ASTES

Advances in Science, Technology and Engineering Systems Journal
Vol. 6, No. 6, 60-65 (2021)
www.astesj.com
Special Issue on Innovation in Computing, Engineering Science & Technology

ASTES Journal
ISSN: 2415-6698

Modelling and Testing Services with Continuous Time SRML

Ning Yu, Martin Wirsing*

Institute for Informatics, LMU Munich, 80538 Munich, Germany

ARTICLE INFO

Article history:
Received: 31 July, 2021
Accepted: 26 October, 2021
Online: 20 November, 2021

Keywords:
continuous time SRML
Service-Oriented Hybrid Doubly Labelled Transition System
differential equations
traffic control system

ABSTRACT

The SENSORIA Reference Modeling Language (SRML) aims at modelling composite services at a high level. Continuous time SRML extends SRML so that it can model services whose components can perform both discrete processes and continuous processes. In order to show how continuous time SRML is applied, in this paper, we systematically introduce our study on continuous time SRML in the following approach: First we introduce the theoretical foundation of continuous time SRML, the Service-Oriented Hybrid Doubly Labeled Transition Systems. This is the semantic domain over which continuous time SRML is defined and interpreted. Then we design a case study of a traffic control system. In the case study, we illustrate the scenario of the system, explain the continuous time SRML model of the system, and show how to transform the model to a kind of Deterministic Finite Automata that can be used for testing and verification. Finally, we show our idea of testing our model with the IBM WebSphere Process Server. With this approach, we come to a conclusion that continuous time SRML can be used to model certain systems in the real-world, and can be tested with proper tools.

1 Introduction

Service-Oriented Computing (SOC) [1] is a paradigm for distributed computing, in which computation units are abstracted as services. The SENSORIA Reference Modelling Language (SRML) [2, 3] has been developed in the IST-FET integrated project SENSORIA [4], and it is a prototype domain-specific language for abstracting service-oriented systems at a high level abstraction. Hybrid systems arise in embedded control when discrete components are coupled with continuous components. Continuous time SRML [5] is an extension of SRML, and it aims at abstracting service-oriented systems in which hybrid systems are embedded. That is: the discrete components and continuous components are abstracted as components of services. Being different from SRML, continuous time SRML can be used to formally model services, whose components can perform both discrete processes and continuous processes that can be describe by differential equations.

In [6], we introduced the application of continuous time SRML through the case study of a Traffic Control System. In this paper, we give a novel systematic presentation of our approach from the semantic domain of continuous time SRML, to the case study, and finally to the test environment. Continuous time SRML is interpreted over Service-Oriented Hybrid Doubly Labeled Transition Systems [5] (SO-HL^{2}TS). This kind of transition systems extends Service-Oriented Doubly Labeled Transition Systems [7] (SO-L^{2}TS), so that each state of a SO-L^{2}TS is lifted to a function mapping a time interval to an infinite trace of state, over which the differential equations in continuous time SRML can be interpreted. Our approach includes two parts: specifying the system with continuous time SRML modules, and transforming the modules to a hybrid automaton-like Deterministic Finite Automaton (DFA). In order to test the system module, we combine IBM Websphere Integration Designer [8] (WID) and Matlab to form the test environment. WID is used to implement the continuous time SRML module, and Matlab [9] is used to compute the variables controlled by differential equations. Thus, the main contribution of this paper is to provide a way to test or verify the complex systems (like the Traffic Control System) that is specified with continuous time SRML.

This paper is arranged as follows: In Section 2, we introduce the related work such as hybrid automata, model checking tools and the early research of this topic. In Section 3, we introduce HL^{2}TS, the semantic domain of continuous time SRML. In Section 4, we show the case study of the traffic control system. In Section 5, we introduce the testing environment and show the idea of implementing the traffic control system. In Section 6, we make a conclusion of this paper and introduce our future work.

*Corresponding Author: Martin Wirsing, Oettingenstraße 67,80538 Munich (Germany), +49 89 2180 9154 & wirsing@pst.ifi.lmu.de

2 Related work

SRML is brought forward in [7], and it is used to model important properties of services and the business conversations within and among services in SOC. A service specified with SRML can only perform discrete activities. Continuous time SRML extends the semantic domain and syntax of SRML, such that it could express differential equations which represents the continuous process performed by services. This essential property of continuous time SRML enables it be able to model the combination of service-oriented systems and hybrid systems. We will introduce this in Section 3.

Hybrid automata [10] are formal models for hybrid systems. They abstract hybrid systems with graphs and workable representations. A hybrid automaton consists of a set of variables, a control graph, some essential conditions and a set of events. These compositions match the compositions of a service component that is specified with continuous time SRML. Thus, a service specified by continuous time SRML can be transformed to a hybrid automata-like DFA, this will be introduced in Section 4. Hybrid automata are interpreted over timed transition systems, while continuous time SRML is interpreted over SO-HL^{2}TS. Timed transition systems abstract the duration of a continuous process as a parameter of a transition. In this way, it is simple and clear for abstracting continuous processes. SO-HL^{2}TS specify continuous processes as traces of infinite states with definite initial states and finial states. This enables us to know the values of all the variable of an arbitrary state.

In [11], the author provides another way for modelling hybrid systems. In that paper, continuous processes are also interpreted over traces of states. This is the original idea from which the traces of states in SO-HL^{2}TS come. But in [11], discrete processes are not modelled between traces of states. In [12], the author also provides rule schemata for verifying the hybrid systems. Moreover, based on this work, a model checking tools for hybrid systems, KeYmaera X, is brought forward in [13]. In [14], provides an approach for simulating embedded systems with Real-Time Maude. Compared with our approach for testing, this can be another way to test the traffic control model, but the interactions of a service would not be implemented as easy and explicate as that in WID.

3 Continuous time SRML and its semantic domain

Continuous time SRML models composite services, whose elementary components may involve in continuous time executions. The services in continuous time SRML are represented with service modules. A service module is composed of service components, external service interfaces and internal wires linking between them. In Figure 1, we see the compositions of a service module. In the service module, SC is a service component, EX-P is an external service interface (provides-interface), every EX-R is an external service interface (requires-interface), and every IW is an internal wire.

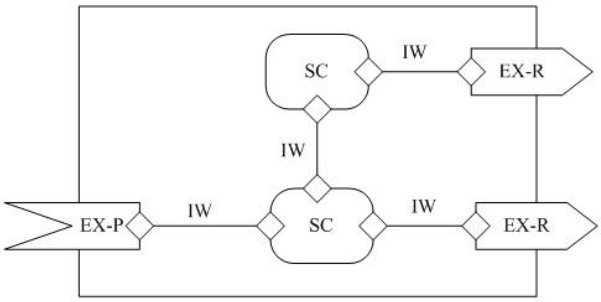

Figure 1: A service module of continuous time SRML

In a service module of continuous time SRML, a service component is specified with a business role, an external service interface is specified with a business protocol, and an internal wire is specified with a connector. The full syntax of continuous time SRML can be found in [5], and they are interpreted over SO-HL^{2}TSs.

SO-HL^{2}TSs are extensions of HL^{2}TSs [5, 15] such that they enrich HL^{2}TSs with service-oriented features. HL^{2}TSs extends the L^{2}TS [16] in that it defines a set of functions Σ. These functions map from the real number domain to the state domain of the HL^{2}TS and can be used to interpret the continuous processes described by differential equations. The following definition of HL^{2}TS is from [5, 15].

Definition 1 (Hybrid Doubly Labeled Transition System) *A Hybrid Doubly Labeled Transition System* (HL^{2}TS) is a tuple

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

where:

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

In Definition 1, the set S of state can be finite or infinite; for every $\sigma \in \Sigma$, the real number r_σ is unique, and function σ on the interval $[0, r_\sigma]$ represents the prolongation of states in the continuous process with the duration r_σ; function L is a state labeling function that defines which propositions are true in each state, and these propositions are from the fixed set of atomic propositions AP.

In order to add service-oriented features to HL^{2}TSs, we refine HL^{2}TSs with Service-Oriented Transition Systems (SO-TSs)

[2]. SO-TSs are used that models the communications in service-oriented systems during the computing processes. In a SO-TS, states and transitions are labelled with certain sub-sets of events that appear in the computing processes. With such refinements, we can obtain SO-HL^{2}TSs. The following definition of SO-HL^{2}TSs is from [5, 15].

Definition 2 (Service-Oriented Hybrid L^2TS) A *Service-Oriented Hybrid L^2TS (SO-HL^{2}TS)* that refines a SO-TS $\langle S, \rightarrow, s_0, G\rangle$ is a tuple

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

where:

- $Act = \{e! : e \in E\} \cup \{e¡ : e \in E\} \cup \{e? : e \in E\} \cup \{e¿ : e \in E\}$;
- $R \subseteq \{\sigma(r_\sigma) : \sigma \in \Sigma\} \times 2^{Act} \times \{\sigma(0) : \sigma \in \Sigma\}$ is such that:,

  $\sigma(r_\sigma) \rightarrow \sigma'(0)$ *iff* $(\sigma(r_\sigma), \alpha, \sigma'(0)) \in R$ for some $\alpha \in 2^{Act}$
- $AP = Act \cup PLG$;
- *For every* $\sigma \in \Sigma$ *and every* $\zeta \in [0, r_\sigma]$, $L : S \rightarrow 2^{AP}$ is such that:

  $$L(\sigma(\zeta)) = \{e!, e¡, e?, e¿ : e \in HST^{\sigma(\zeta)}\} \cup PLG^{\sigma(\zeta)}$$

In Definition 2, the set Act is refined with the sets of events that are published (denoted by e!), delivered (denoted by e¡), executed (denoted by e?)and discarded (denoted by e¿) respectively, E is the set of all the events. The set AP of atomic propositions is labelled with the set of actions and the set of pledges that associate with every interaction (the propositions that hold during the time when the interaction is valid). Every state is labelled with the actions whose associating events are from the history of the events of that state, and the pledges that hold in the state. We also specify the time in each state with the function $TIME$. The full definition can be seen in [5]

When applying a function $\sigma \in \Sigma$ to its interval $[0, r_\sigma]$, we obtain a infinite *trace of states* $\sigma_0, \ldots, \sigma_{r_\sigma}$. By connecting traces of states with possible actions, *paths of states* are formed. Because a trace of states represent the evolution of a continuous process, actions which represent the discrete processes can only take place at the initial and finial states of each trace. Thus, a transition in a path only exists between the last state of one trace and the first state of another trace. This can be seen in the definition of transition relation R in Definition 2. The following definition of traces and paths of states is from [5, 15].

Definition 3 (Paths of SO-HL^{2}TSs) *Given a SO-HL^{2}TS* $m = \langle S, s_0, \Sigma, Act, R, AP, L, TIME, \Pi\rangle$, *paths of m are defined as follows:*

- *For every* $\sigma \in \Sigma$, $\sigma(0)\ldots\sigma(r_\sigma)$ *denotes the trace of states of* σ *where* $\sigma(0)$ *is the first state of the trace and* $\sigma(r_\sigma)$ *is the last state of the trace.* $\sigma(0)\ldots\sigma(r_\sigma)$ *includes all the states in the set* $\{\sigma(\xi)|0 \leq \xi \leq r_\sigma\}$, *which can be finite or infinite.*
- $\rho = (\sigma_1(0)\ldots\sigma_1(r_{\sigma_1}), \sigma_2(0)\ldots\sigma_2(r_{\sigma_2}), \ldots)$ *is a path of m if there exists an* $\alpha \in 2^{Act}$ *such that for every* $\sigma_i(r_{\sigma_i})$ *and* $\sigma_{i+1}(0)$ *with* $i \in \mathbb{N}$, *there exists an* $\alpha \in 2^{Act}$ *such that* $(\sigma_i(r_{\sigma_i}), \alpha, \sigma_{i+1}(0)) \in R$;
- *The states in a path are ordered lexicographically such that, for every* $i, j = 1, 2, \ldots$ *and* $\zeta \in [0, r_{\sigma_i}], \xi \in [0, r_{\sigma_j}]$, *there is* $\sigma_i(\zeta) \prec \sigma_j(\xi)$ *iff either* $i < j$, *or* $i = j$ *and* $\zeta < \xi$;
- *For the states in a path, there is:*
  - *for every* $i, j = 1, 2, \ldots$ *and* $\zeta \in [0, r_{\sigma_i}], \xi \in [0, r_{\sigma_j}]$, *there is* $TIME^{\sigma_i(\zeta)} \leq TIME^{\sigma_j(\xi)}$ *if* $i < j$, *and* $TIME^{\sigma_i(\zeta)} < TIME^{\sigma_j(\xi)}$ *if* $i = j$ *and* $\zeta < \xi$;

  In particular, if $TIME^{\sigma_i(r_{\sigma_i})} < TIME^{\sigma_{i+1}(0)}$ *then the transition* $(\sigma_i(r_{\sigma_i}), \alpha, \sigma_{i+1}(0))$ *takes time, otherwise if* $TIME^{\sigma_i(r_{\sigma_i})} = TIME^{\sigma_{i+1}(0)}$ *the transition is executed in zero time;*
- *A path* ρ *terminates if it is a finite sequence* $\sigma_1(0)\ldots\sigma_1(r_{\sigma_1}), \ldots, \sigma_n(0)\ldots\sigma_n(r_{\sigma_n})$. *In such case the first state of the trace* $\sigma_1(0)$ *is denoted by* $first\rho$ *and the last state* $\sigma_n(r_{\sigma_n})$ *is denoted by* $last\rho$;
- *The concatenation of traces* $\rho_1 = (\sigma_1(0)\ldots\sigma_1(r_{\sigma_1}), \sigma_2(0)\ldots\sigma_2(r_{\sigma_2}), \ldots)$ *and* $\rho_2 = (\varsigma_1(0)\ldots\varsigma_1(r_{\varsigma_1}), \varsigma_2(0)\ldots\varsigma_2(r_{\varsigma_2}), \ldots)$, *denoted by* $\rho_1 \circ \rho_2$, *is defined as follows:*
  - $\rho_1 \circ \rho_2 = (\sigma_1(0)\ldots\sigma_1(r_{\sigma_1}), \ldots, \sigma_n(0)\ldots\sigma_n(r_{\sigma_n}), \varsigma_1(0)\ldots\varsigma_1(r_{\varsigma_1})\ldots)$ *iff* ρ_1 *terminates at* $\sigma_n(r_{\sigma_n})$ *and* $(\sigma_n(r_{\sigma_n}), \alpha, \varsigma_0(0)) \in R$;
  - $\rho_1 \circ \rho_2 = \rho_1$ *iff* ρ_1 *does not terminate;*
  - $\rho_1 \circ \rho_2$ *is not defined in other cases;*
- λ *is an empty hybrid trace such that for any arbitrary hybrid trace* ρ, $\rho \circ \lambda = \lambda \circ \rho = \rho$.

The full semantics of continuous time SRML is defined based on Definition 2 and Definition 3, and can be found in [5].

4 Traffic control system and the transformation process

In order to show the application of continuous time SRML, we design a case study of a traffic control system to control the self-driving trains. A traffic control system includes a local traffic control center, a built-in intelligent driving system on a self-driving train, and a monitoring center. Suppose depending on the data obtained from the sensors on the train and the built-in intelligent driving system, the self-driving train can move on their own in most of the driving time without communicating to the outside. But when approaching train stations, they need to get additional information from the local traffic control center, e.g. to know if the station is free, in order to guarantee safe driving. In addition, the monitoring center needs to know the status of the train in some condition, so that they can react in time if the train is not driving safely. In our case study, we show in a traffic control system, how the intelligent driving system on a self-driving train communicates with the local traffic control center and the monitoring center when the train is approaching a train station. The control scheme of the intelligent driving system is adapted from the model of a train system in [17], which extends ETCS [18] level 3 for rail-road crossings.

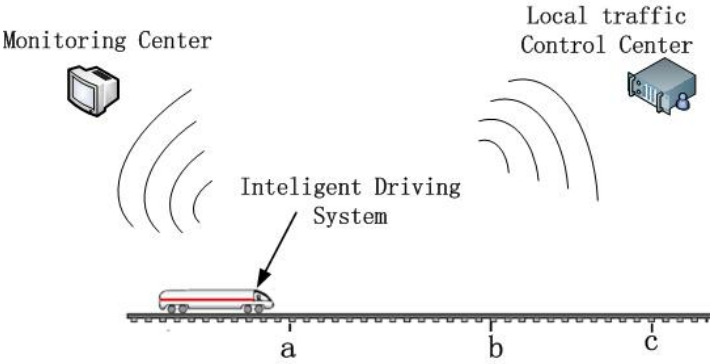

Figure 2: A brief scenario of the traffic control system

```
BUSINESS ROLE IDS is
    INTERACTIONS
        s&r VControl
            _receive newSpeed:speed
        rcv VMoveOn
            _receive newSpeed:speed
        s&r Brake
            _receive dec:acc
        r&s BrakeInfo
            _send currPos:position
                 currTime:time
    ORCHESTRATION
    var C:position, B:acc, d²C:acc, dC:speed, v₀:speed, v₁:speed, t:time
    init C=0∧v₀=100∧v₁=0∧B=0∧t=0 ∧dC=v₀∧d²C=B
    transition Nego
            trigger VControl_receive
            guard dC=v₀
            effect dC=v₁
    transition Corr
            trigger Brake_receive
            guard dC=v₁
            effect d²C=B
    transition Cont
            trigger VMoveOn_receive
            guard dC=v₁
            effect dC=v₀
    transition Stop
            trigger  BrakeInfo_receive
            guard d²C=B
            effect d²C=B
```

Figure 3: Business Role *IDS*

Figure 2 shows a brief view of the scenario. In this figure, we assume that the train station is at point c, and on reaching point a, the train moves with a constant speed v_0. When the train reaches point a, as it needs to move at a lower speed, the intelligent driving system sends a request to the local traffic control center, to ask with which speed the train can move on. Based on the condition of the current traffic condition, the local traffic control center replies with the value of the proper speed. On receiving this message, the intelligent driving system sets a constant speed v_1 of this value to the train. If the local traffic control center finds the traffic is lighter and the train can move with a higher speed, it sends a message to the intelligent driving system with the value of the higher speed. On receiving this message, if the train hasn't passed point b, the intelligent driving system sends another request to the local traffic control center, to ask with which lower speed it can move on in the future. Before the train reaches point b, this process can repeat for several times. When the train reaches point b, the intelligent driving system sends a request to the local traffic control center, to ask with which deceleration the train can brake. Based on the condition of the current traffic condition, the local traffic control center replies with the value of the proper deceleration. On receiving this message, the intelligent driving system sets the deceleration of the train to the required deceleration. After the train brakes, if the intelligent driving system receives the request from the monitoring center, it sends the current condition (position, speed, deceleration, and so on) of the train to the monitoring center.

We abstract the whole system with continuous time SRML as a traffic control service module, and abstract the intelligent Driving System with business role *IDS*, the Local Traffic Control Center with business protocol *TCC*, and the Monitoring Center with business protocol *MOC*. In this paper, we concentrate on the computational reality of a service module, and ignore the interactions caused by communications with the outside parties at the service interfaces, thus, we only make study to the business role *IDS*.

A business role includes a set of interactions and an orchestration in which these interactions are organized. An interaction consists of a type (e.g. *s&r*) and a name (e.g. *VControl*) and attributes of event types (e.g. *_receive*) and parameters (e.g. *newSpeed:speed*). With this structure, we can specify events that associate with an interaction (e.g. *VControl_receive*), and parameters that associate with an event (e.g. *VControl_receive.newSpeed*). An orchestration of a business role includes a set of variables that is local to the service component abstracted by the business role, a set of initial values of the variables that identifies the initial state of the service component, and a set of transitions that model the activities performed by the service component. A transition consists of a *trigger*, a *guard* and an *effect*. A *trigger* is an event or a state condition that specifies the condition for the transition to occur. A *guard* is a condition that identifies the states in which the transition can occur. An *effect* is the proposition that is true in the state that follows the state specified by the guard condition, and represents the effect of the transition. In the declaration of a transition, we don't model the actions caused by the transition (e.g. the assignments of local variables) because they are not considered in the verification procedure.

Figure 3 shows the specification of business role *IDS*. *IDS* contains four interactions and four transition rules. Interaction *VControl* denotes the communication between *IDS* and *TCC*: when the train reaches point a, the *IDS* sends a request to *TCC* to ask for a lower speed, and it can receive the reply from *TCC* with parameter *newSpeed* when the train is between point a and point b, the value of *newSpeed* is then assigned to local variable v_1; interaction *VMoveOn* denotes the communication between *IDS* and *TCC*: when the train is between point a and point b and the traffic is lighter, *IDS* can receive a message from *TCC* with parameter *newSpeed*, indicating that the train can move with a higher speed, the value of *newSpeed* is then assigned to local variable v_0; interaction *Brake* denotes the communication between *IDS* and *TCC*: when the train reaches point b, the *IDS* sends a request to *TCC* to ask for a braking deceleration, and it can receive the reply from *TCC* with parameter *dec* when the train is after point b, the value of *dec* is then assigned to local variable B; interaction *BrakeInfo* denotes the communication between *IDS* and *MOC*: when the train is between point b and point c, and it brakes, *IDS* can receive a request from *MOC* asking the current position of the train and the current time, and it replies to

MOC with parameter *currPos* and *currTime*, which take the values of local variable C and t respectively. Transition *Nego* describes that the train changes its speed from v_0 to v_1 when it receives the slow down message from the local traffic control center; transition *Cont* describes that the train changes its speed from v_1 to v_0 when it receives the speed up message from the local traffic control center; transition *Corr* describes that the train brakes with the deceleration B when it receives the braking message from the local traffic control center; transition *Stop* describes that the train receives the request message from the monitoring center, and replies with its current position and the current time. In business roles, the continuous time executions are specified in the form of differential equations. (e.g. $dC = v_0$ in Figure 3). In general, in continuous time SRML, the n *th* derivative of variable a to time variable t, $\frac{\partial^n a}{\partial t^n}$, is denoted by $d^n a$. The formal semantics of business roles can be found in [5].

Next we take transition *Nego* as an example to illustrate how a transition in *IDS* can be interpreted over a SO-HL^{2}TSs: Suppose a SO-HL^{2}TSs $h = \langle S, s_0, \Sigma, Act, R, AP, L, TIME, \Pi\rangle$ satisfies *IDS* (the satisfaction of a business role over a SO-HL^{2}TS can be found in [5]), $\sigma(r_\sigma) \xrightarrow{\alpha} \sigma'(0)$ is a transition in h ($\sigma, \sigma' \in \Sigma$ and $\alpha \in Act$), if $VControl\_receive \in \alpha$ and $\sigma(r_\sigma) \models dC = v_0$, then $\sigma'(0)\cdots\sigma'(r_{\sigma'}) \models dC = v_1$. Particularly, formula $dC = v_1$ represents the differential equation $\frac{\partial C}{\partial t} = v_1$, where t denotes the continuous time variable. Such a formula expresses that variable C changes according to the differential equation $\frac{\partial C}{\partial t} = v_1$ along the trace $\sigma'(0)\cdots\sigma'(r_{\sigma'})$ of h.

In order to be able to test or verify the Traffic-Control module that is explained in this section, we show how to transform the business role *IDS* to a hybrid automata-like DFA. A Deterministic Finite Automata (DFA) [19] consumes a string of input symbols over a set of states, and it operates as follows: when inputting a symbol, the DFA transits from one state to another state, and it does the transitions until all input symbols have been consumed. In a business role, if we take the trigger and guard of every transition as an input symbol, and label the effect of every transition to a state, then we can construct a DFA based on the business role. According to the scenario of the traffic control system and the specification of business role IDS, we can construct the DFA shown in Figure 4.

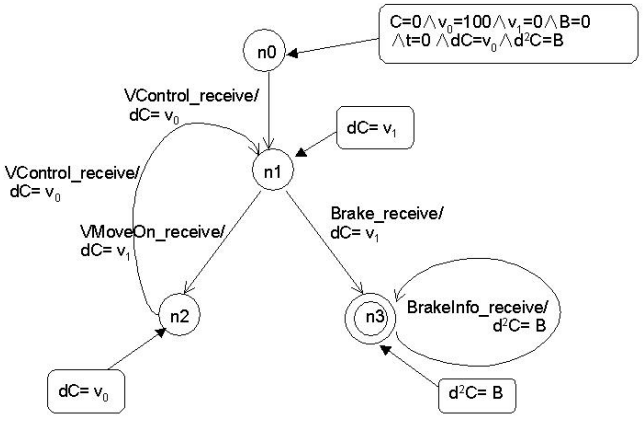

Figure 4: The DFA for business role *IDS*

5 The testing environment

IBM WebSphere Process Server (WPS) is an integration platform for Service Oriented Architecture (SOA), and supports the Service Component Architecture (SCA) programming model. As a process engine, WPS provides a hosting environment for business processes and several Web-based applications. These features of WPS match the structure and the requirements of continuous time SRML modules, so we choose WPS as the basic testing environment.

As the implementing tools, we combine WID with Matlab. In WID, an assembly editor is the central panel where service modules are built by creating and linking services components and service interfaces. A service component is specified with an implementation, some interfaces defining the inputs, outputs and faults of the service component, and zero or several references which denote the interface of other service components connecting to this service component. Service components can be implemented in various types, such as Java objects, business processes, and business state machines. We choose business state machines to implement our Traffic-Control module, because from Figure 5, we find that the hybrid automaton based DFA can be mapped to WID business state machines almost directly. Figure 5 shows the mapping relations between a continuous time SRML module and a WPS service module.

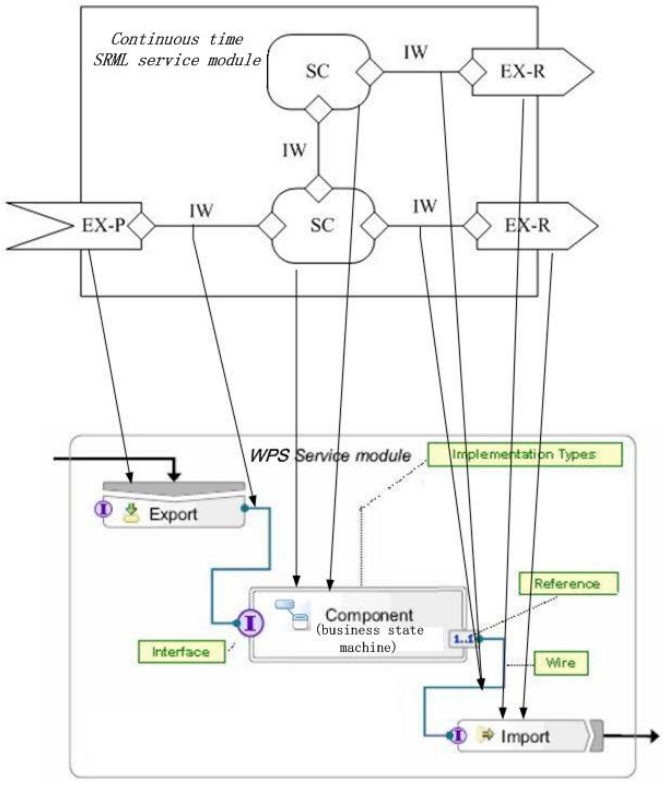

Figure 5: Mappings from continuous time SRML module to WPS service module

In a service component, the continuous processes described with differential equations can be directly implemented with Java code when there are exact solutions of the differential equations. Otherwise, we need to solve the differential equations numerically.

In such case, we combine WID with Matlab to achieve our goal. Matlab [9] is a multi-paradigm numerical computing environment, and it allows interfacing with programs written in languages such as C, C++ and Java. In the latest versions of Matlab, there is a function to wrap programs to Java packages. This function enables us to write Matlab programs to solve the differential equations in the service components in WID, and pass the solutions to the service components by importing the programs packed into Java packages to WID. We'll continue to do the implementation and testing in our future work.

6 Conclusion and future work

In this paper, we give a complete introduction to continuous time SRML, including the semantic domain of continuous time SRML, the application of continuous time SRML, and the testing environment. Continuous time SRML abstracts hybrid system-embedded service-oriented systems at a high level. The semantic foundation of continuous time SRML is SO-HL^{2}TS, over which this modelling language is interpreted. The transitions of SO-HL^{2}TSs enable continuous time SRML to model the communications between services or within a service; and the traces of states of SO-HL^{2}TSs enable continuous time SRML to model the continuous processes performed by service components. Systems modelled with continuous time SRML can be transformed to hybrid automata-like DFA for testing and verification. This can be demonstrated by the case study of the Traffic Control System. By combining IBM WID and Matlab, we are able to implement the hybrid automata-like DFA.

In our future work, we will show the implementation of the hybrid automata-like DFA for the Traffic Control model, and provide the result of testing. We will also try to verify the Traffic Control system with KeYmaera X, which is a newly developed model checking tool for hybrid systems. For the transformation presented in Section 4, we are also working on a systematic approach so that this can be done automatically by programs. Further, we will try to enrich continuous time SRML with the expressions of actions, so that models specified with continuous time SRML match the business state machines of IBM WID better.

References

[1] D. Georgakopoulos, M. Papazoglou, Service-Oriented Computing, The MIT Press, Cambridge, Massachusetts, 2009.

[2] J. Fiadeiro, A. Lopes, J. Abreu, "A formal model for service-oriented interactions," Science of Computer Programming, **77**, 577–608, 2012, doi:10.1016/j.scico.2011.12.003.

[3] J. Fiadeiro, A. Lopes, L. Bocchi, J. Abreu, "The Sensoria Reference Modelling Language," Lecture Notes in Computer Science, **6582**, 61–114, 2011, doi:10.1007/978-3-642-20401-2_5.

[4] M. Wirsing, M. Hölzl, Rigorous Software Engineering for Service-Oriented Systems, Springer, 2011.

[5] N. Yu, Injecting Continuous Time Execution into Service-Oriented Computing, Munich University , Ph.D. thesis, 2016.

[6] N. Yu, M. Wirsing, "The Application of Continuous Time SRML," in Proceedings of 2020 ICICSE&ICACTE, 99–103, 2020, doi:10.1145/3424311.3424316.

[7] J. Abreu, Modelling Business Conversations in Service Component Architectures,University of Leicester, Ph.D. thesis, 2009.

[8] IBM, IBM Integration Designer V8.5.5 documentation, https://www.ibm.com/support/knowledgecenter/SSTLXK_8.5.5/com.ibm.wbpm.wid.main.doc/kc-welcome.html, 2014.

[9] MathWorks, https://www.mathworks.com/,2014-2021.

[10] T. A. Henzinger, "The theory of hybrid automata," in Proceedings of 11th Annual IEEE Symposium on Logic in Computer Science, 1996, doi:10.1109/LICS.1996.561342.

[11] K. Cordwell, A. Platzer, "Towards Physical Hybrid Systems," in Proceedings of International Conference on Automated Deduction, CADE'19, 216–232, 2019, doi:10.1007/978-3-030-29436-6_13.

[12] A. Platzer, Logical Foundation of Cyber-Physical Systems, Springer,Cham, 2018.

[13] A. Platzer, KeYmaeraX: An aXiomatic Tactical Theorem Prover for Hybrid Systems, http://www.ls.cs.cmu.edu/KeYmaeraX/index.html, 2021.

[14] M. Fadlisyah, E. Ábrahám, P. C. Lepri, D. Ölveczky, "A Rewriting-Logic-Based technique for modelling Thermal Systems," in Proceedings of First International Workshop on Rewriting Techniques for Real-Time Systems, PTRTS 2010, 82–100, 2010, doi:10.4204/EPTCS.36.5.

[15] N. Yu, M. Wirsing, "A SOC-Based Formal Specification and Verification of Hybrid Systems," in Proceedings of the 22nd International Workshop on Algebraic Development Techniques (WADT2014), 151–169, 2015, doi:10.1007/978-3-319-28114-8_9.

[16] R. De Nicola, F. Vaandrager, "Three Logics for Branching Bisimulation," Journal of the ACM, **42(2)**, 458–487, 1995, doi:10.1145/201019.201032.

[17] A. Platzer, J. Ousel, "European Train Control System: A Case Study in Formal Verification," in Proceedings of 11th International Conference on Formal Engineering Methods (ICFEM 2009), 246–265, 2009, doi:10.1007/978-3-642-10373-5_13.

[18] The Worldwide Railway Organization (UIC), The European Train Control System, https://uic.org/etcs#Standards-and-Specifications, 2015.

[19] J. E. Hopcroft, R. Motwani, J. D. Ullman, Introduction to Automata Theory, Languages, and Computation (3rd Edition)., Addison-Wesley Longman publishing Co., Inc., USA., 2006.