# A SOC-Based Formal Specification and Verification of Hybrid Systems

Ning Yu(✉) and Martin Wirsing

Programmierung und Softwaretechnik, Institut für informatik,
Ludwig-Maximilians-Universität München, Oettingenstrasse 67,
80538 Munich, Germany
{yu,wirsing}@pst.ifi.lmu.de

**Abstract.** In order to specify hybrid systems in a SOC paradigm, we define Hybrid Doubly Labeled Transition Systems and the hybrid trace of it. Then we extend SRML notations with a set of differential equation-based expressions and hybrid programs and interpret the notations over Hybrid Doubly Labeled Transition Systems. By redefining the dynamic temporal logic dTL, we provide a logic basis for reasoning about the behavior of hybrid transition systems. We illustrate our approach by a case study about the control of a moving train, in which the movement of the train is regulated by ordinary differential equations.

**Keywords:** Hybrid transition systems · SRML · Differential equations · dTL

## 1 Introduction

Service-Oriented Computing (SOC) is a computing paradigm that utilizes services as fundamental elements to support rapid, low-cost development of distributed applications in heterogeneous environments [1]. In SOC, a service is defined as an independent and autonomous piece of functionality which can be described, published, discovered and used in a uniform way. Within the development of SOC, complex systems are more and more involved. A typical type of complex systems are the hybrid systems, which arise in embedded control where discrete components are coupled with continuous components. In an abstract point of view, hybrid systems are mixtures of real-time (continuous) dynamics and discrete events [2]. In order to address these two aspects into SOC paradigm, we make our approach by giving a SOC-based formal specification and verification to hybrid systems.

The SOC-based specification of hybrid systems are realized by giving a hybrid extension to the SENSORIA Reference Modeling Language (SRML). SRML is a modeling language that can address the higher levels of abstraction of "business modeling"[3], developed in the project SENSORIA – the IST-FET Integrated Project that develops methodologies and tools such as Web Services [10] for dealing with the challenges arose in Service-Oriented Computing. To make this approach, we first define: Hybrid Doubly Labeled Transition System (HL$^2$TSs), which extends the semantic domain of UCTL [11]; hybrid traces of HL$^2$TSs, which represent traces of the system evolution; and service-oriented Hybrid Doubly Labeled Transition Systems (SO-HL$^2$TSs), which extends HL$^2$TSs, as the SRML semantic domain. Then we extend SRML by extending *the language of business role* and *the language of business protocol*. The language of business role is extended by defining formulas and differential equation-based terms for *transition specifications*, and interpreting them over SO-HL$^2$TSs. The language of business protocol is extended by redefining *hybrid programs* and formulas of the dynamic logic temporal logic dTL [12], which provides modalities for quantifying over traces of hybrid systems, for *behaviour constraints*.

We illustrate our approach though a case study of a *Train-Control system* verification. The Train-Control system abstracts the European Train Control System (ETCS)[19], which is a a signalling, control and train protection system designed to replace the many incompatible safety systems currently used by European railways. In such a system the displacement of the train is continuous on time within the system evolution and is governed by ordinary differential equations. On specifying the system with extended SRML, we verify a safety constraint of it with a set of sequent calculus provided in [12] for verifying hybrid systems.

## 2 A General Introduction to SRML

In this section we give an overview of SRML composition model and each element of the composition introduced in [5].

SRML is designed for modeling composite services, whose business logic involves a number of interactions among more elementary service components within services and among different services via interfaces. This idea comes from the concepts proposed in Service Component Architectures (SCA)[4]. The basic units of business logic are called *service modules*, which are composed of *service components* and *external interfaces*, and are orchestrated by control and data flows. Service components are computational units central to the composite services. Each service component is modeled by means of the execution pattern that involves a set of interactions and orchestrations related to them. In a service module, external interfaces are used for modeling external parties that either provide services to or require services from this module. Each interface specifies a set of interactions internal to this module and some constraints to which the module expect the external parties to adhere. Service components and external interfaces of the same module are connected to each other through internal *wires* that are used to support and coordinate the interactions among them. Figure 1 shows an example of a service module.

The orchestrations of services components can be seen static, since they are pre-define at design time and do not invoke services of any external party. While the constraints of the external interfaces are dynamically configured at each run time, when modules are discovered and bound to different external parties.

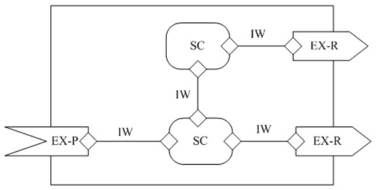


**Fig. 1.** Service composition

In this paper, we only discuss the way of defining the module, but not the runtime configuration. Next we show in detail the composition of a service module.

**Business Role.** Service components are specified through business roles, each of which is specified by declaring a set of interactions and the way they are orchestrated. We give the following introduction to each part:

***Interactions*** involve two parties and can be in both directions. They are defined from the point of view of the party in which they are defined. ***Local*** specifies the variables that provide an abstract view of the state of the local component.

***Initialization*** designates a specific initial state.

***Transitions*** model the activities performed by the component. A transition has an optional name and some possible features. These features are classified as follows: (i) A *trigger* is a condition that specifies the occurrence of an event or a state condition; (ii) A *guard* is a condition that identifies the states in which the transition can take place; (iii) Each sentence in *effects* specifies the effects of the transition in the local state.

**Business Protocol.** External interfaces are specified through business protocols. They declare similar interactions to those in business roles, but from the external parties' point of view. Instead of an orchestration, a business protocol provide a set of properties that the external party is expected to follow.

***Behaviour*** models the behaviors that users can expect form a service. Based on temporal logic [14], they specify which message exchange sequences are supported by the service via a number of ***behaviour constraints***.

**Interaction Protocol.** Wires that connect service components and external interfaces are specified through interaction protocols, and are labeled by connectors that coordinate the interactions in which the parties are jointly involved. Our work doesn't relate to this part, so it is not introduced in details.

## 3 Hybrid Extension of SRML

### 3.1 A Hybrid Extension of SRML Semantic Domain

Since SRML is a control/data flow driven modeling language, the following data signature is adopted as a basic semantic domain:

$$\Omega = \langle D, F \rangle \tag{1}$$

where $D$ is a set of data sorts and $F$ is a $D^* \times D$-indexed family of sets of operations over the sorts. We assume that $time \in D$ is a datatype that represents the usual concept of time. And a fixed algebra $\mathcal{U}$ denotes the interpretation of $\Omega$.

SRML is interpreted over Service-oriented Doubly Labeled Transition Systems ($SO\text{-}L^2TSs$), whose structure bases on the UCTL [11] semantic domain – $L^2TS$. In order to extend SRML over the combination of hybrid systems and transition systems, we define Hybrid $L^2TSs$ ($HL^2TS$) by extending $L^2TSs$ with a set of continuous functions $\Sigma$, and define the trace of a $HL^2TS$ to describe the system evolution. Then we define $SO\text{-}HL^2TS$ over which extended SRML could be interpreted.

**Definition 1 (Hybrid Doubly Labeled Transition System).** *A hybrid doubly Labelled Transition System* ($HL^2TS$) is a tuple

$$\langle S, s_0, Act, R, \Sigma, AP, L\rangle$$

*where:*

- *$S$ is a set of states;*
- *$s_0 \in S$ is the initial state;*
- *$Act$ is a finite set of observable actions;*
- *$R \subseteq S \times 2^{Act} \times S$ is the transition relation. A transition $(s, \alpha, s') \in R$ is denoted by $s \xrightarrow{\alpha} s'$;*
- *$\Sigma$ is a set of functions and for every function $\sigma \in \Sigma, \sigma : [0, r_\sigma] \to S$ with $r_\sigma \in \mathbb{R}$ and $r_\sigma \geq 0$, $\sigma$ is continuous on the interval $[0, r_\sigma]$;*
- *$AP$ is a set of atomic propositions;*
- *$L : S \to 2^{AP}$ is a labelling function such that $L(s)$ is the subset of all atomic propositions that are true in state $s$.*

The evolution of a $\mathrm{HL}^2\mathrm{TS}$ is described by *traces*, which represent sequences of pieces of continuous functions and discrete jumps in the $\mathrm{HL}^2\mathrm{TS}$ evolution.

**Definition 2 (Trace).** Let $\langle S, s_0, Act, R, \Sigma, AP, L\rangle$ be a $HL^2TS$ then*:*

- *For every $\sigma \in \Sigma$, $\sigma[0, r_\sigma]$ denotes the trace of infinitely many states $\sigma(0), \ldots,$ $\sigma(r_\sigma)$ along $\sigma$ over the interval $[0, r_\sigma]$;*
- *$\rho$ is a hybrid trace from $s_0$ if $\rho = (s_0 \xrightarrow{\alpha_0} \sigma_1, \sigma_1[0, r_{\sigma_1}], \sigma_1(r_{\sigma_1}) \xrightarrow{\alpha_1} \sigma_2(0),$ $\sigma_2[0, r_{\sigma_2}], \ldots)$ where $(s_0, \alpha_0, \sigma_1(0)) \in R$ and $\sigma_i(r_{\sigma_i}), \alpha_i, \sigma_{i+1}(0)) \in R$ with $i \in \mathbb{N}$;*
- *A position* of $\rho$ is a pair *$(i, \zeta)$* with $i \in \mathbb{N}$ and $\zeta$ in the interval $[0, r_{\sigma_i}]$*;* the state or transition of $\rho$ at $(i, \zeta)$ *is* $\sigma_i(\zeta)$. Positions of $\rho$ are ordered lexicographically by $(i, \zeta) \prec (j, \xi)$ iff either $i < j$, or $i = j$ and $\zeta < \xi$;
- *A trace $\rho$ starting from the initial state $s_0$ terminates if it is a finite sequence ($\rho = (s_0 \xrightarrow{\alpha} \sigma_1, \sigma_1[0, r_{\sigma_1}], \sigma_1(r_{\sigma_1}) \xrightarrow{\alpha_1} \sigma_2(0), \sigma_2[0, r_{\sigma_2}], \ldots, \sigma_n[0, r_{\sigma_n}])$, and the first state of the trace $s_0$ is denoted by $first\rho$, the last state of the trace $\sigma_n(r_{\sigma_n})$ is denoted by $last\rho$;*

– *The concatenation of traces* $\rho_1 = (s_1 \xrightarrow{\alpha_0} \sigma_1(0), \sigma_1[0, r_{\sigma_1}], \ldots)$ *and* $\rho_2 = (s_2 \xrightarrow{\alpha'_0} \varsigma_1(0), \varsigma_1[0, r_{\varsigma_1}], \ldots)$*, denoted by* $\rho_1 \circ \rho_2$*, is defined as follows:*

$$\rho_1 \circ \rho_2 = \begin{cases} (s_0 \xrightarrow{\alpha_0} \sigma_1(0), \ldots, & \text{if } \rho_1 \text{ terminates at } \sigma_n(r_{\sigma_n}) \\ \ \sigma_n[0, r_{\sigma_n}], s'_0 \xrightarrow{\alpha'_0} \varsigma_1(0), \ldots) & \quad \text{and } \sigma_n(r_{\sigma_n}) = s'_0 \\ \rho_1 & \text{if } \rho_1 \text{ does not terminate} \\ \text{not defined} & \text{otherwise} \end{cases}$$

SO-HL$^2$TSs extends HL$^2$TSs and Service-Oriented Transition Systems (SO-TSs). SO-TSs are defined in [6]. By combining these two types of transition systems, we get the semantic domain for SRML extension.

**Definition 3 (Service-Oriented HL$^2$TSs).** *The Service-Oriented HL$^2$TS (denoted SO-HL$^2$TS) that abstracts a SO-TS* $\langle S, \rightarrow, s_0, G\rangle$ *is the tuple*

$$\langle S, s_0, Act, R, \Sigma, AP, L, TIME, \Pi\rangle$$

*where:*

– $\langle S, s_0, Act, R, \Sigma, AP, L\rangle$ *is the corresponding HL$^2$TS;*
– $Act = \{e! : e \in E\} \cup \{e¡ : e \in E\} \cup \{e? : e \in E\} \cup \{e¿ : e \in E\}$*;*
– $R \subseteq S \times 2^{Act} S$ *is such that:*
  - $s \xrightarrow{\alpha} s'$ *iff* $(s, \alpha, s' \in R$ *for some* $\alpha \in Act^2$*;*
  - *For every* $(s), \alpha, s') \in R$*:*

$$\begin{aligned} \alpha = &\{e! : e \in PUB^{s \xrightarrow{\alpha} s'} \cup \{e¡ : e \in ADLV^{s \xrightarrow{\alpha} s'}\} \cup \\ &\{e? : e \in EXC^{s \xrightarrow{\alpha} s'}\} \cup \{e¿ : e \in DSC^{s \xrightarrow{\alpha} s'}\} \end{aligned}$$

– $AP = \{e! : e \in E\} \cup \{e¡ : e \in E\} \cup \{e? : e \in E\} \cup$
  $\{e¿ : e \in E\} \cup \{a.pledge : a \in 2WAY\}$*;*
– $L : S \rightarrow 2^{AP}$ *is such that:*

$$\begin{aligned} L(s) = &\{e! : e \in HST!^s\} \cup \{e¡ : e \in HST¡^s\} \cup \\ &\{e? : e \in HST?^s\} \cup \{e¿ : e \in HST¿^s\} \\ &\cup PLG^s \end{aligned}$$

  *with* $s \in S$*;*
– $TIME$ *assigns to each state* $s \in S$ *the instant* $TIME^s$*;*
– $\Pi$ *assigns to each state* $s \in S$ *the parameter interpretation* $\Pi^s$*.*

In Definition 3, $E$ is the set of all events of a configuration defined in [6]. $a$ is an interaction and *a.pledge* is the pledge that is associated with that interaction in the configuration. $2WAY$ is the set of interactions that take place in both directions in the configuration. $HST!, HST¡, HST?$ and $HST¿$ are subsets of events in a computation state, $PLG$ is the set of pledges that holds in the computation state and $TIME \in time_{\mathcal{U}}$. $PUB, ADLV, EXC$ and $DSC$ are subsets of events in a computation step, where computation state and computation step are used to describe the computation of a configuration and they are also defined in [6].

In the rest of this section, we present the semantics of SRML extension interpreted over *SO-HL$^2$TSs* defined in Sect. 2.2. The SRML extension consists of two parts: the extension of business role and the extension of business protocol. The latter includes an extension of dTL formulas which is used to specify and verify *behaviours* specified by the new extended *behaviour constraint* in business protocol.

In order to define the SRML extension, throughout the remaining of this section we consider:

- $sig = \langle NAME, PARAM \rangle$ (defined in [6]) to be an interaction signature where $Act$ is the set of actions associated with $sig$;
- $VAR$ (defined in [6]) to be an attribute declaration.
- $\Xi = \langle N, W, PLL, \Psi, 2WAY, 1WAY \rangle$ (defined in [6]) to be a configuration;
- $II$ (defined in [6]) to be an interaction interpretation for $sig$ over $2WAY \cup 1WAY$ local to some node $n \in N$;
- $tr = \langle S, s_0, Act, R, \Sigma, AP, L, TIME, \Pi \rangle$ to be a SO-HL$^2$TS for $\Xi$;
- $\Delta$ (defined in [6]) to be an attribute interpretation for $VAR$ over $m$;
- $m = \langle N, W, C, client, spec, prov \rangle$ (defined in [6]) to be a service module.

## 3.2 Business Role Extension

Business role is defined over $sig$ and $VAR$. We extend it by introducing new formulas and predicates into *transitions* (see Fig. 3). These formulas and predicates are defined based on a set of terms.

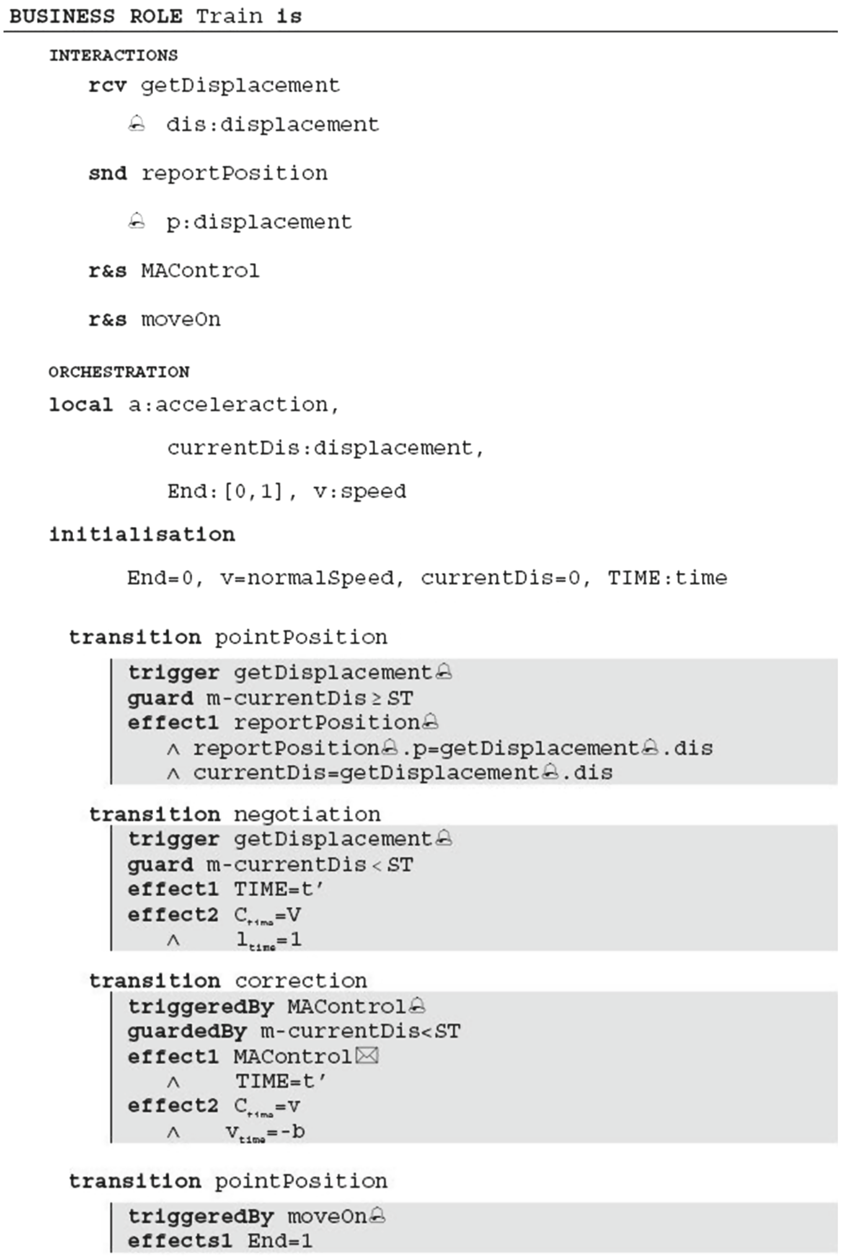
State terms denote the values of the variables and parameters of events in states. They are interpreted over states.

**Definition 4 (State Terms).** *The $D-$indexed family of sets $STERM$ of state terms is defined as follows:*

- *If $c \in F_d$, then $c \in STERM_d$ for every $d \in D$;*
- *If $f \in F_{<d_1,\ldots,d_{n+1}>}$ and $\overrightarrow{p} \in STERM_{<d_1,\ldots,d_n>}$, then $f(\overrightarrow{p}) \in STERM_{d_{n+1}}$ for every $d_1, \ldots, d_n, d_{n+1} \in D$;*
- *If $a \in NAME$ and $param \in PARAM(a)_d$, then $a.param \in STERM_d$ for every $d \in D$;*
- *If $t \in VAR_{time}$, then $t \in STERM_{time}$;*
- *If $v \in VAR_d$, then $v \in STERM_d$ for every $d \in D$ and $d \neq time$.*

For example in Fig. 2, in the **guard** of **transition** *negotiation*, terms "*m*", "*currentDis*" and "*ST*" are state terms.

**Definition 5 (Interpretation of State Terms).** *The interpretation of a state term $T \in STERM$ in a state $s \in S$, written $[\![T]\!]_s$, is defined as follows, where view is the function that defines how the parameter is observed:*



```
BUSINESS ROLE Train is
  INTERACTIONS
     rcv getDisplacement
        🔔 dis:displacement

     snd reportPosition

        🔔 p:displacement

     r&s MAControl

     r&s moveOn

  ORCHESTRATION
  local a:acceleraction,
          currentDis:displacement,
          End:[0,1], v:speed
  initialisation
       End=0, v=normalSpeed, currentDis=0, TIME:time

    transition pointPosition
      | trigger getDisplacement🔔
      | guard m-currentDis ≥ ST
      | effect1 reportPosition🔔
      |    ∧ reportPosition🔔.p=getDisplacement🔔.dis
      |    ∧ currentDis=getDisplacement🔔.dis
     transition negotiation
      | trigger getDisplacement🔔
      | guard m-currentDis < ST
      | effect1 TIME=t'
      | effect2 C_time=V
      |    ∧     l_time=1
     transition correction
      | triggeredBy MAControl🔔
      | guardedBy m-currentDis<ST
      | effect1 MAControl✉
      |    ∧     TIME=t'
      | effect2 C_time=v
      |    ∧    v_time=-b

    transition pointPosition
      | triggeredBy moveOn🔔
      | effects1 End=1
```

**Fig. 2.** Business role: train

- $[\![c]\!]_s = c_{\mathcal{U}}$
- $[\![f(T_1, \ldots, T_n)]\!]_s = f_{\mathcal{U}}([\![T_1]\!]_s, \ldots, [\![T_n]\!]_s)$

- $[\![a.param]\!]_s = view(II(a).param'^{\Pi^s})$
- $[\![t]\!]_s = TIME^s$
- $[\![v]\!]_s = v^{\Delta(s)}$

State predicates are defined based on state terms, and specify the properties of states. The satisfaction of state predicates is defined for states.

**Definition 6 (State Predicates).** *The state predicates* SP *is defined as follows:*

$$\chi ::= T_1 = (>,<,\neq)T_2 \mid \chi \wedge \chi' \mid \neg\chi \mid \chi \rightarrow \chi'$$

*with* $T_1, T_2 \in TERM_d$ *for some* $d \in D$.

For example in Fig. 2, **guard** $m - currentDIS \geq ST$ of **transition** *negotiation* is a state predicate.

**Definition 7 (Satisfaction of State Predicates).** *The satisfaction relation of state predicates is defined for every state* $s \in S$ *follows:*

- $s \models T_1 = (>,<,\neq)T_2$ *iff* $[\![T_1]\!]_s = (>,<,\neq)[\![T_2]\!]_s$
- $s \models \chi \wedge \chi'$ *iff* $s \models \chi$ *and* $s \models \chi'$
- $s \models \neg\chi$ *iff not* $s \models \chi$
- $s \models \chi \rightarrow \chi'$ *iff* $s \models \chi \rightarrow s \models \chi'$

Effect terms denote the values of the variables and parameters of events in transitions, so terms denoting variable values in the source state$(v, time)$ and in the target state $(v', time')$ within a transition are included. Effect terms are interpreted over transitions.

**Definition 8 (Effect Terms).** *The* $D$*-indexed family of sets* $ETERM$ *of effect terms is defined inductively as follows:*

- *The effect terms* $c, f(\overrightarrow{p})$ *and* $a.param$ *are defined the same way as state terms;*
- *If* $t \in VAR_{time}$ *then* $t, t' \in ETERM_{time}$*;*
- *If* $v \in VAR_d$*, then* $v, v' \in ETERM_d$ *for every* $d \in D$ *and* $d \neq time$.

For example in Fig. 2, in **effect1** of **transition** *pointPosition*, terms "$C_0$", "*currentDis*", "*reportPosition*🔔*.p*" and "*getDisplacement*🔔*.dis*" are effect terms.

**Definition 9 (Interpretation of Effect Terms).** *The interpretation of an effect term* $T \in ETERM$ *over a transition* $s \xrightarrow{\alpha} s'$ *written* $[\![T]\!]_{s \xrightarrow{\alpha} s'}$ *is defined as follows, where:* $II(param) = \langle param', view \rangle$*:*

- $[\![c]\!]_{s \xrightarrow{\alpha} s'} = c_{\mathcal{U}}$
- $[\![f(T_1, \ldots, T_n)]\!]_{s \xrightarrow{\alpha} s'} = f_{\mathcal{U}}([\![T_1]\!]_{s \xrightarrow{\alpha} s'}, \ldots, [\![T_n]\!]_{s \xrightarrow{\alpha} s'})$
- $[\![a.param]\!]_{s \xrightarrow{\alpha} s'} = view(II(a).param'^{\Pi^{\sigma'(0)}})$
- $[\![v]\!]_{s \xrightarrow{\alpha} s'} = v^{\Delta(s)}$
- $[\![v']\!]_{s \xrightarrow{\alpha} s'} = v^{\Delta(s')}$

- $[\![t]\!]_{s \xrightarrow{\alpha} s'} = TIME^{\sigma(s)}$
- $[\![t']\!]_{s \xrightarrow{\alpha} s'} = TIME^{s'}$

Effect formulas are defined based on effect terms, and specify the effects of state transitions. The satisfaction relation of effect formulas is defined for transitions.

**Definition 10 (Effect Formulas).** *The Effects Formulas* EF *is defined as follows:*

- $\chi ::= true \mid T_1 = T_2 \mid ini \mid \chi \wedge \chi' \mid \neg\chi$

*where* $T_1, T_2 \in ETERM_d$ *for some* $d \in D$, *and* $ini \in En^{INI}$.

For example in Fig. 2, $c_0 = getDisplacement$🔔$.dis$ in **effect1** of **transition** *pointPosition* is an effect formula.

**Definition 11 (Satisfaction of Effect Formulas).** *The satisfaction relation of effect formulas* EF *is defined for every transition* $s \xrightarrow{\alpha} s'$ *as follows:*

- $s \xrightarrow{\alpha} s' \models true$
- $s \xrightarrow{\alpha} s' \models T_1 = T_2$ *iff* $[\![T_1]\!]_{s \xrightarrow{\alpha} s'} = [\![T_2]\!]_{s \xrightarrow{\alpha} s'}$
- $s \xrightarrow{\alpha} s' \models ini$ *iff* $II(ini) \in PUB^{s \xrightarrow{\alpha} s'}$
- $s \xrightarrow{\alpha} s' \models \chi \wedge \chi'$ *iff* $s \xrightarrow{\alpha} s' \models \chi$ *and* $s \xrightarrow{\alpha} s' \models \chi'$
- $s \xrightarrow{\alpha} s' \models \neg\chi$ *iff not* $s \xrightarrow{\alpha} s' \models \chi$

Extended effect terms denote the values of variables along a trace of state $\sigma[0, r_\sigma]$; They extend effect terms by introducing the term $v_{time}$, which is used to denote the time derivative of variable $v$ at any time point $TIME^{\sigma(\zeta)}$. Where $\sigma \in \Sigma$ and $\zeta \in [0, r_\sigma]$. They are interpreted along traces.

**Definition 12 (Extended Effect Terms).** *The D-indexed family E-ETERM of sets of extended effect terms is defined inductively as follows:*

- *The extended effect terms* $c, f(\overrightarrow{p}), t, t', v$ *and* $v'$ *are defined the same way as effect terms;*
- *If* $v \in VAR_d$, *then* $v_{time} \in$ *E-ETERM*$_{d'}$.

For example in Fig. 2, in **effect2** of **transition** *correction*, terms "$C_{time}$", "$V$", "$V_{time}$" and "$-b$" are extended effect terms.

**Definition 13 (Interpretation of Extended Effect Terms).** *An extended semantics of an effect term* $T \in$ *E-ETERM is interpreted along a trace* $\sigma(0, r_\sigma)$, *written* $[\![T]\!]_{\sigma(0,r_\sigma)}$ *is defined as follows:*

- $[\![c]\!]_{\sigma(0,r_\sigma)} = c_{\mathcal{U}}^{\sigma(0,r_\sigma)}$
- $[\![f(T_1, \ldots, T_n)]\!]_{\sigma(0,r_\sigma)} = f_{\mathcal{U}}([\![T_1]\!]_{\sigma(0,r_\sigma)}, \ldots, [\![T_n]\!]_{\sigma(0,r_\sigma)})$
- $[\![v_{time}]\!]_{\sigma(0,r_\sigma)} = v_{time}^{\Delta(\sigma(0,r_{\sigma'}))}$
- $[\![v]\!]_{\sigma(0,r_\sigma)} = v^{\Delta(\sigma(0,r_{\sigma'}))}$

- $[\![t]\!]_{\sigma(0,r_\sigma)} = TIME^{\sigma(0,r_{\sigma'})}$

Extended effect formulas are defined based on extended effect terms, and specify the first order differential equations about certain variables and time (in Fig. 3 for example, in transition *negotiation*, $C_{time} = v_0$ is a differential equation about the displacement of a train $C$ and time, where $C$ is a globally defined variable). The satisfaction relation of extended effect formulas is defined for traces of states.

**Definition 14 (Extended Effect Formulas).** *The Extend Effects Formulas E-EF is defined as follows:*

- $\chi ::= true \mid v_{time} = T \mid \chi \wedge \chi' \mid \neg\chi$

*where* $v_{time}, T \in$ *E-ETERM.*

For example in Fig. 2, $C_{time} = V$ in **effect2** of **transition** *correction* is an extended effect formula.

**Definition 15 (Satisfaction for Extended Effect Formulas).** *The satisfaction relation for the extended effect formulas E-EF is defined for every trace* $\sigma[0, r_\sigma]$ *as follows:*

- $\sigma[0, r_\sigma] \models true$
- $\sigma[0, r_\sigma] \models v_{time} = T$ *iff* $v$ *is continuous over* $TIME^{\sigma[0,r_\sigma]}$ *and has a time derivative of value* $[\![T]\!]_{\sigma(\zeta)}$ *at each state* $\sigma(\zeta)$ *with* $\zeta \in (0, r_\sigma)$*;*
- $\sigma[0, r_\sigma] \models \chi \wedge \chi'$ *iff* $\sigma[0, r_\sigma] \models \chi$ *and* $\sigma[0, r_\sigma] \models \chi'$
- $\sigma[0, r_\sigma] \models \neg\chi$ *iff not* $\sigma[0, r_\sigma] \models \chi$

Using state predicates, effect formulas and extended effect formulas, we can specify a transition of a business role component in a SO-HL$^2$TS.

**Definition 16 (Transition Specification).** *A transition specification is a triple*

$$\langle trigger, guard, effect1, effect2 \rangle$$

*where* $trigger \in Act$*,* $guard \in SP$*,* $effect1 \in EF$ *and* $effect2 \in$ *E-EF.*

The satisfaction relation of transitions is defined for SO-HL$^2$TSs.

**Definition 17 (Transition Satisfaction).** *The SO-HL$^2$TS m satisfies a transition specification*

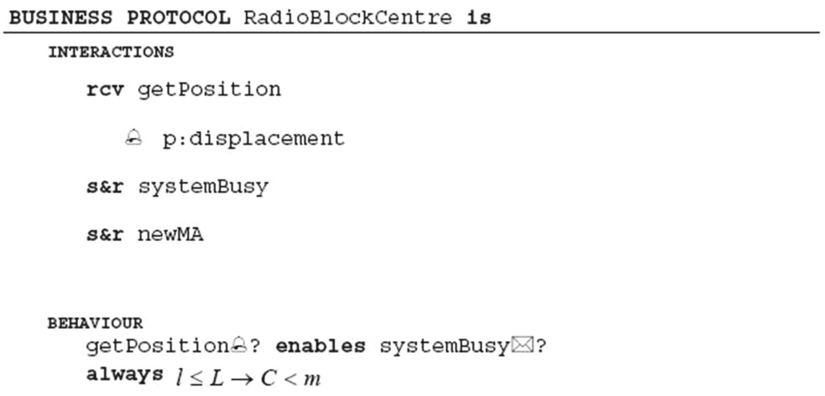
$$r = \langle trigger, guard, effects1, effect2 \rangle$$

*written* $m \models r$*, iff for every transition* $s \xrightarrow{\alpha} \sigma(0)$ *the following property hold: If* $II(trigger) \in \alpha, s \models guard$*, then*

$$s \xrightarrow{\alpha} \sigma(0) \models effect1 \text{ and } \sigma[0, r_\sigma] \models effect2.$$

### 3.3 Business Protocol Extensions

As introduced in Sect. 2.1, the behaviors of business protocols are specified through a set of behavior constraints. We introduce a new behavior constraint (see Fig. 4: **always** $l \leq L \rightarrow C < m$) by defining *hybrid programs* and *dTL formulas*. The behavior constraint captures common requirements along all the traces of a system run.



```
BUSINESS PROTOCOL RadioBlockCentre is
  INTERACTIONS
    rcv getPosition
       🔔 p:displacement
    s&r systemBusy
    s&r newMA

  BEHAVIOUR
    getPosition🔔? enables systemBusy✉?
    always l ≤ L → C < m
```

**Fig. 3.** Business protocol: RadioBlockCentre

Hybrid programs [8] generalize real-time programs [9] to hybrid change and are used to describe the behaviour of hybrid systems. They provide uniform discrete jumps and continuous evolutions along differential equations. In [12], hybrid programs are defined over a set variables and terms and used to specify dTL formulas. In this paper, we define hybrid programs with transitions specified in last section, and interpret them along hybrid traces.

**Definition 18 (Redefined Hybrid Programs).** *The set of hybrid programs HP of a* SO-H$L^2$ *TSs* m *is inductively defined as follows:*

- *If a transition* $r = \langle trigger, guard, effect1, effect2 \rangle$ *over* $m$ *has the satisfaction relation* $m \models r$*, then* $r \in HP$*;*
- *If* $\beta \in HP$*, then* $first\beta \in HP$*;*
- *If* $\beta, \gamma \in HP$*, then* $(\beta \cup \gamma) \in HP$*;*
- *If* $\beta, \gamma \in HP$*, then* $(\beta; \gamma) \in HP$*;*
- *If* $\beta \in HP$*, then* $(\beta^*) \in HP$*;*

**Definition 19 (Trace Semantics of Redefined Hybrid Program).** *The trace semantics of a redefined hybrid program* $\beta$*, written* $[\![\beta]\!]$*, is defined as follows:*

- $[\![trigger, guard, effect1, effect2]\!] = \{s \xrightarrow{\alpha} \sigma(0), \sigma[0, r_\sigma] : trigger \in \alpha, s \models guard, s \xrightarrow{\alpha} \sigma(0) \models effect1$ *and* $\sigma[0, r_\sigma] \models effect2\}$*;*

- $[\![first\beta]\!] = \{s \xrightarrow{\alpha} \sigma(0), \sigma[0,0] : (s \xrightarrow{\alpha} \sigma(0), \sigma[0, r_\sigma]) \in [\![\beta]\!]\}$
- $[\![\beta \cup \gamma]\!] = [\![\beta]\!] \cup [\![\gamma]\!]$;
- $[\![\beta; \gamma]\!] = \{\sigma \circ \varsigma : \sigma \in [\![\beta]\!], \varsigma \in [\![\gamma]\!]$ *when* $\sigma \circ \varsigma$ *is defined*$\}$;
- $[\![\beta^*]\!] = \bigcup_{n\in\mathbb{N}[\![\beta^n]\!]}$, *where* $\beta^{n+1} = (\beta^n; \beta)$ *for* $n \geq 1$.

Given a service module $m = \langle N, W, C, client, spec, prov\rangle$, the function $hp : m \to HP$ maps SRML specifications into hybrid programs. hp is constructed similar to the method provided in [6] (for details see [7]). For example, the hybrid program of the business role component *Train* in module *Train-Control* (see Fig. 2) is: $hp(Train) = [pointPosition^*] \cup [pointPosition^*; negotiation]$ $\cup[pointPosition^*; negotiation; correction]$.

Based on the definition of hybrid programs, we redefine dTL formulas as follows:

**Definition 20 (Redefined dTL Formulas).** *The sets* Fml *of dTL state formulas and $Fml_T$ of dTL trace formulas are inductively defined as the smallest set such that ($\phi \in Fml$ and $\pi \in Fml_T$):*

$$\phi ::= true \mid sp \mid \neg\phi \mid \phi \wedge \phi' \mid \phi \vee \phi' \mid \phi \to \phi' \mid \forall t\phi \mid \exists t\phi \mid [\beta]\pi \mid \langle\beta\rangle\pi$$
$$\pi ::= \phi \mid \Box\phi \mid \Diamond\phi$$

with $sp \in SP$, $t \in VAR_{time}$ and $\beta \in HP$.

Formulas without $\Box$ and $\Diamond$ are called *non-temporal* d$\mathcal{L}$ *formulas* [8]. Unlike in UCTL, state formulas are true on a trace if they hold for the last state of that trace but not for the first. Thus, $[\beta]\phi$ expresses that $\phi$ is true at the end of each trace of $\beta$. In contrast, $[\beta]\Box\phi$ expresses that $\phi$ is true all along all states of every trace of $\beta$. According to the valuation of dTL formulas defined in [12], we define the semantics of dTL formulas as follows:

**Definition 21 (Satisfaction of dTL Formulas).** Let $\langle S, s_0, Act, R, \Sigma, AP, L, TIME, \Pi\rangle$ *be a SO-HL$^2$TS. The satisfaction relation for dTL state formulas on each state $s \in S$ is defined as follows, where $s[t \mapsto \tilde{t}]$ denotes the modification that agrees with state $s$ on all variables except for variable $t \in VAR_{time}$:*

- $s \models true$;
- $s \models sp$ iff $sp \in L(s)$
- $s \models \neg\phi$ *iff not* $s \models \phi$;
- $s \models \phi \wedge \phi'$ *iff* $s \models \phi$ *and* $s \models \phi'$;
- $s \models \phi \vee \phi'$ *iff* $s \models \phi$ *or* $s \models \phi'$;
- $s \models \phi \to \phi'$ *iff* $s \models \phi \to s \models \phi'$;
- $s \models \forall t\phi$ *iff* $s[t \mapsto \tilde{t}] \models \phi$ *for all* $\tilde{t} \in VAR_{time}$;
- $s \models \exists t\phi$ *iff* $s[t \mapsto \tilde{t}] \models \phi$ *for some* $\tilde{t} \in VAR_{time}$;
- $s \models [\beta]\pi$ *iff for each trace* $\rho \in [\![\beta]\!]$ *with* $first\rho = s$, *if the satisfaction relation between* $\rho$ *and* $\pi$ *is defined then* $\rho \models \pi$;
- $s \models \langle\beta\rangle\pi$ *iff there is a trace* $\rho \in [\![\beta]\!]$ *with* $first\rho = s$, *if the satisfaction relation between* $\rho$ *and* $\pi$ *is defined then* $\rho \models \pi$.

*The satisfaction relation for dTL trace formulas with respect to trace* $\rho = (s \xrightarrow{\alpha} \sigma_1, \sigma_1[0, r_{\sigma_1}], \sigma_1(r_{\sigma_1}) \xrightarrow{\alpha_1} \sigma_2(0), \sigma_2[0, r_{\sigma_2}], \ldots)$ *is defined as follows where* $\phi$ *is a state formula and* $\Lambda$ *denotes the failure of a system run:*

- $\rho \models \phi$ *iff* $\rho$ *terminates and last*$\rho \models \phi$*, whereas the satisfaction relation between* $\rho$ *and* $\phi$ *is not defined if* $\rho$ *does not terminates;*
- $\rho \models \Box\phi$ *iff* $\sigma_i(\zeta) \models \phi$ *for all positions* $(i, \zeta)$ *of* $\rho$ *with* $\sigma_i(\zeta) \neq \Lambda$*;*
- $\rho \models \Diamond\phi$ *iff* $\sigma_i(\zeta) \models \phi$ *for some positions* $(i, \zeta)$ *of* $\rho$ *with* $\sigma_i(\zeta) \neq \Lambda$*;*

In the end we can define the new behaviour constraint ***always*** *s* for extending business protocol:

**Definition 22 (Hybrid Behaviour Constraint).** *For any service module* $m$ *with the corresponding sets* $C$ *and* $WW \in W$*:*

- *"**always** sp" stands for*

$$[hp(C, WW)]\Box sp$$

*(sp is true in each state along every trace of hybrid program* $hp(C, WW)$ *starting from the initial state, where* $sp \in SP$*).*

For example in Fig. 3, the **behaviour** "**alwalys** $l < L \rightarrow C < m$" stands for $[hp(Train)]\Box(l \leq L \rightarrow C < m)$.

## 4 Case Study: The Verification of Train-Control System

The model of Train-Control System is inspired by the European Train Control System(ETCS). As shown in Fig. 4, the system is composed by three components: Train, Radio Block Centers (RBC) and Balise which is melded with the railway. RBC grant or deny movement authorities (MA) to individual train by wireless communication. A train can not exceed the current MA (say $m$) in order to guarantee safety driving. The balise reports to the train its current position periodically, so the train knows how far it still is from the end of MA. Before entering negotiation at some point ST (in the "*far*" region), the train has sufficient distance to MA and can regulate its speed freely. When the train enters the region "*neg*", it sends a request to the RBC to apply for the MA extension and proceed with a constant speed $v_0$. If the train receives negative response from the RBC, it enters the "*cor*" region and proceed with acceleration $-b$. With the restriction of the scenario above, we have the hybrid program $hp(Train - ControlSystem(Train)) = [pointPosition^*] \cup [pointPosition^*; negotiation] \cup [pointPosition^*; negotiation; correction]$.

Figure 5 shows the SRML module *Train-Control*. Each element of the module is described as follows:

- business role: TR – a component that coordinates the movement process of the train, of type *Train*;

- business protocol: RBC– the external interface of the module which provides service to the external parties for knowing the current position of the train and issuing movement authority, of type *RadioBlockCentre*;
- business protocol: BA – the external interface of the module which requires service from the external parties for getting the current positioning signal, of type *Balise*;
- interaction protocol: RT, TB – two internal wires that make the partner relationship between RBC and TR, TR and BA explicitly.

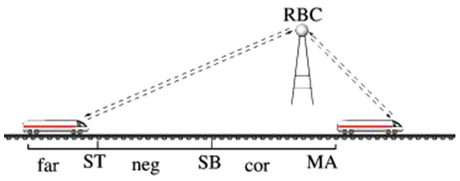


**Fig. 4.** ETCS train coordination

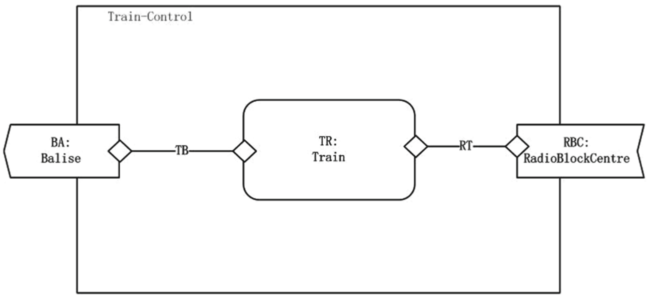


**Fig. 5.** Train-Control module

The whole SRML specification can be found in Appendix 2. In business role *Train*, $C$ : *displacement* is a global variable of the train displacement which is continuous in time. Differential equation $C_{time} = v_0$ describes the movement of the train in region "(neg)"; and differential equation set $C_{time} = V, V_{time} = -b$ describes the movement of the train in region "*cor*".

Next we show the verification of the behaviour, **always** $l < L \rightarrow C < m$, specified in business protocol *RadioBlockCentre*. This behaviour expresses that,under a initial condition $\phi$ for parameters, a train will always remain within its MA $m$, as long as the accumulated RBC negotiation latency $l$ is at most $L$. We assume that every transition $\sigma(t_\sigma) \xrightarrow{\alpha} \sigma'(0)$ takes so short time that we could approximately have $C^{\Delta(\sigma(r_\sigma))} = C^{\Delta(\sigma'(0))}, l^{\Delta(\sigma(r_\sigma))} = l^{\Delta(\sigma'(0))}$.

So we have $\psi \rightarrow [run(\text{Train-Control System})]\Box(l \leq L \rightarrow C < m)$, where $\psi$ is the set of initial propositions and $run(\text{Train-Control System}) = [pointPosition^*]$

$\cup\ [pointPosition^*; negotiation]\ \cup\ [pointPosition^*; negotiation; correction]$. According to the scenario, the train is in region "*far*" along trace $pointPosition^*$, we always have $C < m$. Thus the proof of this part can be omitted. In the end we have the following formula:

$$\begin{aligned}
\psi &\rightarrow [negotiation; correction]\Box\phi \qquad (2)\\
\phi &\equiv l \leq L \rightarrow C < m\\
\psi &\equiv C_0 < m \wedge v_0 > 0 \wedge l_0 = 0 \wedge L \geq 0\\
negotiation &\equiv getDisplacement\text{🔔}, m - currentdis < ST, true,\\
&\phantom{\equiv} C_{time} = v_0 \wedge l_{time} = 1\\
correction &\equiv MAControl\text{🔔}, m - currentdis < ST, MAControl\text{✉},\\
&\phantom{\equiv} C_{time} = V \wedge V_{time} = -b
\end{aligned}$$

where $C_0$ is the initial displacement and $l_0$ is the initial negotiation latency. As shown in Fig. 2, the train first negotiate with RBC while keeping a constant speed $V$ and the movement is controlled by equation ($C_{time} = V$) in transition *negotiation*. Then in transition *correction* the train brakes with acceleration $-b$ and the movement is controlled by equations ($C_{time} = V \wedge V_{time} = -b$).

We use the rule schema of the dTL calculus provided in [7] for our proof. The rule schema can be find in Appendix 1 and $\{\!|\cdot|\!\}$ brackets are used instead of modalities to visually identify the update prefix. We omit all the events, variables and parameters that don't appear in the state predicate $l < L \rightarrow C < m$.

The dTL proof of the constraint in (2) splits into two cases as follows:

$$\mathrm{P3}\,\dfrac{\mathrm{T1}\,\dfrac{\dfrac{\cdots}{\psi \vdash [negotiation]\Box\phi} \qquad \dfrac{\cdots}{\psi \vdash [negotiation][correction]\Box\phi}}{\psi \vdash [negotiation; correction]\Box\phi}}{\vdash \psi \rightarrow [negotiation; correction]\Box\phi}$$

The left branch proves that if $\phi$ holds during *negotiation*, an open condition $Lv_0 + C_0 < m$ should be satisfied. The proof is shown as follows:

$$\mathrm{T3}\,\dfrac{\mathrm{D7}\,\dfrac{\mathrm{D3}\,\dfrac{\mathrm{P10}\,\dfrac{\psi \vdash Lv_0 + C_0 < m}{\psi \vdash \forall t \geq 0 (t \leq L \rightarrow v_0 t + C_0 < m)}}{\begin{array}{c}\psi \vdash \forall t \geq 0 \{\!|getDisplacement\text{🔔}, m - currentdis < ST,\\ C = v_0 t + C_0 \wedge l = t, true|\!\}\phi\end{array}}}{\begin{array}{c}\psi \vdash [getDisplacement\text{🔔}, m - currentdis < ST, true,\\ C_{time} = v_0 \wedge l_{time} = 1]\phi\end{array}}}{\psi \vdash [negotiation]\Box\phi}$$

In the proof above, in the step applying P10, $L$ is replaced by $\forall t \geq 0$ to obtain a general case.In the step applying D3, $v_0 t + C_0$ is substituted by $C$ and $t$ is substituted by $l$. In the step applying D7, $C = v_0 t + C_0$ and $l = t$ are special solutions of differential equations $C_{time} = v_0$ and $l_{time} = 1$ respectively.

The right branch proves that if $\phi$ continues to holds after *negotiation* has completed when continuing with an adjusted acceleration $a$, an open condition

$v_0^2 < 2b(m - Lv_0 - C_0) \wedge Lv_0 + C_0 < m$ should be satisfied. The proof is shown as follows:

$$\begin{array}{rl}
 & \psi, t \ge 0 \vdash v_0^2 < 2b(m - Lv_0 - C_0) \wedge Lv_0 + C_0 < m \\ \cline{2-2}
\text{P10} & \psi, t \ge 0 \vdash \{\!|getDisplacement\Bell, m - currentdis < ST, \\
 & \qquad\qquad C = v_0t + C_0 \wedge l = t, true|\!\} \\
 & \forall \tilde{t} \ge 0(l \le L \to -\frac{b}{2}\tilde{t}^2 + v_0\tilde{t} + C_0 < m) \\ \cline{2-2}
\text{D3} & \psi, t \ge 0 \vdash \{\!|getDisplacement\Bell, m - currentdis < ST, \\
 & \qquad\qquad C = v_0t + C_0 \wedge l = t, true|\!\} \\
 & \forall \tilde{t} \ge 0 \{\!|MAControl\Bell, m - currentdis < ST, \\
 & \qquad\qquad MAControl\boxtimes \wedge C = -\frac{b}{2}\tilde{t}^2 + v_0\tilde{t} + C_0, true|\!\}\phi \\ \cline{2-2}
\text{T3,D7} & \psi, t \ge 0 \vdash \{\!|getDisplacement\Bell, m - currentdis < ST, \\
 & \qquad\qquad C = v_0t + C_0 \wedge l = t, true|\!\} \\
 & [MAControl\Bell, m - currentdis < ST, \\
 & \; MAControl\boxtimes, C_{time} = V \wedge V_{time} = -b]\Box\phi \\ \cline{2-2}
\text{P3} & \psi \vdash t \ge 0 \to \{\!|getDisplacement\Bell, m - currentdis < ST, \\
 & \qquad\qquad C = v_0t + C_0 \wedge l = t, true|\!\} \\
 & [correction]\Box\phi \\ \cline{2-2}
\text{P10} & \psi \vdash \forall t \ge 0 \{\!|getDisplacement\Bell, m - currentdis < ST, \\
 & \qquad\qquad C = v_0t + C_0 \wedge l = t, true|\!\} \\
 & [correction]\Box\phi \\ \cline{2-2}
\text{D7} & \psi \vdash [negotiation][correction]\Box\phi
\end{array}$$

In the proof above, in the first step applying P10, $\forall \tilde{t} \ge 0$ is substituted to obtain a general case. In the step applying D3, $-\frac{b}{2}\tilde{t}^2 = v_0\tilde{t} + C_0$ is substituted by $C$. In the step applying T3 and D7, $-\frac{b}{2}\tilde{t}^2 = v_0\tilde{t} + C_0$ is a special solution of differential equation set $C_{time} = V, V_{time} = -b$.

## 5 Concluding Remarks and Related Work

In this paper, we extended SRML semantic domain by defining HL$^2$TSs and it's hybrid traces which represent the system evolution. Based on this, we made a formal extension of SRML, which includes the extension of the language of business role and the language of business protocol. For our case study, we specified a Train-Control system with SRML and verified a safety constraint of it.

This work has been done mainly on the basis of [6,12]. In [6] a formal specification of SRML is provided. In [12], hybrid programs and the logic dTL for reasoning about the temporal behaviour of hybrid programs are defined, and a set of calculus for deductive verification is provided. In the SRML extension, we redefined hybrid programs and dTL formulas over the extended SRML semantic domain, thus enables the evolution of a service-oriented transition system with continuous traces can be reasoned about with the calculus in [12]. Furthermore, to define the HL$^2$TSs, we referenced [13] for Hybrid Automata; to redefine hybrid programs and dTL, we referenced [14,15] for Temporal Logic and Dynamic Logic basis. Different from various approaches for modeling and verifying hybrid systems, such as that provided in [17,18], our approach deals with hybrid transition systems, which integrate interactions among components with hybrid systems.

Although our work extends SRML, which is defined to specify service-oriented transition systems, it does not include the content of service discovery and binding. In [16] a formal operational semantics for service discovery and binding is brought forward. A prospect of our future work might be applying continuous time execution to this approach.

## Appendix 1: A Rule Schema of the dTL Calculus

A rule schema of the dTL calculus can be found in Table 1. In these rules, $\phi, \psi \in Fml$ and $\pi \in Fml_T$; $\chi_1 \in EF$ and $\chi_2 \in E\text{-}EF$, $T_1, \ldots, T_n \in ETERM$, $T \in E\text{-}ETERM$ and $t \in ETERM_{time}$. In D3, $M$ is a first-order formula and the substitution of $M_{T_1 \ldots T_n}^{T_1' \ldots T_n'}$, which replaces $T_1 \ldots T_n$ by $T_1' \ldots T_n'$ in M. In D7-D8,

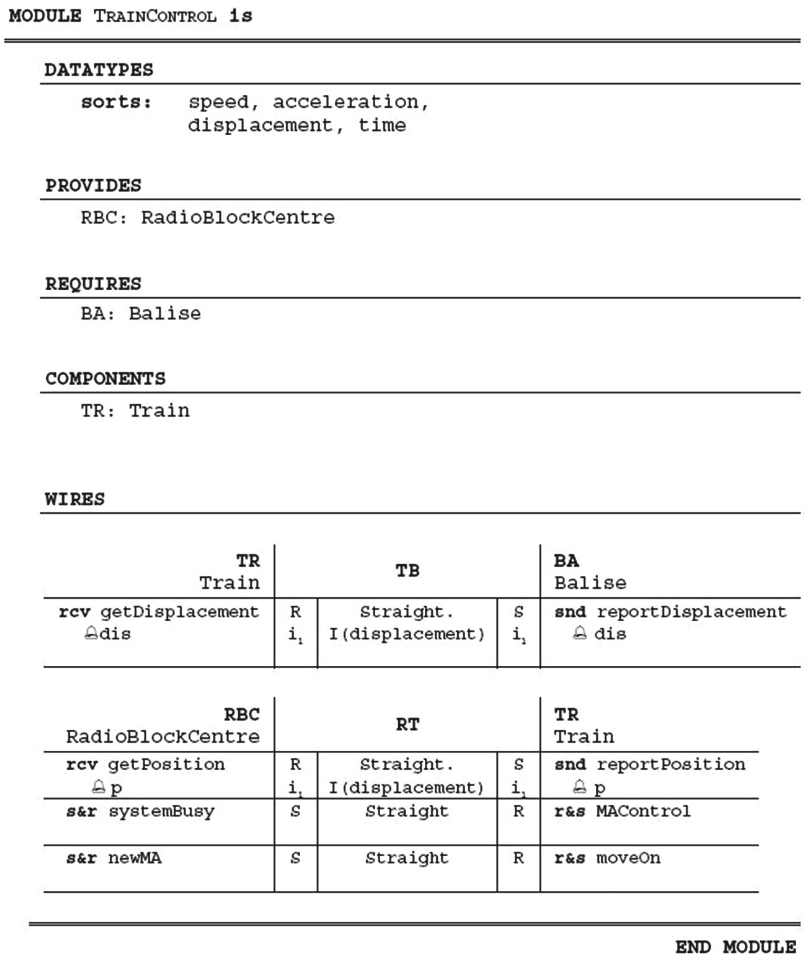
**Table 1.** Rule schemata of the temporal dynamic dTL verification calculus

| | |
|---|---|
| (P1) $\dfrac{\vdash\ \phi}{\neg\phi\ \vdash}$ | (P2) $\dfrac{\phi\ \vdash}{\vdash\ \neg\phi}$ |
| (P3) $\dfrac{\phi\ \vdash\ \psi}{\vdash\ \phi \to \psi}$ | (P4) $\dfrac{\phi, \psi\ \vdash}{\phi \wedge \psi\ \vdash}$ |
| (P5) $\dfrac{\vdash\ \phi \quad \vdash\ \psi}{\vdash\ \phi \wedge \psi}$ | (P6) $\dfrac{\vdash\ \phi \quad \psi\ \vdash}{\phi \to \psi\ \vdash}$ |
| (P7) $\dfrac{\phi\ \vdash \quad \psi\ \vdash}{\phi \vee \psi \vdash}$ | (P8) $\dfrac{\vdash\ \phi, \psi}{\vdash\ \phi \vee \psi}$ |
| (P9) $\dfrac{}{\phi\ \vdash\ \phi}$ | (P10) $\dfrac{M_0\ \vdash\ G_0}{M\ \vdash\ G}$ |
| (D1) $\dfrac{[\beta]\pi \wedge [\gamma]\pi}{[\beta \cup \gamma]\pi}$ | (D2) $\dfrac{\langle\beta\rangle\pi \vee \langle\gamma\rangle\pi}{\langle\beta \cup \gamma\rangle\pi}$ |
| (D3) $\dfrac{M_{T_1 \ldots T_n}^{T_1' \ldots T_n'}}{\{\!\vert trigger, guard, T_1 = T_1' \wedge \ldots \wedge T_n = T_n' \wedge \chi_1, \chi_2 \vert\!\} M}$ | (D4) $\dfrac{\{\!\vert\beta\vert\!\}\{\!\vert\gamma\vert\!\}\phi}{\{\!\vert\beta; \gamma\vert\!\}\phi}$ |
| (D5) $\dfrac{\phi \wedge [\beta; \beta^*]\phi}{[\beta^*]\phi}$ | (D6) $\dfrac{\phi \vee \langle\beta; \beta^*\rangle\phi}{\langle\beta^*\rangle\phi}$ |
| (D7) $\dfrac{\forall t \geq 0 [trigger, guard, v = f_{v_0}(T_1, \ldots, T_n, t) \wedge \chi_1, \chi_2]\phi}{[trigger, guard, \chi_1, v_{time} = T \wedge \chi_2]\phi}$ | |
| (D8) $\dfrac{\exists t \geq 0 \langle trigger, guard, v = f_{v_0}(T_1, \ldots, T_n, t) \wedge \chi_1, \chi_2\rangle\phi}{\langle trigger, guard, \chi_1, v_{time} = T \wedge \chi_2\rangle\phi}$ | |
| (T1) $\dfrac{[\beta]\Box\phi \wedge [\beta][\gamma]\Box\phi}{[\beta; \gamma]\Box\phi}$ | (T2) $\dfrac{\langle\beta\rangle\Diamond\phi \vee \langle\beta\rangle\langle\gamma\rangle\Diamond\phi}{\langle\beta; \gamma\rangle\Diamond\phi}$ |
| (T3) $\dfrac{[\beta]\phi \wedge [first\beta]\phi}{[\beta]\Box\phi}$ | (T4) $\dfrac{\langle\beta\rangle\phi}{\langle\beta\rangle\Diamond\phi}$ |
| (T5) $\dfrac{[\beta; \beta^*]\Box\phi}{[\beta^*]\Box\phi}$ | (T6) $\dfrac{\langle\beta; \beta^*\rangle\Diamond\phi}{\langle\beta^*\rangle\Diamond\phi}$ |

$f_{v_0}$ is the solution of the initial value problem $v_{time} = T, v^{\triangle(\sigma(0))} = v_0$, where $\sigma(0)$ is the state in which variable $v$ has the value $v_0$. In P10, $Cl_\forall(F_0 \rightarrow G_0) \rightarrow Cl_\forall(F \rightarrow G)$ is an instance of a first-order tautology of real arithmetic and $Cl_\forall$ is the universal closure.

## Appendix 2: SRML Specification of Module *Train-Control*

The SRML specification of service module *Train-Control* is shown in Fig. 6.



**MODULE** TRAINCONTROL **is**

**DATATYPES**

```
sorts:   speed, acceleration,
         displacement, time
```

**PROVIDES**

RBC: RadioBlockCentre

**REQUIRES**

BA: Balise

**COMPONENTS**

TR: Train

**WIRES**

| **TR** Train | | **TB** | | **BA** Balise |
|---|---|---|---|---|
| **rcv** getDisplacement 🔔dis | R $i_1$ | Straight. I(displacement) | S $i_1$ | **snd** reportDisplacement 🔔 dis |

| **RBC** RadioBlockCentre | | **RT** | | **TR** Train |
|---|---|---|---|---|
| **rcv** getPosition 🔔 p | R $i_1$ | Straight. I(displacement) | S $i_1$ | **snd** reportPosition 🔔 p |
| **s&r** systemBusy | S | Straight | R | **r&s** MAControl |
| **s&r** newMA | S | Straight | R | **r&s** moveOn |

**END MODULE**

**Fig. 6.** Module *Train-Control*

## References

1. Georgakopoulos, D., Papazoglou, M.: Service-Oriented Computing. The MIT Press Cambridge, Massachusetts (2009)
2. van der Schaft, A., Schumacher, H.: An Introduction to Hybrid Dynamical Systems. Springer, London, UK (1999)
3. Abreu, J., Bocchi, L., Fiadeiro, J.L., Lopes, A.: Specifying and composing interaction protocols for service-oriented system modelling. In: Derrick, J., Vain, J. (eds.) FORTE 2007. LNCS, vol. 4574, pp. 358–373. Springer, Heidelberg (2007)
4. Building Systems using a Service Oriented Architecture. White paper version 0.9 (2005)
5. Fiadeiro, J., Lopes, A., Bocchi, L., Abreu, J.: The Sensoria reference modelling language. In: Wirsing, M., Hölzl, M. (eds.) SENSORIA. LNCS, vol. 6582, pp. 61–114. Springer, Heidelberg (2011)
6. Abreu, J.: Modelling business conversations in service component architectures. Ph.D thesis, University of Leicester (2009)
7. Yu, N.: Injecting continuous time execution into service-oriented computing. Ph.D thesis, Ludwig-Maximilians-Universität München (to be appeared)
8. Platzer, A.: Towards a hybrid dynamic logic for hybrid dynamic systems. In: Blackburn, P., Bolander, T., Braüner, T., de Paiva, V., Villadsen, J. (eds.) LICS International Workshop on Hybrid Logic 2006, Seattle USA, ENTCS (2007)
9. Henzinger, T., Nicollin, X., Sifakis, J., Yovine, S.: Symbolic model checking for read-time systems. In: LICS, pp. 394–406. IEEE Computer Society (1992)
10. Alonso, G., Casati, F., Kuno, H., Machiraju, V.: Web Services. Springer, New York (2004)
11. ter Beek, M.H., Fantechi, A., Gnesi, S., Mazzanti, F.: An action/state-based model-checking approach for the analysis of communication protocols for service-oriented applications. In: Leue, S., Merino, P. (eds.) FMICS 2007. LNCS, vol. 4916, pp. 133–148. Springer, Heidelberg (2008)
12. Platzer, A.: AVACS - Automatic verification and analysis of complex systems. Technical report No. 12, AVACS (2007)
13. Henzinger, T.A.: The theory of hybrid automata. In: Lnan, M.K., et al. (eds.) LICS 1996, vol. 170, pp. 256–292. Springer, Berlin (2000)
14. Emerson, E.A.: Temporal and modal logic. In: van Leeuwen, J. (ed.) HTCS, vol. A, pp. 995–1072. Elsevier, Amsterdam (1995)
15. Harel, D., Kozen, D.: Dynamic Logic. The MIT Press Cambridge, Massachusetts (2000)
16. Fiadeiro, J., Lopes, A., Bocchi, L.: An abstract model of service discovery and binding. In: Formal Aspects of Computing, vol. **23**, pp. 433–463. Springer, Berlin (2011)
17. Fadlisyah, M., Ölveczky, P.C., Ábrahám, E.: Formal modeling and analysis of human body exposure to extreme heat in HI-maude. In: Durán, F. (ed.) WRLA 2012. LNCS, vol. 7571, pp. 139–161. Springer, Heidelberg (2012)
18. Quesel, J., Mitsch, S., Loos, S., Aréchiga, N., Platzer, A.: How to model and prove hybrid systems with KeYmaera: A tutorial on safety. In: STTT (2015)
19. European Train Control System (ETCS) Open Proofs - Open Source. http://openetcs.org/