Specification and Verification of Dynamics in Cognitive Agent Models

Tibor Bosse[1], Catholijn M. Jonker[2], Lourens van der Meij[1],
Alexei Sharpanskykh[1], and Jan Treur[1]

[1] *Vrije Universiteit Amsterdam, Department of Artificial Intelligence,*
De Boelelaan 1081a, 1081 HV Amsterdam, The Netherlands
[2] *Radboud Universiteit Nijmegen, Nijmegen Institute for Cognition and Information,*
Montessorilaan 3, 6525 HR Nijmegen, The Netherlands
{tbosse, lourens, sharp, treur}@few.vu.nl, C.Jonker@nici.ru.nl

Abstract

Within many domains, among which biological and cognitive areas, multiple interacting processes occur among agents with dynamics that are hard to handle. Current approaches to analyse the dynamics of such processes, often based on differential equations, are not always successful. As an alternative to differential equations, this paper presents the predicate logical Temporal Trace Language (TTL) for the formal specification and analysis of dynamic properties. This language supports the specification of both qualitative and quantitative aspects, and therefore subsumes specification languages based on differential equations. A software environment has been developed for TTL, that supports editing TTL properties and enables the formal verification of properties against a set of traces. The TTL environment proved its value in a number of projects within different domains.

1. Introduction

In domains such as Biology and Cognitive Science, the dynamics of the multiple interacting processes among different agents involved poses modelling challenges. Currently, differential equations are among the techniques most often used to address this challenge, with partial success. For example, in the area of intracellular processes, hundreds or more reaction parameters (for which reliable values are rarely available) are needed to model the processes in question. Thus, describing these processes in terms of differential equations can seriously compromise the feasibility of the model. Likewise, in the area of Cognitive Science, the Dynamical Systems Theory that is also based on differential equations (DST, see e.g., [21]), is well practiced and successful. However, the models typically only address lower-level agent cognitive processes such as sensory or motor processing. DST has less to offer for modelling the dynamics of higher-level processes with a mainly qualitative character, such as

agent reasoning, complex task performance, and certain capabilities of language processing.

For formal qualitative modelling of processes at a high level of abstraction, logic-based methods have proved useful. For example, variants of modal temporal logic [2, 12, 14, 19, 24] gained popularity in agent technology. However, many of the logic-based methods lack the quantitative expressivity, needed, e.g., for modelling processes for which precise timing relations play an essential role (e.g., biological and chemical processes).

Thus, within several disciplines the need exists for general modelling and analysis techniques capable to deal with complex agent systems that comprise both quantitative and qualitative aspects. This paper introduces the Temporal Trace Language (TTL) as such a technique for the analysis of dynamic properties within complex domains, and especially, for the cognitive domain. In Section 2, a novel perspective is put forward for the development of such a technique, based on the idea of checking dynamic properties on given sets of traces. Section 3 shows how dynamics of an agent system can be modelled using the TTL language. Examples of the application of TTL are presented in Section 4. Section 5 describes the tools that support the TTL modelling environment in detail. In particular, the TTL Property Editor and the TTL Checker Tool are discussed. Section 6 is a conclusion.

2. Perspective of this Paper

As follows from the discussion above, the demands for dynamic modelling and analysis approaches suitable for specifying agent systems in natural domains are nontrivial. In particular, the possibility of both discrete and continuous modelling of a system at different aggregation levels is demanded. Furthermore, numerical expressivity is required for modelling systems with explicitly defined quantitative relations best presented by difference or differential equations. Moreover, for specifying qualitative aspects of a system, modelling languages

should be able to express logical relationships between parts of a system.

Desiderata for analysis techniques include both the generation and formalisation of simulated and empirical trajectories or traces, as well as analysis of complex dynamic properties of such traces and relationships between such properties. A *trace* as used here represents a temporally ordered sequence of states of an agent system. Each state is characterised by a number of *state properties* that hold. Simulated traces may be obtained by performing simulations based on both quantitative (or continuous) and qualitative (or discrete) variables.

Taken together, the desiderata for modelling languages and analysis techniques described above are not easy to fulfil. On the one hand, high expressivity is desired, on the other hand feasible analysis techniques are demanded. To provide automated support for these analyses the expressivity of the modelling language can be limited, thereby compromising the desiderata for modelling languages. For example, the expressivity may be limited to difference and differential equations as in DST (excluding logical relationships), or to propositional modal temporal logics (excluding numerical relationships). In the former case, calculus can be exploited to do simulation and analysis based on continuous variables only [21]. In the latter case, simulation is based on a specific logical executable format, which does not allow expressions involving continuous variables (e.g., executable temporal logic [2]). Another possibility is to use a number of dedicated formal languages with limited expressiveness and related to them analysis techniques for checking different particular static and dynamic aspects of a system (e.g., structural consistency of a model, dynamic aspects of execution), as proposed in the methodology for the development of correct software KORSO [11]. The languages used in this project describe different formats of system specifications, relations between them (e.g., by refinement based on proof obligations) and the temporal development of these specifications for all phases of the software life cycle. However, in order to guarantee the overall correctness of a system some properties are required to be expressed using more than one language with different types of semantics. Thus, the problem of verification across different not related proof systems arises that is not addressed in this project.

The problem of checking relationships between dynamic properties of a system, identified above as one of the desiderata for analysis techniques, is essentially the problem of justifying entailment relations between sets of properties defined at different aggregation levels of a system's representation. In general, entailment relations can be established either by logical proof procedures or by checking properties of a higher aggregation level on the set of all theoretically possible traces generated by executing a system specification that consists of properties of a lower aggregation level (i.e., by performing model checking [12, 19, 24]). To make it feasible to check relationships between dynamic properties, expressivity of the language for these properties has to be sacrificed to a large extent. However, checking properties on a given set of traces of practical size (instead of all theoretically possible ones), obtained empirically or by simulation, is computationally much cheaper. Therefore, in that case the language for these properties can be more expressive, such as the sorted predicate logic temporal trace language TTL described in this paper. TTL fulfils all of the identified above desiderata for modelling languages and can be used both for formalisation of empirical and simulated traces and for analysis of properties on traces. Although TTL cannot be used to generate traces by simulation, an executable sublanguage of TTL, such as LEADSTO, cf. [6], may be defined for this purpose. Moreover, decidable fragments of TTL may be defined for the analysis of relationships between dynamic properties of a system.

Finally, having a language for simulation and languages for analysis within one subsuming language also opens the possibility of having a declarative specification of a simulation model, and thus to involve simulation models in logical analyses.

3. A Language to Model Agent Behaviour

The Temporal Trace Language (TTL) presented here is developed from the assumption that the dynamics of an agent system can be described as evolution of states of agents and an environment over time, as for modal temporal logics, see e.g., [2, 12, 14, 19, 24]. TTL has some similarities with situation calculus, see [22] and event calculus, see [16]. Time in TTL is assumed to be linearly ordered and depending on the application, it may be dense (e.g., the real numbers), or discrete (e.g., the set of integers or natural numbers or a finite initial segment of the natural numbers), or any other form with a linear ordering. An agent interacts with a dynamic environment via its *input* and *output* (interface) states. At its input the agent receives observations from the environment whereas at its output it generates actions that can change a state of the environment.

An agent state at a certain point in time as used here is an indication of which of the state properties of the agent and its environment (e.g., observations and actions) are true (hold) at that time point. For specifying state properties for the input, output, internal, and external states of an agent A, ontologies, named IntOnt(A), InOnt(A), OutOnt(A), and ExtOnt(A) respectively, are used which are specified by a number of sorts, sorted constants, variables, functions and predicates (i.e., a signature). State

properties are specified using a standard multi-sorted first-order predicate language based on such ontologies. For example, a state property expressed as a predicate pain may belong to IntOnt(A), whereas the atom has_temperature(environment, 7) may belong to ExtOnt(A).

To characterize the dynamics of the agent and the environment, *dynamic properties* relate properties of states at certain points in time.

To enable reasoning about the dynamic properties of arbitrary systems the language TTL includes special sorts, such as: TIME (a set of linearly ordered time points), STATE (a set of all state names of an agent system), TRACE (a set of all trace names; a trace or a trajectory can be thought of as a timeline with for each time point a state), and STATPROP (a set of all state property names). Throughout the paper, variables such as t, t1, t2, t', t" stand for variables of the sort TIME; and variables such as γ, $\gamma 1$, $\gamma 2$ stand for variables of the sort TRACE.

A state of an agent is related to a state property via the satisfaction relation $\models$ formally defined as a binary infix predicate (or by holds as a binary prefix predicate in the software environment). For example, "in the output state of agent A in trace γ at time t property p holds" is formalized by state(γ , t, output(A)) $\models$ p. If the indication of an agent aspect is not essential, the third argument is left out: state(γ, t) $\models$ p.

Both state(γ , t, output(A)) and p are terms of the TTL language. TTL terms are constructed by induction in a standard way for sorted predicate logic from variables, constants and functional symbols typed with TTL sorts. Dynamic properties are expressed by TTL-formulae inductively defined by:

(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.
(2) If τ_1, τ_2 are terms of any TTL sort, then $\tau_1 = \tau_2$ is an atomic TTL formula.
(3) If t_1, t_2 are terms of sort TIME, then $t_1 < t_2$ is an atomic TTL formula.
(4) The set of well-formed TTL-formulae is defined inductively in a standard way based on atomic TTL-formulae using boolean propositional connectives and quantifiers.

For example, the dynamic property

'in any trace γ, if at any point in time t1 agent A observes that it is dark in the room, whereas earlier a light was on in this room, then there exists a point in time t2 after t1 such that at t2 in the trace γ agent A switches on a lamp'

is expressed in formalized form as:

$\forall$t1 [[state(γ, t1, input(A)) $\models$ observed(dark_in_room) &
$\exists$t0<t1 [state(γ, t0, input(A)) $\models$ observed(light_on)]
$\Rightarrow$ $\exists$t2 $\geq$ t1 state(γ, t2, output(A)) $\models$ performing_action(switch_on_light)]

As TTL uses order-sorted predicate logic as a point of departure, it inherits the standard semantics of this variant of predicate logic. That is, the semantics of TTL is defined in a standard way, by interpretation of sorts, constants, functions and predicates, and a variable assignment. However, in addition the semantics involves some specialised aspects. As a number of standard sorts are present, the elements of these sorts are limited to instances of specified terms in these sorts, as is usual, for example, in logic programming semantics. For example, for the sort TIME it is assumed that in its semantics its elements consist of the time points of the fixed time frame chosen. Moreover, for the sort TRACE, it is assumed that in its semantics its elements consists of a (limited) number of elements named by constants. Furthermore, for the sort STATPROP for state properties it is assumed that in its semantics its elements consist of the set of terms denoting the propositions built in a chosen state language (this is called reification). A full description of the technical details of TTL's semantics is beyond the scope of the current paper. For this purpose, see [23].

By executing dynamic properties traces can be generated and visualised, for example as in Figure 1. Here, the time frame is depicted on the horizontal axis. The names of predicates are shown on the vertical axis. A dark box on top of the line indicates that the predicate is true during that time period.
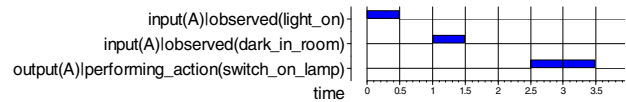

Figure 1. Example visualisation of a trace

4. Application Areas

The TTL language and its supporting software environment have been applied in research projects addressing different topics in cognitive domains, such as human reasoning, conditioning, consciousness, psychotherapy, and philosophy of mind. The main research goal in these projects was to analyse the behavioural dynamics of the agents involved (e.g., [5, 7, 8, 9, 10]). TTL was used to formalise dynamic properties of these processes at a high level of abstraction. Next, such properties were automatically checked against simulated or empirical traces. Examples of the application of TTL in different areas are presented in this section.

4.1 Modelling and Analysis of Hybrid Systems

Hybrid systems incorporate both continuous and discrete components. The dynamics of the former can be described by differential equations, those of the latter can be represented by finite-state automata. Both continuous and discrete dynamics of components influence each other. In particular, the input to the continuous dynamics is the result of some function of the discrete state of a system; whereas the input of the discrete dynamics is determined by the value of the continuous state.

A modelling method for hybrid systems should be capable of expressing both quantitative and qualitative properties of the system and integrating them into one model. TTL satisfies this requirement. Systems of differential equations can be expressed in TTL using discrete or dense time frames. As an example, Euler's method, see [20], for solving differential equations is modelled in TTL. Euler's method approximates a differential equation $dy/dt = f(y)$ with the initial condition $y(t_0)=y_0$ by a difference equation $y_{i+1}=y_i+h*f(y_i)$ ($i \geq 0$ is the step number and $h>0$ is the integration step size). This equation can be modelled in TTL in the following way:

$$\forall \gamma \ \forall t \ \forall v: \text{LVALUE}^{\text{GTERMS}} \ \text{state}(\gamma, t) \models \text{has\_value}(y, v) \quad \Rightarrow$$
$$\text{state}(\gamma, t+1) \models \text{has\_value}(y, v + h \bullet f(v))$$

States specify the respective values of y at different time points and the difference equation is modelled by a transition rule from the current to the successive state. The traces γ satisfying the above dynamic property are the solutions of the difference equation. More precise and stable numerical approximation methods (e.g., Runge-Kutta, dynamic step size, see [20]) can be expressed in TTL in a similar manner.

4.2 Analysis of Trace Conditioning in TTL

The example given in this section is taken from [5]. In that paper, TTL is used to analyse the temporal dynamics of trace conditioning. In general, research into conditioning is aimed at revealing the principles that govern associative learning. An important issue in conditioning processes is the adaptive timing of the conditioned response to the appearance of the unconditioned stimulus. This feature is most apparent in an experimental procedure called *trace conditioning*. In this procedure, a trial starts with the presentation of a *warning stimulus* (S1, comparable to a conditioned stimulus). After a blank interval, called the *foreperiod*, an *imperative stimulus* (S2, comparable to an unconditioned stimulus) is presented to which the participant responds as fast as possible. The *reaction time* to S2 is used as an estimate of the conditioned state of preparation at the moment S2 is presented. In this case, the conditioned response obtains its maximal strength, here called *peak level*, at a moment in time, called *peak time*, that closely corresponds to the moment the unconditioned stimulus occurs.

Machado [18] developed a basic model that describes the dynamics of these conditioning processes in terms of differential equations. The structure of this model is shown in Figure 2. The model posits a layer of *timing nodes* and a single *preparation node*. Each timing node is connected both to the next (and previous) timing node and to the preparation node. The connection between each timing node and the preparation node (called *associative*

link) has an adjustable weight associated to it. Upon the presentation of a warning stimulus, a cascade of activation propagates through the timing nodes according to a regular pattern. Owing to this regularity, the timing nodes can be likened to an internal clock or pacemaker. At any moment, each timing node contributes to the activation of the preparation node in accordance with its activation and its corresponding weight. The activation of the preparation node reflects the participant's preparatory state, and is as such related to reaction time. The weights reflect the state of conditioning, and are adjusted by learning rules, of which the main principles are as follows. First, *during* the foreperiod extinction takes place, which involves the decrease of weights in real time in proportion to the activation of their corresponding timing nodes. Second, *after* the presentation of the imperative stimulus a process of reinforcement takes over, which involves an increase of the weights in accordance with the current activation of their timing nodes, to preserve the importance of the imperative moment. Machado describes the more detailed dynamics of the process by a mathematical model (based on linear differential equations), representing the (local) temporal relationships between the variables involved. For example,

$$dX(t,n)/dt = \lambda X(t,n-1) - \lambda X(t,n)$$

expresses how the activation level of the n-th timing node $X(t+dt,n)$ at time point $t+dt$ relates to this level $X(t,n)$ at time point t and the activation level $X(t,n-1)$ of the (n-1)-th timing node at time point t. Similarly, as another example,

$$dW(t,n)/dt = -\alpha X(t,n)W(t,n)$$

relates the n-th weight $W(t+dt,n)$ at time point $t+dt$ to this weight $W(t,n)$ at time point t and the activation level $X(t,n)$ of the n-th timing node at time point t.

S1

Timing nodes with activation level X

Associative links of variable weight W

Preparation node
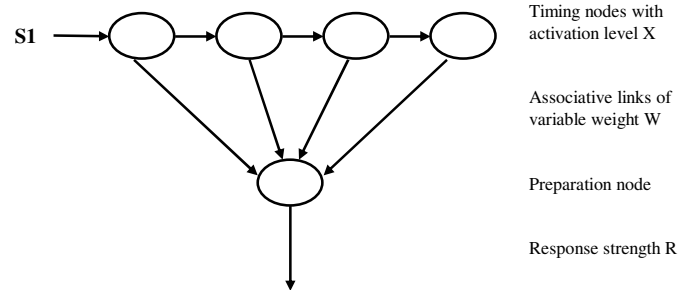
Response strength R

Figure 2. Structure of Machado's conditioning model.

In [5], a number of dynamic properties relevant for trace conditioning have been formalised in TTL. These properties were taken from the existing literature on conditioning, such as [17], in which they were mainly expressed informally. TTL turned out useful to express these properties in a formal manner. An example of such a property (taken from [17], p.372) is given below, both in informal, semi-formal and in formal notation:

Global Hill Preparation

Informal: 'The state of conditioning implicates an increase and decay of response-related activation as a critical moment is bypassed in time'.

Semi-formal: 'In trace γ, if at t1 a stimulus s1 starts, then the preparation level will increase from t1 until t2 and decrease from t2 until t1 + u, under the assumption that no stimulus occurs too soon (within u time) after t1.' *Formally:*

has_global_hill_prep(γ:TRACE, t1,t2:TIME, u:INTEGER) ≡
∀t',t":TIME ∀p',p":REAL
[state(γ, t1) ⊨ stimulus_occurs &
¬ stimulus_starts_within(γ, t1, t1+u) &
state(γ, t') ⊨ preparation_level(p') &
state(γ, t") ⊨ preparation_level(p")
⟹ [t1 ≤ t' < t" ≤ t2 & t" ≤ t1 + u ⟹ p' < p"] &
[t2 ≤ t' < t" ≤ t1 + u ⟹ p' > p"]]

Here, stimulus_starts_within is defined as follows:

stimulus_starts_within(γ:TRACE, t1,t2:TIME) ≡
∃t:TIME [state(γ, t) ⊨ stimulus_occurs & t1 < t < t2]

These (and various similar) properties were automatically verified using the TTL checker tool against a number of (empirical and simulation) traces. Among these properties were also properties that compare different traces, such as:

'the conditioned response takes more time to build up and decay and its corresponding asymptotic value is lower when its corresponding critical moment is more remote from the warning signal.' (cf. [17])

Such properties cannot be expressed, for example, in modal temporal logics, just like familiar properties such as 'exercise improves skill', expressing that the more intensive a training history, e.g., of an athlete, the better the skill will be.

4.3 Application of TTL in Other Areas

Besides the conditioning area, TTL has been applied in many other domains as well. In order to give a representative overview in limited space, below a number of TTL formulae used in other domains are presented (both in informal and formal notation):

Proper Rejection Grounding

From the domain of human reasoning [10]:
'In any trace γ, if an assumption is rejected, then earlier on there was a prediction for it that did not match the corresponding observation result'.

∀t ∀A:INFO_EL ∀S1:SIGN
state(γ, t) ⊨ rejected(A,S1) ⟹
[∃t':TIME ∃B:INFO_EL ∃S2,S3:SIGN
state(γ, t') ⊨ prediction_for(B, S2, A, S1) &
state(γ, t') ⊨ observation_result(B, S3) &
S2 ≠ S3 & t' ≤ t1]

Representational Content of c

From a paper about representational content (cf. [15]) for the mental state of an agent that intensively interacts with the environment [9]:

'In any trace γ, internal state c occurs iff in the past once observation o1 occurred, then action a1(1), then o2(1), then a1(2), then o2(2), then a1(3), and finally o2(3)'.

∀t1,t2,t3,t4,t5,t6,t7 [t1≤t2≤t3≤t4≤t5≤t6≤t7
& state(γ, t1, input) ⊨ o1
& state(γ, t2, output) ⊨ a1(1) & state(γ, t3, input) ⊨ o2(1)
& state(γ, t4, output) ⊨ a1(2) & state(γ, t5, input) ⊨ o2(2)
& state(γ, t6, output) ⊨ a1(3) & state(γ, t7, input) ⊨ o2(3)
⟹ ∃t8 ≥ t7 state(γ, t8, internal) ⊨ c]
& ∀t8 [state(γ, t8, internal) ⊨ c ⟹
∃t1,t2,t3,t4,t5,t6,t7 t1≤t2≤t3≤t4≤t5≤t6≤t7≤t8
& state(γ, t1, input) ⊨ o1
& state(γ, t2, output) ⊨ a1(1) & state(γ, t3, input) ⊨ o2(1)
& state(γ, t4, output) ⊨ a1(2) & state(γ, t5, input) ⊨ o2(2)
& state(γ, t6, output) ⊨ a1(3) & state(γ, t7, input) ⊨ o2(3)]

Learning Behaviour of Aplysia

From a study [8] of adaptive processes of the sea hare Aplysia Californica [13]:
'In any trace γ, if a siphon touch occurs, and at three different earlier time points t1, t2, t3, a siphon touch occurred, directly followed by a tail shock, then the animal will contract'.

∀t [state(γ, t) ⊨ siphon_touch &
∃t1, t2, t3, t4, t5, t6
t1<t2 & t2<t3 & t3<t4 & t4<t5 & t5<t6 & t6<t &
state(γ, t1) ⊨ siphon_touch & state(γ, t2) ⊨ tail_shock &
state(γ, t3) ⊨ siphon_touch & state(γ, t4) ⊨ tail_shock &
state(γ, t5) ⊨ siphon_touch & state(γ, t6) ⊨ tail_shock]
⟹ ∃t7 t7≥t & state(γ, t7) ⊨ contraction

Food Delivery Succesfulness

From an analysis [7] of the domain of ant colony behaviour [4]:
'In any trace γ, there is at least one ant that brings food back to the nest'.

∃t ∃a:ANT ∃l:LOCATION ∃e:edge
state(γ, t) ⊨ is_at_location_from(a, l, e) &
state(γ, t) ⊨ nest_location (l) &
state(γ, t) ⊨ to_be_performed(a, drop_food)

5. Tools

This section presents the software environment[1] that was built in SWI-Prolog to support the process of specification and automated verification of dynamic properties on a limited set of traces. Basically, this software environment consists of two closely integrated tools: the Property Editor and the Checker Tool.

The Property Editor provides a user-friendly way of building and editing properties in TTL. By means of graphical manipulation and filling in forms a TTL specification can be constructed. TTL specifications may also be provided as plain text. When a TTL specification is created, the Checker Tool can be used to verify automatically whether a TTL property from the specification holds for a given set of traces. User interaction with the tools involves three separate actions:

---

[1] The software can be downloaded from the following URL: http://www.cs.vu.nl/~wai/TTL.

IEEE
COMPUTER
SOCIETY

1. Loading, editing, and saving a TTL specification in the Property Editor (see Figure 3).
2. Loading and inspecting traces to be checked by activating the Trace Manager. Both, traces produced by simulations (see [6]) and empirical traces can be used for verification. Empirical traces provided to the TTL Checker may be obtained by formalizing empirical data from log-files produced by information systems or from results of experiments.
3. Checking a property against a set of loaded traces by the Checker Tool. The property is compiled and checked, and the result is presented to the user.
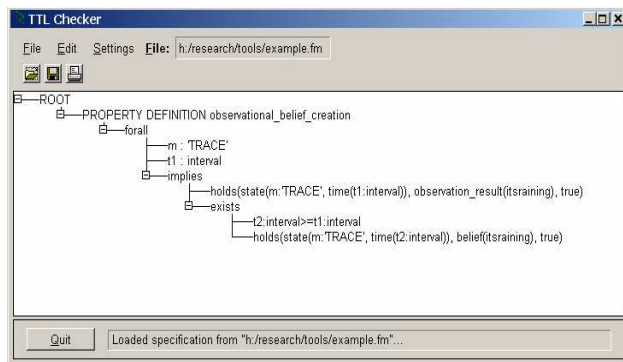

Figure 3. The TTL Checking Environment

The following sections provide more details on implementation of these tools. In particular, Section 5.1 describes the implementation of the TTL Editor and Section 5.2 discusses the verification procedure underlying the TTL checker.

5.1 Implementation of the TTL Editor

A *TTL specification* constructed in the TTL Property Editor consists of a number of user-defined property definitions and sort definitions. A property definition consists of a header (property name and arguments, i.e., prop_name(v1:s1, v2:s2)) and a body (a TTL formula). Arbitrary sorts may be defined by enumerating their elements.

A *TTL formula* is constructed from atomic TTL formulae by conjunction, (Formula1 and Formula2), disjunction (Formula1 or Formula2), negation (not Formula), implication and quantification (forall ([v1:s1, v2:s2], Formula), exists ([v1:s1, v2:s2 < term2], Formula)).

Atomic TTL formulae correspond to user-defined properties, *holds atoms* (e.g., holds(state(trace1, t, output(ew)), a1 $\wedge$ a2) or state(trace1, t, output(ew)) $\vDash$ a1 $\wedge$ a2), mathematical expressions (e.g. term1 = term2, term1 > term2) and built-in properties (i.e., complex properties encoded into the implementation language).

All TTL formulae are constructed from *terms* that are implemented as Prolog terms (e.g., fn(t1,t2) , n1, t1 + t3, 1.3).

Constants, variables and functions from which terms are constructed should be typed with appropriate sorts. For example, each variable should be declared as variable_name: sort. The software supports a number of built-in sorts, among which sorts for integer, real and range of integers (i.e., sorts integer, real, between(i1:integer,i2:integer)), the sort for the set of all states (STATE) and the sort for the set of all traces (TRACE). Furthermore, libraries with predefined general purpose and domain-specific sorts and functions are available for creating terms.

5.2 Verification by the TTL Checker

After a TTL property is specified in the Editor and traces being loaded by the Trace Manager, the Checker Tool may be used to determine if the considered property holds on the loaded traces. To perform such verification an algorithm has been developed.

The verification algorithm is a backtracking algorithm that systematically considers all possible instantiations of variables in the TTL formula under verification. However, not for all quantified variables in the formula the same backtracking procedure is used. Backtracking over variables occurring in holds atoms is replaced by backtracking over values occurring in the corresponding holds atoms in traces under consideration. Since there are a finite number of such state atoms in the traces, iterating over them often will be more efficient than iterating over the whole range of the variables occurring in the holds atoms. Formulae that contain variables quantified over infinite sorts not occurring in a holds atom cannot be checked by the TTL Checker.

As time plays an important role in TTL-formulae, special attention is given to continuous and discrete time range variables. Because of the finite variability property of TTL traces (i.e., only a finite number of state changes occur between any two time points), it is possible to partition the time range into a minimum set of intervals within which all atoms occurring in the property are constant in all traces. Quantification over continuous or discrete time variables is replaced by quantification over this finite set of time intervals.

In order to increase the efficiency of verification, the TTL formula that needs to be checked is compiled into a Prolog clause. Compilation is obtained by mapping conjunctions, disjunctions and negations of TTL formulae to their Prolog equivalents, and by transforming universal quantification into existential quantification. Thereafter, if this Prolog clause succeeds, the corresponding TTL formula holds with respect to all traces under consideration.

The complexity of the algorithm has an upper bound in the order of the product of the sizes of the ranges of all

quantified variables. However, if a variable occurs in a holds atom, the contribution of that variable is no longer its range size, but the number of times that the holds atom pattern occurs (with different instantiations) in trace(s) under consideration. The contribution of an isolated time variable is the number of time intervals into which the traces under consideration are divided.

The specific optimizations discussed above make it possible to check realistic dynamic properties with reasonable performance. In particular, checking the property 'Learning Behaviour of Aplysia' given in Section 4.3 (involving eight different time points) against a single trace with three state atoms occurring in the verified formula and 28 changes of atom values over time takes 0.76 sec. on a regular PC. With the increase of the number of traces with similar complexity as the first one, the verification time grows linearly: for 3 traces - 3.9 sec., for 5 traces - 6.59 sec. However, the verification time is polynomial in the number of isolated time range variables occurring in the formula under verification.

6. Conclusion

This paper presents the predicate logical Temporal Trace Language (TTL) for the formal specification and analysis of dynamic properties of cognitive agent models. Although the language has a logical foundation, it supports the specification of both qualitative and quantitative aspects, and subsumes specification languages based on differential equations. TTL allows for explicit reference to time points and time durations, which enables modelling of the dynamics of continuous real-time phenomena. Furthermore, more specialised languages can be defined as a sublanguage of TTL. For the purpose of simulation, the executable language LEADSTO has been developed [6]. For verification of properties, different decidable fragments of predicate logic (e.g., [1]) can be defined as sublanguages of TTL.

TTL has some similarities with the situation calculus [22] and the event calculus [16], which are two well-known formalisms for representing and reasoning about temporal domains. However, a number of important syntactic and semantic distinctions exist between TTL and both calculi. In particular, the central notion of the situation calculus - a situation - has different semantics than the notion of a state in TTL. That is, by a situation is understood a history or a finite sequence of actions, whereas a state in TTL is associated with the assignment of truth values to all state properties (a "snapshot" of the world). Moreover, in contrast to the situation calculus, where transitions between situations are described by actions, in TTL actions are in fact properties of states.

Moreover, although a time line has been recently introduced to the situation calculus [22], still only a single path (a temporal line) in the tree of situations can be explicitly encoded in the formulae. In contrast, TTL provides more expressivity by allowing explicit references to different temporally ordered sequences of states (traces) in dynamic properties. For example, this can be useful for expressing the property of trust monotonicity:

'For any two traces γ1 and γ2, if at each time point t agent A's experience with public transportation in γ2 at t is at least as good as A's experience with public transportation in γ1 at t, then in trace γ2 at each point in time t, A's trust is at least as high as A's trust at t in trace γ1'.

∀γ1, γ2
[∀t, ∀v1:VALUE [state(γ1, t) ⊨ has_value(experience, v1) &
[∀v2:VALUE state(γ2, t) ⊨ [has_value(experience, v2) → v1≤ v2]]] ⇒
[∀t, ∀w1:VALUE [state(γ1, t) ⊨ has_value(trust, w1) &
[∀w2:VALUE state(γ2, t) ⊨ [has_value(trust, w2) → w1≤ w2]]]]]

Other examples of such properties, where different histories are compared are given in Section 4.2 above on trace conditioning.

In contrast to the event calculus, TTL does not employ the mechanism of events that initiate and terminate fluents. Events in TTL are considered to be functions of the external world that can change states of components, according to specified properties of a system. Furthermore, similarly to the situation calculus, also in the event calculus only one time line is considered.

TTL can also be related to temporal languages that are often used for verification (e.g., propositional temporal logic (PTL) and linear-time logic (LTL) [3, 12, 14]). The general idea of translation of a LTL formula into a TTL expression is rather straightforward: by replacing the temporal operators of LTL by quantifiers over time. E.g., the following LTL formula

G(observation_result(itsraining) → *F*(belief(itsraining)))

where the temporal operator *G* means 'for all later time points', and *F* 'for some later time point' is translated into the following TTL expression:

∀t1 [state(γ, t1) ⊨ observation_result(itsraining) ⇒
∃t2 > t1 state(γ, t2) ⊨ belief(itsraining)]

Note that the translation is not bi-directional, i.e., it is not always possible to translate TTL expressions into LTL expressions. An example of a TTL expression that cannot be translated into LTL is again the property of trust monotonicity.

Furthermore, TTL also allows expressivity provided by different extensions of PTL. In particular, the extended temporal logic (ETL) [25] provides a possibility to express any property definable by a regular expression on sequences of states, which cannot be expressed in PTL. Due to the fact that the syntax of TTL provides quantifiers, predicates, and arithmetic functions, such properties can be also expressed in TTL. For example, the property "a given proposition p has to be true in every

even state of a sequence" can be expressed in TTL as follows: ∀t state(γ, 2•t) ⊨ p.

To support the formal specification and analysis of dynamic properties in TTL, special software tools (the Property Editor and the Checker Tool) have been developed. The Property Editor has an intuitive graphical interface for building and editing TTL properties, and the Checker Tool employs an efficient algorithm for the formal verification of properties against a limited set of traces. Although this form of checking is not as exhaustive as model checking (which essentially means checking properties on the set of all traces generated by model execution), in return, it allows more expressivity in specifying properties.

The TTL environment has been tested and proved its value in a number of projects within different domains; e.g., [5, 7, 8, 9, 10]). During this work, the TTL environment has been further developed to provide automated support.

References

[1] Andreka, H., Nemeti, I., and van Benthem, J. (1998). Modal Languages and Bounded Fragments of Predicate Logic. Journal of Philosophical Logic, 27(3): 217-274, 1998.

[2] Barringer, H., M. Fisher, D. Gabbay, R. Owens, & M. Reynolds (1996). The Imperative Future: Principles of Executable Temporal Logic, Research Studies Press Ltd. and John Wiley & Sons.

[3] Benthem, J.F.A.K., van (1983). The Logic of Time: A Model-theoretic Investigation into the Varieties of Temporal Ontology and Temporal Discourse, Reidel, Dordrecht.

[4] Bonabeau, J. Dorigo, M. and Theraulaz, G. (1999). Swarm Intelligence: From Natural to Artificial Systems. Oxford University Press, New York.

[5] Bosse, T., Jonker, C.M., Los, S.A., Torre, L. van der, and Treur, J. (2005). Formalisation and Analysis of the Temporal Dynamics of Conditioning. In: Mueller, J.P. and Zambonelli, F. (eds.), *Proceedings of the Sixth International Workshop on Agent-Oriented Software Engineering, AOSE'05*, pp. 157-168.

[6] Bosse, T., Jonker, C.M., Meij, L. van der, and Treur, J. (2005). LEADSTO: a Language and Environment for Analysis of Dynamics by SimulaTiOn. In: Eymann, T., et al. (eds.), *Proc. of the Third German Conference on Multi-Agent System Technologies, MATES'05*. LNAI, vol. 3550. Springer Verlag, pp. 165-178

[7] Bosse, T., Jonker, C.M., Schut, M.C., and Treur, J. (2004). Simulation and Analysis of Shared Extended Mind. *Simulation Journal* (Transactions of the Society for Modelling and Simulation), vol. 81, 2005, pp. 719 - 732.

[8] Bosse, T., Jonker, C.M., and Treur, J. (2006). An Integrative Modelling Approach for Simulation and Analysis of

Adaptive Agents. In: *Proc. of the 39th Annual Simulation Symposium.* IEEE Computer Society Press, pp. 312-319.

[9] Bosse, T., Jonker, C.M., and Treur, J. (2005). Representational Content and the Reciprocal Interplay of Agent and Environment. In: Leite, J., Omincini, A., Torroni, P., and Yolum, P. (eds.), *Proc. of the Second Int. Workshop on Declarative Agent Languages and Technologies, DALT'04*. LNAI, vol. 3476. Springer Verlag, pp. 270-288.

[10] Bosse, T., Jonker, C.M., and Treur, J. (2006). Formalization and Analyisis of Reasoning by Assumption. *Cognitive Science Journal*, vol. 30, issue 1, pp. 147-180.

[11] Broy, M., and Jahnichen, S. (1995). KORSO: Methods, Languages, and Tools for the Construction of Correct Software - Final Report. LNCS, vol. 1009. Springer Verlag.

[12] Clarke, E.M., Grumberg, O., and Peled, D.A. (2000). Model Checking. MIT Press.

[13] Gleitman, H. (1999). *Psychology*. W.W. Norton & Company, New York.

[14] Goldblatt, R. (1992). Logics of Time and Computation, 2nd edition, CSLI Lecture Notes 7.

[15] Kim, J. (1996). *Philosophy of Mind*. Westview Press.

[16] Kowalski, R., and Sergot, M. (1986). A logic-based calculus of events, New Generation Computing, 4: 67-95.

[17] Los, S.A. and Heuvel, C.E, van den. (2001). Intentional and Unintentional Contributions to Nonspecific Preparation During Reaction Time Foreperiods. *Journal of Experimental Psychology: Human Perception and Performance*, vol. 27, pp. 370-386.

[18] Machado, A. (1997). Learning the Temporal Dynamics of Behaviour. *Psychological Review*, vol. 104, pp. 241-265.

[19] Manna, Z., and Pnueli, A. (1995). Temporal Verification of Reactive Systems: Safety. Springer Verlag.

[20] Pearson, C.E. (1986). Numerical Methods in Engineering and Science. CRC Press.

[21] Port, R.F., Gelder, T. van (eds.) (1995). Mind as Motion: Explorations in the Dynamics of Cognition. MIT Press, Cambridge, Mass.

[22] Reiter, R. (2001). *Knowledge in Action: Logical Foundations for Specifying and Implementing Dynamical Systems*, Cambridge MA: MIT Press.

[23] Sharpanskykh, A. and Treur, J. (2005). Verifying Interlevel Relations within Multi-Agent Systems: Formal Theoretical Basis, Technical Report TR-1701AI. VU Amsterdam, 2005. http://hdl.handle.net/1871/9777

[24] Stirling, C. (2001). Modal and Temporal Properties of Processes. Springer Verlag.

[25] Wolper, P. (1983). Temporal logic can be more expressive. *Information and Control*, vol. 56(1-2), pp. 72-99.

COMPUTER SOCIETY