and the transport company
Customer	C	Generates an order for the transport company; sends inquiries about the delivery status

Sales Person	SP	Assigns an order to a certain load manager, based on the type and the region of a delivery
Load Manager	LM	Assigns orders to suitable trucks and available drivers; assigns fleet managers to drivers; provides CR department with up-to-date information about delivery; provides a driver with instructions in case of a severe problem; informs CR department about possible delays with delivery
Fleet Manager	FM	Keeps constant contact with the assigned drivers; updates automatic support system with actual data on the delivery status; provides consultations for drivers in case of minor problems in transit
Driver	D	Delivers goods; informs a superior fleet manager about the delivery status; interacts (by means of observations and actions) with the conceptualized part of the environment
Environment	Env	Represents the conceptualized environment; in this example only a driver interacts with it

The information distribution property RP1(OP) of role OP defined at aggregation level 2, which expresses that when a severe problem with some delivery occurs, OP should generate a message to CR about possible delay is used in this example as a property of the higher aggregation level. For the purpose of verification, this property is expressed in CTL as follows:

AG (truck_state_T_incident_severe_incident & truck_property_T_operated_by_D & order_property_A20_assigned_to_D →

AF performing_action_preparation_output_OP_communicate_from_to_OP_CR_inform_order_state_A20_delay_severe_incident)

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 higher level property RP1(OP) can be logically related to the conjunction of dynamic properties at the lower aggregation level 3 in the following way:

EP1(Env, T, severe_incident) & EP2(Env, T) & EIP1(Env, D) & RP1(D) & TP1(D, FM) & RP2(FM) & TP2(FM, LM) & RP3(LM) & RP4(LM) & ILP1(LM, OP) ⇒ RP1(OP) (1)

Let us consider the informal and formalized expressions for the properties from the relation (1) that hold for any trace γ

EP1(Env, T, severe_incident) Incident occurrence

Informal description:

In the environment a severe incident with the truck T occurs

Formalization:

∃t1:TIME state(γ, t1, environment)|= truck_state(T, incident, severe_incident)

EP2(Env, T) Stable information about the environnent

Informal description:

Role D (a driver) operates the truck T and is assigned to deliver the order A20; role D is assigned to the fleet manager FM, and FM is in the region of the load manager LM

Formalization:

∀t1:TIME state(γ, t1, environment) |= [truck_property(T, operated_by, D) ∧ order_property(A20, assigned_to, D) ∧ assigned_to(D, FM) ∧ in_region(FM, LM)]

EIP1(Env, D) Incident observation

Informal description:

If an incident happens with a truck, then a driver responsible for this truck will observe this incident

Formalization:

∀t1:TIME ∀T:TRUCK_TYPE ∀D:DRIVER ∀ins:INCIDENT state(γ , t1, environment))|= [truck_state(T, incident, ins) ∧ truck_property(T, operated_by, D)] ⇒ ∃t2 t2>t1 state(γ , t2, input(D)) |= observation_result(truck_state(T, incident, ins), true)

RP1(D) Request for incident solution

Informal description:

If a driver observes an incident with his truck, then s/he will react by generating a request for advice to his fleet manager

Formalization:

∀t1:TIME ∀T:TRUCK_TYPE ∀D:DRIVER ∀ins:INCIDENT ∀FM: FLEET_MANAGER state(γ , t1, input(D)) |= observation_result(truck_state(T, incident, ins), true) & state(γ, t1, environment) |= assigned_to(D, FM)

⇒ ∃t2 t2>t1 state(γ , t2, output(D)) |= to_be_performed(communicate_from_to(D, FM, ask, solution_for_problem(ins, T)))

TP1(D, FM) Request transfer to Fleet Manager

Informal description:

If a driver sends a request to his fleet manager, the fleet manager will receive this request

Formalization:

∀t1:TIME ∀D:DRIVER ∀FM: FLEET_MANAGER ∀req: REQUEST state(γ, t1, output(D)) |= to_be_performed(communicate_from_to(D, FM, ask, req)) & state(γ, t1, environment) |= assigned_to(D, FM)

⇒ ∃t2 t2>t1 state(γ , t2, input(FM)) |= observation_result(communicate_from_to(D, FM, ask, req))

RP2(FM) Request for solution propagation

Informal description:

If a fleet manager receives a request from a driver for advice to solve a severe problem, then s/he will propagate this request further to the regional load manager

Formalization:

∀t1:TIME ∀D:DRIVER ∀T:TRUCK_TYPE ∀FM: FLEET_MANAGER ∀LM: LOAD_MANAGER state(γ , t1, input(FM)) |= observation_result(communicate_from_to(D, FM, ask, solution_for_problem(severe_incident, T))) & state(γ, t1, environment) |= in_region(FM, LM)

⇒ ∃t2 t2>t1 state(γ , t2, output(FM)) |= to_be_performed(communicate_from_to(FM, LM, ask, solution_for_problem(severe_incident, T)))

TP2(FM, LM) Request transfer to Load Manager

Informal description:

If a fleet manager sends a request to a regional load manager, the regional load manager will receive this request

Formalization:

∀t1:TIME ∀FM: FLEET_MANAGER ∀LM: LOAD_MANAGER ∀req: REQUEST state(γ ,t1, output(FM)) |= to_be_performed(communicate_from_to(FM, LM, ask, req))

⇒ ∃t2 t2>t1 state(γ , t2, input(LM)) |= observation_result(communicate_from_to(FM, LM, ask, req))

RP3(LM) Change of a delivery status

Informal description:

If a load manager receives a request from a fleet manager for advice to solve a severe problem, then s/he officially identifies the incident as severe and changes into “delay” the state of the corresponding delivery order in the information system.

Formalization:

∀D:DRIVER ∀t1:TIME ∀FM: FLEET_MANAGER ∀T:TRUCK_TYPE ∀LM: LOAD_MANAGER ∀ON:ORDER_NUM state(γ , t1, input(LM)) |= observation_result(communicate_from_to(FM, LM, ask, solution_for_problem(severe_incident, T))) & state(γ, t1, environment) |= [order_property(ON, assigned_to, D) ∧ truck_property(T, operated_by, D)]

⇒ ∃t2 t2>t1 state(γ , t2, output(LM)) |= to_be_performed(change(order_state(ON, delay, severe_incident)))

RP4(LM) Informing CR about a delivery status

Informal description:

If a load manager changes a state of a delivery order object, then the information about this change is generated at the output of the load manager role for the customer relation role.

Formalization:

∀t1:TIME ∀LM: LOAD_MANAGER ∀ON:ORDER_NUM ∀st: STATE_TYPE ∀r: REASON state(γ , t1, output(LM)) |= to_be_performed(change(order_state(ON, st, r)))

⇒ ∃t2 t2>t1 state(γ, t2, output(LM))|= to_be_performed(communicate_from_to(LM, CR, inform, order_state(ON, st, r)))

ILP1(LM, OP) Generation of information about the state change of a delivery order object

Informal description:

If a load manager communicates information about the change of a delivery status to the customer relation role, then the operational department role transmits this information to the customer relation department role.

Formalization:

∀t1:TIME ∀LM: LOAD_MANAGER ∀ON:ORDER_NUM ∀st: STATE_TYPE ∀r: REASON state(γ, t1, output(LM))|= to_be_performed(communicate_from_to(LM, CR, inform, order_state(ON, st, r)))

⇒ ∃t2 t2>t1 state(γ, t2, output(OP))|= to_be_performed(communicate_from_to(OP, CR, inform, order_state(ON, st, r)))

By applying the procedure described in Section 3 to the specification that comprises all identified above properties defined at aggregation level 3 is transformed into the finite state transition system format required for performing model checking. The specification for the considered example is given below.

present_time(t) & ¬performing_action(preparation(output_LM_communicate_from_to_LM_CR_inform_order_state_A20_delay_severe_incident)) → present_time(t+1)

truck_state_T_incident_severe_incident & truck_property_T_operated_by_D → observed(input_D_truck_state_T_incident_severe_incident)

present_time(t) & observed(input_D_truck_state_T_incident_severe_incident) → memory(t, observed(input_D_truck_state_T_incident_severe_incident))

memory(t, observed(input_D_truck_state_T_incident_severe_incident)) → memory(t, observed(input_D_truck_state_T_incident_severe_incident))

present_time(t) & memory(t, observed(input_D_truck_state_T_incident_severe_incident)) & assigned_to_D_FM → qcprep1

present_time(t) & qcprep1 → preparation(output_D_communicate_from_to_D_FM_ask_solution_for_problem_severe_incident_T)

preparation(output_D_communicate_from_to_D_FM_ask_solution_for_problem_severe_incident_T) → performing_action(output_D_communicate_from_to_D_FM_ask_solution_for_problem_severe_incident_T)

performing_action(output_D_communicate_from_to_D_FM_ask_solution_for_problem_severe_incident_T) → observed(input_FM_communicate_from_to_D_FM_ask_solution_for_problem_severe_incident_T)

present_time(t) & observed(input_FM_communicate_from_to_D_FM_ask_solution_for_problem_severe_incident_T) → memory(t, observed(input_FM_communicate_from_to_D_FM_ask_solution_for_problem_severe_incident_T))

memory(t, observed(input_FM_communicate_from_to_D_FM_ask_solution_for_problem_severe_incident_T)) → memory(t, observed(input_FM_communicate_from_to_D_FM_ask_solution_for_problem_severe_incident_T))

present_time(t) & memory(t, observed(input_FM_communicate_from_to_D_FM_ask_solution_for_problem_severe_incident_T)) & in_region_FM_LM → qcprep2

present_time(t) & qcprep2 → preparation(output_FM_communicate_from_to_FM_LM_ask_solution_for_problem_severe_incident_T)

preparation(output_FM_communicate_from_to_FM_LM_ask_solution_for_problem_severe_incident_T) → performing_action(output_FM_communicate_from_to_FM_LM_ask_solution_for_problem_severe_incident_T)

performing_action(output_FM_communicate_from_to_FM_LM_ask_solution_for_problem_severe_incident_T) → observed (input_LM_communicate_from_to_FM_LM_ask_solution_for_problem_severe_incident_T)

present_time(t) & observed (input_LM_communicate_from_to_FM_LM_ask_solution_for_problem_severe_incident_T) → memory(t, observed(input_LM_communicate_from_to_FM_LM_ask_solution_for_problem_severe_incident_T))

memory(t, observed(input_LM_communicate_from_to_FM_LM_ask_solution_for_problem_severe_incident_T)) →

memory(t, observed(input_LM_communicate_from_to_FM_LM_ask_solution_for_problem_severe_incident_T))

present_time(t) & memory(t, observed(input_LM_communicate_from_to_FM_LM_ask_solution_for_problem_severe_incident_T)) & order_property_A20_assigned_to_D & truck_property_T_operated_by_D → qcprep3

present_time(t) & qcprep3 → preparation(change_order_state_A20_delay_severe_incident)

preparation(change_order_state_A20_delay_severe_incident) → performing_action(preparation(change_order_state_A20_delay_severe_incident))

performing_action(preparation(change_order_state_A20_delay_severe_incident)) →

performing_action(preparation(output_LM_communicate_from_to_LM_CR_inform_order_state_A20_delay_severe_incident))

performing_action(preparation(output_LM_communicate_from_to_LM_CR_inform_order_state_A20_delay_severe_incident)) → performing_action(preparation(output_OP_communicate_from_to_OP_CR_inform_order_state_A20_delay_severe_incident))

The automatic verification in the SMV model checking tool of the property RP1(OP) on the considered model showed that the previously identified logical relation (1) indeed holds.

REFERENCES

[1] Fitting, M. First-order Logic and Automated Theorem Proving. 2nd edition, Springer-Verlag, 1996.