

A Temporal Trace Language for the Formal Analysis of Dynamic Properties

Tracking number: 234

Word count: 5548

Abstract. Within many domains dynamics are crucial. Current approaches to analyse dynamics, often based on differential equations or modal temporal logics, are not always successful. As an alternative, this paper presents the sorted predicate logical Temporal Trace Language (TTL) for the formal specification and analysis of dynamic properties. This language supports specification of both qualitative and quantitative aspects, and subsumes languages based on differential equations. TTL has a high expressivity and the possibility to define sublanguages for simulation and verification of entailment relations. A software environment for formal verification of properties against a set of traces has been developed. TTL proved its value in a number of different domains.

1 INTRODUCTION

Modelling dynamics poses real challenges for modellers in other disciplines, e.g., biologists and cognitive scientists. Currently, differential equations are most often used to address this challenge, with limited 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, which can compromise the feasibility of the model. Likewise, in the area of Cognitive Science it is advocated to take the Dynamical Systems Theory (DST, see e.g., [19]), as a point of departure. However, DST often only addresses lower-level cognitive processes such as sensory or motor processing. Areas for which the (quantitative) DST-approach is less suitable are higher-level processes with a qualitative character, as found in reasoning, complex task performance, and language processing. As illustrated, within several disciplines it is felt that more abstract modelling techniques are required. To this end, this paper introduces the Temporal Trace Language (TTL) for the analysis of dynamic properties within complex domains.

Desiderata for dynamic modelling approaches are nontrivial: (1) modelling at the right level of abstraction, (2) expressivity for logical relationships, (3) expressivity for quantitative relationships, (4) both discrete and continuous modelling, (5) difference and differential equations should be subsumed, and (6) expressivity for dynamics of varying complexity, e.g., adaptivity. Moreover, analysis techniques that would be desirable concern 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 an ordered sequence of states of a system. Each state is characterised by a number of state properties that hold. *Desiderata for analysis techniques* include: (a) generating traces by simulation based on continuous variables, (b) generating traces by simulation based on qualitative, logical notions, (c) formalisation of simulated or empirical traces, (d) analysis of properties of simulated traces, (e) analysis of properties of empirical traces, and (f) analysis of relationships between (e.g., global and local) properties. Taken

together, these desiderata are not easy to fulfill. Sometimes they are mutually exclusive. On the one hand, high expressivity is desired, but on the other hand feasible analysis techniques are demanded. To make automated support for analyses feasible, often the expressivity of the modelling language is limited, thereby compromising on the first list of desiderata. For example, the expressivity is limited to difference and differential equations as in DST (excluding logical relationships, compromising at least (2)), or to propositional modal temporal logics (excluding numerical relationships, compromising at least (3), (5), (6)). In the former case calculus can be exploited to do simulation and analysis [19], fulfilling (a) and (c) but not (b), (d), (e) and (f). In the latter case, for example, simulation can be based on a specific executable format (e.g., executable temporal logic [1], fulfilling (b) and (c) but not (a), (d), (e) and (f)), and model checking techniques can be exploited for analysis of relationships between dynamic properties, fulfilling (d) to (f), e.g., [8, 15, 21].

Within the literature on analysis of properties (verification), much emphasis is put on computation of entailment relations. This essentially means checking properties on the set of all theoretically possible traces of a process. To make that feasible, expressivity of the language for these properties has to be sacrificed to a large extent. However, checking properties on a practically given set of traces (instead of all theoretically possible ones) is computationally much cheaper, and therefore the language for these properties can be more expressive. Such a set can consist of one or a number of traces, obtained empirically or by simulation. By limiting the desiderata by giving up (f), but still keeping (c) to (e), a much more expressive language for properties can be dealt with; the sorted predicate logic temporal trace language TTL described here is an example of this.

For simulation it is essential to have limitations to the language. Therefore, an *executable language* can be defined as a sublanguage of the *overall language* for analysis. Moreover, also *analysis languages* that allow analysis in the sense of (f) can be embedded in the overall language. Thus the following situation is obtained. At the top level there is an expressive overall language, in our case TTL, which fulfills all of the desiderata for modelling languages, i.e., (1) to (6). Concerning the desiderata for analysis techniques, it fulfills (c) to (e), but sacrifices (a), (b) and (f). In addition, a sublanguage can be defined for execution (fulfilling (1) to (5) and (a) and (b)), and a sublanguage can be defined for analysis of relationships between properties in the sense of (f), thereby also fulfilling (1) and (2). For the case of TTL, one of the executable sublanguages that already exist is the LEADSTO language, cf. [3]. Moreover, for the sublanguage for analysis one could think of any standard temporal logic, such as LTL or CTL, see, e.g., [2, 10].

Having the language for simulation and the languages for analysis within one subsuming language provides the possibility to have a declarative specification of a simulation model, and thus to involve a simulation model in logical analyses.

Section 2 describes the syntax and the semantics of the TTL language. In Section 3, it is shown how dynamic properties that are expressed in related languages can be translated into TTL. Section 4 provides application examples of TTL. Section 5 describes the TTL Checker tool that supports the formal specification and analysis of dynamic properties in TTL in detail. Section 6 is a conclusion.

2 SYNTAX AND SEMANTICS OF TTL

The language TTL is a variant of an order-sorted predicate logic [16]. Whereas standard multi-sorted predicate logic is a language to reason about static properties only, TTL is an extension of such language with facilities for reasoning about the dynamic properties of arbitrary systems expressed by static languages.

For specifying state properties for system components, ontologies 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 an ontology. For every system component A several types of ontologies can be distinguished that are used for specifying state properties of different types. That is, the ontologies $\text{IntOnt}(A)$, $\text{InOnt}(A)$, $\text{OutOnt}(A)$, and $\text{ExtOnt}(A)$ are used for expressing respectively internal, input, output and external state properties of the component A . For example, a state property expressed as a predicate pain may belong to $\text{IntOnt}(A)$, whereas the atom $\text{has_temperature}(\text{environment}, 7)$ may belong to $\text{ExtOnt}(A)$. Often in cognitive science input ontologies contain elements for describing perceptions of an agent from the external world (e.g. $\text{observed}(a)$ means that a component has an observation of state property a), whereas output ontologies are used for describing actions of agents within the external world (e.g., $\text{performing_action}(b)$ represents action b performed by a component in its environment).

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

In TTL, formulae of the state language are used as objects. To provide names of object language formulae ϕ in TTL, the operator $(*)$ is used (written as ϕ^*), which maps variable sets, term sets and formula sets of the state language to the elements of sorts S^{GTERMS} , S^{TERMS} , S^{VARS} and STATPROP in the following way:

- (1) Each constant symbol c from the state sort S is mapped to the constant name c' of sort S^{GTERMS} .
- (2) Each variable $x: S$ from the state language is mapped to the constant name $x' \in S^{\text{VARS}}$.
- (3) Each function symbol $f: S_1 \times S_2 \times \dots \times S_n \rightarrow S_{n+1}$ from the state language is mapped to the function name $f: S_1^{\text{TERMS}} \times S_2^{\text{TERMS}} \times \dots \times S_n^{\text{TERMS}} \rightarrow S_{n+1}^{\text{TERMS}}$.
- (4) Each predicate symbol $P: S_1 \times S_2 \times \dots \times S_n$ is mapped to the function name $P: S_1^{\text{TERMS}} \times S_2^{\text{TERMS}} \times \dots \times S_n^{\text{TERMS}} \rightarrow \text{STATPROP}$.
- (5) The mappings for state formulae are defined as follows:

- a. $(\neg\phi)^* = \text{not}(\phi^*)$
- b. $(\phi \ \& \ \psi)^* = \phi^* \wedge \psi^*$, $(\phi \ | \ \psi)^* = \phi^* \vee \psi^*$
- c. $(\phi \ \rightarrow \ \psi)^* = \phi^* \rightarrow \psi^*$, $(\phi \ \leftrightarrow \ \psi)^* = \phi^* \leftrightarrow \psi^*$
- d. $(\forall x \ \phi(x))^* = \forall x' \ \phi^*(x')$, where x is variable over sort S and x' is any constant of S^{VARS} , the same for \exists

It is assumed that the state language and the TTL define disjoint sets of expressions. Therefore, further in TTL formulae we shall use the same notations for the elements of the object language (i.e. constants, variables, functions, predicates) and for their names in the TTL without introducing any ambiguity. Moreover we shall use t with subscripts and superscripts for variables of the sort TIME ; and γ with subscripts and superscripts for variables of the sort TRACE .

A state is described by a function symbol $\text{state}: \text{TRACE} \times \text{TIME} \rightarrow \text{STATE}$. A trace is a temporally ordered sequence of states. A time frame is assumed to be fixed, linearly ordered, for example, the natural or real numbers. Such an interpretation of a trace contrasts to Mazurkiewicz traces [17] that are frequently used for analysing behaviour of Petri nets. Mazurkiewicz traces represent restricted partial orders over algebraic structures with a trace equivalence relation. Furthermore, as opposed to some interpretations of traces in the area of software engineering [12], a formal logical language is used here to specify properties of traces.

The set of function symbols of TTL includes $\wedge, \vee, \rightarrow, \leftrightarrow, \text{STATPROP} \times \text{STATPROP} \rightarrow \text{STATPROP}$; $\text{not}: \text{STATPROP} \rightarrow \text{STATPROP}$, and $\forall, \exists: S^{\text{VARS}} \times \text{STATPROP} \rightarrow \text{STATPROP}$, of which the counterparts in the state language are boolean propositional connectives and quantifiers. Further we shall use $\wedge, \vee, \rightarrow, \leftrightarrow$ in infix notation and \forall, \exists in prefix notation for better readability. For example, using such function symbols the state property about external world expressing that there is no rain and no clouds can be specified as: $\text{not}(\text{rain}) \wedge \text{not}(\text{clouds})$.

For formalising relations between sorts VALUE and TIME , functional symbols $-, +, /, \bullet: \text{TIME} \times \text{VALUE} \rightarrow \text{TIME}$ are introduced. Furthermore, for performing arithmetical operations on the sort VALUE the corresponding arithmetical functions are included.

States are related to state properties via the satisfaction relation denoted by the infix predicate \models (also denoted by the prefix predicate holds): $\text{state}(\gamma, t) \models p$ (or $\text{holds}(\text{state}(\gamma, t), p)$), which denotes that state property p holds in trace γ at time point t .

Both $\text{state}(\gamma, t)$ and p are terms of the TTL language. In general, TTL terms are constructed by induction in a standard way from variables, constants and function symbols typed with all before-mentioned TTL sorts.

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

- (1) If v_1 is a term of sort STATE , and u_1 is a term of the sort STATPROP , then $\text{holds}(v_1, u_1)$ 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) If v_1, v_2 are terms of sort VALUE , then $v_1 < v_2$ is an atomic TTL formula.

The set of *well-formed TTL-formulae* is defined inductively in a standard way using boolean connectives and quantifiers over variables of TTL sorts. An example of the TTL formula, which describes observational belief creation of an agent, is given below:

'In any trace, if at any point in time t_1 the agent A observes that it is raining, then there exists a point in time t_2 after t_1 such that at t_2 in the trace the agent A believes that it is raining'.

$$\forall \gamma \forall t_1 [\text{state}(\gamma, t_1) \models \text{observation_result}(\text{itsraining}) \Rightarrow \exists t_2 > t_1 \ \text{state}(\gamma, t_2) \models \text{belief}(\text{itsraining})]$$

An *interpretation* of a TTL formula is the standard interpretation of an order sorted predicate logic formula and is defined by a mapping \mathcal{I} that associates each:

- (1) sort symbol S to a certain set (subdomain) D_S , such that if $S \subseteq S'$ then $D_S \subseteq D_{S'}$
- (2) constant c of sort S to some element of D_S
- (3) function symbol f of type $\langle X_1, \dots, X_n \rangle \rightarrow X_{n+1}$ to a mapping: $\mathcal{I}(X_1) \times \dots \times \mathcal{I}(X_n) \rightarrow \mathcal{I}(X_{n+1})$
- (4) predicate symbol P of type $\langle X_1, \dots, X_n \rangle$ to a relation on $\mathcal{I}(X_1) \times \dots \times \mathcal{I}(X_n)$

A *model* M for the TTL is a pair $M = \langle \mathcal{I}, V \rangle$, where \mathcal{I} is an interpretation function, and V is a variable assignment, mapping each variable x of any sort S to an element of D_S . We write $V[x/v]$ for the assignment that maps variables y other than x to $V(y)$ and maps x to v . Analogously, we write $M[x/v] = \langle \mathcal{I}, V[x/v] \rangle$.

If $M = \langle \mathcal{I}, V \rangle$ is a model of the TTL, then *the interpretation of a TTL term* τ , 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) $(c)^M = \mathcal{I}(c)$, where c is a constant of one of the TTL sorts.
- (3) $f(\tau_1, \dots, \tau_n)^M = \mathcal{I}(f)(\tau_1^M, \dots, \tau_n^M)$, where f is a TTL function of type $S_1 \times \dots \times S_n \rightarrow S$ and τ_1, \dots, τ_n are terms of TTL sorts S_1, \dots, S_n .

The truth definition of TTL for the model $M = \langle \mathcal{I}, V \rangle$ is inductively defined by:

- (1) $\models_M P_i(\tau_1, \dots, \tau_k)$ iff $\mathcal{I}(P_i)(\tau_1^M, \dots, \tau_k^M) = true$
 - (2) $\models_M \neg \phi$ iff $\not\models_M \phi$
 - (3) $\models_M \phi \wedge \psi$ iff $\models_M \phi$ and $\models_M \psi$
 - (4) $\models_M \forall x(\phi(x))$ iff $\models_{M[x/v]} \phi(x)$ 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.

A number of important properties of TTL are formulated in form of axioms:

- (1) Equality of traces:
 $\forall \gamma_1, \gamma_2 [\forall t [\text{state}(\gamma_1, t) = \text{state}(\gamma_2, t)] \Rightarrow \gamma_1 = \gamma_2]$
- (2) Equality of states:
 $\forall s_1, s_2 [\forall a: \text{STATPROP} [\text{truth_value}(s_1, a) = \text{truth_value}(s_2, a)] \Rightarrow s_1 = s_2]$
- (3) Truth value in a state:
 $\text{holds}(s, p) \Leftrightarrow \text{truth_value}(s, p) = true$
- (4) State consistency axiom:
 $\forall \gamma, t, p (\text{holds}(\text{state}(\gamma, t), p) \Rightarrow \neg \text{holds}(\text{state}(\gamma, t), \text{not}(p)))$
- (5) State property semantics:
 - a. $\text{holds}(s, (p_1 \wedge p_2)) \Leftrightarrow \text{holds}(s, p_1) \wedge \text{holds}(s, p_2)$
 - b. $\text{holds}(s, (p_1 \vee p_2)) \Leftrightarrow \text{holds}(s, p_1) \vee \text{holds}(s, p_2)$
 - c. $\text{holds}(s, \text{not}(p_1)) \Leftrightarrow \neg \text{holds}(s, p_1)$

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

$\text{holds}(s, (\exists(x, F))) \Leftrightarrow \exists x': S^{\text{GTERMS}} \text{holds}(s, G)$, and $\text{holds}(s, (\forall(x, F))) \Leftrightarrow \forall x': S^{\text{GTERMS}} \text{holds}(s, G)$ 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. $\forall t \leq t$ (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.-c. The same as for the sort TIME
 - 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) (Optional) Finite variability property (for any trace γ):
 $\forall t_0, t_1 t_0 < t_1 \Rightarrow \exists \delta > 0 [\forall t [t_0 \leq t \wedge t \leq t_1] \Rightarrow \exists t_2 \leq t \forall t_3 [t_2 \leq t_3 \wedge t_3 \leq t_2 + \delta] \Rightarrow \text{state}(\gamma, t_3) = \text{state}(\gamma, t)]$

3 EXPRESSING OTHER LANGUAGES

In this section, it is shown how a number of existing languages can be expressed in TTL. To show how modelling techniques used in the Dynamical Systems approach (DST) [19] can be represented in TTL, in particular the discrete case is considered. An example is the use of logistic difference equations to model growth of various cognitive phenomena, e.g., the growth of a child's lexicon between 10 and 17 months, cf. [9]. The logistic difference equation used is $L(n+1) = L(n) (1 + r - r L(n)/K)$. Here r is the growth rate and K the carrying capacity. This equation can be expressed in TTL using a constant L of state sort STATPARAM and constants r, K and d that belong to state sort LVALUE (an ordered set of numbers with the arithmetic functions defined as for the TTL sort VALUE):

$\forall \gamma \forall t \forall v: \text{LVALUE}^{\text{GTERMS}}$

$\text{state}(\gamma, t) \models \text{has_value}(L, v) \Rightarrow$

$\text{state}(\gamma, t+d) \models \text{has_value}(L, v \bullet (1 + r \bullet d - r \bullet d \bullet v/K))$

The traces γ satisfying the above dynamic property are the solutions of the difference equation. In a similar manner, it is even possible to address the continuous case, i.e., to express differential equations in TTL. Due to space limitations, this case is not addressed here. See [13] for a detailed explanation.

Executable languages can be defined as sublanguages of TTL. An example of such a language, which was designed for simulation of dynamics in terms of both qualitative and quantitative concepts, is the LEADSTO language, cf. [3]. The LEADSTO language models direct temporal dependencies between two state properties in states at different points in time. A specification of dynamic properties in LEADSTO format has as advantages that it is executable and that it can often easily be depicted graphically. The format of LEADSTO is defined as follows. Let α and β be state properties of the form 'conjunction of atoms or negations of atoms', and e, f, g, h non-negative real numbers (constants of sort VALUE). In LEADSTO the notation $\alpha \Rightarrow_{e, f, g, h} \beta$, means:

If state property α holds for a certain time interval with duration g , then after some delay (between e and f) state property β will hold for a certain time interval of length h .

In terms of TTL, the fact that the above statement holds for a trace γ is expressed as follows:

$\forall t [\forall t' [t_1 - g \leq t' \wedge t' < t_1 \Rightarrow \text{state}(\gamma, t') \models \alpha] \Rightarrow$

$\exists d: \text{VALUE} [e \leq d \wedge d \leq f \wedge \forall t' [t_1 + d \leq t' \wedge t' < t_1 + d + h \Rightarrow \text{state}(\gamma, t') \models \beta]]$

Besides executable languages, also languages often used for the verification of entailment relations can be defined as sublanguages of TTL. Examples of such languages are LTL and CTL, see, e.g., [2, 10]. It is briefly shown how dynamic properties expressed as formulae in LTL can be translated to TTL. The general idea is rather straightforward: by replacing the temporal operators of LTL by quantifiers over time. E.g., consider the following LTL formula:

$G(\text{observation_result}(\text{itsraining}) \rightarrow F(\text{belief}(\text{itsraining})))$

where the temporal operator G means 'for all later time points', and F 'for some later time point'. The first operator can be translated into a universal quantifier, whereas the second one can be translated into an existential quantifier. Using TTL, this formula then can be expressed, for example, as follows:

$\forall t_1 [\text{state}(\gamma, t_1) \models \text{observation_result}(\text{itsraining}) \Rightarrow$

$\exists t_2 > t_1 \text{state}(\gamma, t_2) \models \text{belief}(\text{itsraining})]$

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 the property 'Trust Monotonicity': the more positive your experiences, the higher your trust. This property cannot be expressed in LTL since it involves the comparison of two different traces. This shows that LTL can be considered a proper sublanguage of TTL, i.e., a sublanguage not equal to TTL. Similar

processes are often based on differential equations. However, for a number of applications these approaches have serious limitations. For example, within Cognitive Science, approaches based on differential equations are not particularly suitable to model higher-level processes with mainly a qualitative character. To deal with these limitations, this paper presents the predicate logical Temporal Trace Language (TTL) for the formal specification and analysis of dynamic properties. 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 the possibility of 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 [3]. For the verification of entailment relations, standard temporal languages such as LTL and CTL (e.g., [2, 10]) can be defined as sublanguages of TTL.

TTL has some similarities with the situation calculus [20] and the event calculus [14], 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 [18], 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 γ_1 and γ_2 , if at each time point t agent A 's experience with public transportation in γ_2 at t is at least as good as A 's experience with public transportation in γ_1 at t , then in trace γ_2 at each point in time t , A 's trust is at least as high as A 's trust at t in trace γ_1 .

$$\forall \gamma_1, \gamma_2$$

$$[\forall t, \forall v_1: \text{VALUE} \ [\text{state}(\gamma_1, t) \models \text{has_value}(\text{experience}, v_1) \ \& \ \forall v_2: \text{VALUE} \ \text{state}(\gamma_2, t) \models [\text{has_value}(\text{experience}, v_2) \rightarrow v_1 \leq v_2]] \Rightarrow$$

$$[\forall t, \forall w_1: \text{VALUE} \ [\text{state}(\gamma_1, t) \models \text{has_value}(\text{trust}, w_1) \ \& \ \forall w_2: \text{VALUE} \ \text{state}(\gamma_2, t) \models [\text{has_value}(\text{trust}, w_2) \rightarrow w_1 \leq w_2]]]]]$$

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.

To support the formal specification and analysis of dynamic properties in TTL, a special software environment (the TTL Checker) has been developed. The TTL Checker has an intuitive graphical interface for building and editing TTL properties, and employs an efficient algorithm for the formal verification of properties against a set of traces (for example obtained from experiments or simulation). Although this form of checking is not as exhaustive as model checking (which essentially means checking properties on the set of all theoretically possible traces), in return, this makes it possible to specify more expressive properties.

To conclude, the approach proved its value in a number of research projects in different domains. It has been used to analyse complex processes in cognitive science (e.g., human reasoning [7]), biology (e.g., the dynamics of the heart [6]), social science (e.g., organisational change [11]), and artificial intelligence (e.g., design processes [5]).

References

1. 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.
2. 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.
3. 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 MATES'05*. LNAI, vol. 3550. Springer Verlag, pp. 165-178
4. Bosse, T., Jonker, C.M., and Treur, J. (2006). An Integrative Modelling Approach for Simulation and Analysis of Adaptive Agents. In: *Proceedings of the 39th Annual Simulation Symposium*. IEEE Computer Society Press.
5. Bosse, T., Jonker, C.M., and Treur, J. (2004). Analysis of Design Process Dynamics. In: R. Lopez de Mantaras, L. Saitta (eds.), *Proceedings of the 16th European Conference on Artificial Intelligence, ECAI'04*, pp. 293-297.
6. Bosse, T., Jonker, C.M., and Treur, J. (2004). Organisation Modelling for the Dynamics of Complex Biological Processes. In: G. Lindemann, D. Moldt, M. Paolucci, B. Yu (eds.), *Proc. of RASTA'02*. LNAI, vol. 2934. Springer Verlag, pp. 92-112.
7. Bosse, T., Jonker, C.M., and Treur, J. (2005). Reasoning by Assumption: Formalisation and Analysis of Human Reasoning Traces. *Cognitive Science Journal*. In press.
8. Clarke, E.M., Grumberg, O., and Peled, D.A. (2000). *Model Checking*. MIT Press.
9. Geert, P. van (1995). Growth Dynamics in Development. In: (Port and van Gelder, 1995), pp. 101-120.
10. Goldblatt, R. (1992). *Logics of Time and Computation*, 2nd edition, CSLI Lecture Notes 7.
11. Hoogendoorn, M., Jonker, C.M., Schut, M., and Treur, J. (2004). Modelling the Organisation of Organisational Change. In: Giorgini, P., and Winikoff, M., (eds.), *Proc. of the Sixth Int. Workshop on Agent-Oriented Information Systems (AOIS'04)*, 2004, pp. 29-46.
12. Iglewski, M., and Mincer-Daszkiwicz, J. (1997). Internal design of modules specified in the trace assertion method. *Science of Computer Programming*, 28, pp. 139-170.
13. Jonker, C.M., and Treur, J. (2003). A Temporal-Interactivist Perspective on the Dynamics of Mental States. *Cognitive Systems Research Journal*, vol. 4, 2003, pp. 137-155
14. Kowalski, R. and Sergot, M.A. (1986). A logic-based calculus of events, *New Generation Computing*, 4, pp. 67-95.
15. Manna, Z., and Pnueli, A. (1995). *Temporal Verification of Reactive Systems: Safety*. Springer Verlag.
16. Manzano, M. (1996). *Extensions of First Order Logic*, Cambridge University Press.
17. Mazurkiewicz, A. (1987). Trace Theory. In: *Advances in Petri nets II: applications and relationships to other models of concurrency*. Springer LNCS, vol. 255, pp. 279-324.
18. Pinto, J. and Reiter, R. (1995). Reasoning About Time in the Situation Calculus. *Ann. Math. Artif. Intell.*, 14(2-4), pp. 251-268.
19. Port, R.F., Gelder, T. van (eds.) (1995). *Mind as Motion: Explorations in the Dynamics of Cognition*. MIT Press, Cambridge, Mass.
20. Reiter, R. (2001). *Knowledge in Action: Logical Foundations for Specifying and Implementing Dynamical Systems*, Cambridge MA: MIT Press.
21. Stirling, C. (2001). *Modal and Temporal Properties of Processes*. Springer Verlag.