A Framework for Data-Driven Automata Design

Yuanrui Zhang, Yixiang Chen(✉), and Yujing Ma

MoE Engineering Research Center for Software/Hardware Co-design Technology and Application, Software Engineering Institute, East China Normal University, Shanghai 20062, China
yxchen@sei.ecnu.edu.cn

Abstract. The traditional model-driven developing methods in requirement engineering (RE) have met challenges. Under the dropback of big data, we propose a new framework of software design method based on requirement data. Given a set of requirement data, which are usually acquired directly from users' intuitive descriptions about systems, we consider the method to analyze them and extract useful information from them in order to build the formal specifications of systems. Here the 'data' could be any form and could describe any prospect of functionalities of systems, the 'model' could be any types of formal models like process algebra, automata, or some forms of state-diagrams. We limit the scope of our discussion in this paper to a special type of data and formal models. We first use some simple examples to clarify the general idea of 'data-driven' we propose. Then as a case study, we apply this idea to the requirement specification of a special type of systems—spatial-temporal systems by proposing a special formal model in order to capture the spatial-time features of them.

Keywords: Requirement engineering · Data-driven · Big data · Automata theory · Spatial-temporal · Spatial hybrid automata

1 Introduction

In software engineering, as system is becoming more and more complex in size and logic, traditional methods for software developing like event-driven design or object-oriented design are becoming more and more difficult to meet our requirements. We need more convenient and intuitive way to design our software while the software we design should fully satisfy our requirements. Recently, big data has become a hot topic in computer science. As data sets grow rapidly in size, people start thinking about new ways of analyzing, capturing and extracting useful information from data sets. This big change lead us to think of a new way to develop our software as big data can also offer large information about the requirements of systems.

Many developing methods of software engineering have been investigated throughout the history. Niklaus Wirth gives the well-known formula: Program = Data structure + Algorithm. This formula has guided us many years to develop

L. Liu and M. Aoyama (Eds.): APRES 2015, CCIS 558, pp. 33–47, 2015.
DOI: 10.1007/978-3-662-48634-4_3

software. Later, one develops the formalization method to establish the (behavior) semantics and relations of (between) programs or data structure. Finite state machine is a classical method to describe such semantics and relations. As the development of large-scale and complex software, one explores the model-driven design framework. UML [1] is a typical platform of model-driven software development. Finite state machine is still the core of UML for description of translation among states.

Wikipedia [2] defines that data-driven programming is a programming paradigm in which the program statements describe the data to be matched and the processing required rather than defining a sequence of steps to be taken.

Wirfs-Brock and Wilkerson in [3] give the concept of data-driven design. They think that data-driven design is the result of adapting abstract data type design methods to object-oriented programming. The data-driven approach to object-oriented design focuses on the structure of the data in a system. This results in the incorporation of structural information in the definitions of classes.

Related to data-driven design, event-driven design is proposed and originated in the area of active databases in the 1980s with the introduction of triggers to database system [4][5]. A trigger is an Event-Condition-Action (ECA) rule which is checked by the DBMS whenever an event occurs that matches the Event part of the rule [6]. In event-based systems, an event can take many forms and data are abstracted as the event instance [7]. Miro Samek in [8] says that state machines are perhaps the most effective method for developing robust event-driven code for embedded systems.

Requirements engineering (RE) [9] refers to the process of defining, documenting and maintaining requirements and to the subfields of systems engineering and software engineering concerned with this process[10][11][12][13].

Formal method is an important method for requirement specification. We proposed a process algebra called STeC for specifying of real-time system with spatial temporal consistency requirements [14].

In this paper, we propose a new developing method for software engineering based on data sets. The core idea is that we can build the specifications of systems based on the data sets which offer requirement information about systems using some mechanism. To make things easier, we restrict our forms of data to be the sequence of actions that the system behave, our models to be the abstract automata. We give some easy-understanding examples to explain our ideas. The paper is organized as follows: in Section 2, we introduce our data-driven framework and basic concepts of each components of the framework. In Section 3, we give some simple examples of data domain and their relevant formal models using classical automata theories. In Section 4 we give a case study and explain how to build formal specifications for a spatial-temporal system based on spatial-temporal requirement data using our data-driven method. We introduce a type of state machine called spatial-hybrid automata as its formal model. At last, we make a conclusion in Section 5.

2 Framework for Data-Driven Automata Design

Fig. 1 gives the core idea of data-driven framework. The data offers the requirements of systems. It can give information about how a system behave, or for instance, some features about a system. The model gives the formal specifications of systems, it needs to be well chosen and generated based on the data. The traces (generated by the model) is analyzed if it fits our requirements. If not, more data is needed to give more concrete requirements. The key part of this procedure is the mechanism to choose and generate the formal model from data, which could be hard and varies from different data domains and requirements. In our paper, we only consider the data as some types of sequences of events, and models as some kinds of automata.

2.1 Data Type, Trace and Word

A data is a set of traces that describe a sequential of events of a system. Since the types of behaviours of systems vary from one to another, the type of data should also be different. For some systems we only consider the logical relationships between their actions, for some others, we also need to consider the time issues together with actions, and for the rest, we even consider their space relationships.

We next give a definition of a data, trace and word. A trace consists of a sequence of words. A word is a tuple of different elements that belong to different domains, which makes data distinguished from one to another.

Definition 21. *A word is a n-tuple* $w \in D_1 \times D_2 \times ... \times D_n$, D_i *(*$i \in \{1, 2, ..., n\}$*) is called data domain.*

Definition 22. *A trace is a finite or infinite sequence of words, denoted as* $Tr = w_1 w_2 ... w_n ...$ *where* $w_i \in D_1 \times D_2 \times ... \times D_n$ *for all* $i \in \{1, 2, ...\}$ *and some* $n \in \mathbb{N}^+$.

Definition 23. *A data is a set of traces for some type of word, denoted as* $Data = \{Tr \mid Tr = w_1 w_2 ... w_n ..., \ w_i \in D_1 \times ... \times D_n \ \forall i \in \mathbb{N}^+\}$.

The data domain D can be any mathematical structure, e.g, $\mathbb{R}$, $\mathbb{R}^+$, $\mathbb{N}$, any partial order sets, etc.

2.2 Languages and Automata

The language in computer science is undertood as a set of words, like data, in our example. So in the data-driven framework of automata design, we can equivalently treat data as languages, with different types of word (Def. 21). For some types of languages, for example, regular languages [16], the mechanism of generating the corresponding automata is determined. That is, for any regular languages, its generated model can fully fit it. For example, determined finite automata (DFA) can be generated from regular languages [17], timed automata [18] can be generated from timed regular expressions [19], hybrid

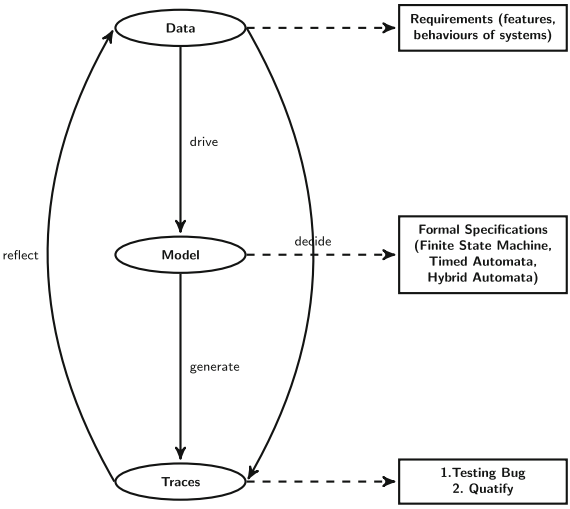

Fig. 1. The Data-Driven Framework of Software Development

automata [20] can be generated from hybrid traces, etc. The same for the context-free languages or ω-languages. However, not all languages can be easily translated into automata, for example, timed non-regular languages can not be translated into a timed automata. Also, for the languages that contains an informal type of words (not the word that can be accepted by automata), the algorithms for translations is not direct. In Section 4, we build a type of automata called spatial-hybrid automata for the model of spatial-temporal data types based on hybrid automata.

3 Some Examples of Data Domain

We give some examples [1] of different types of data and their generated formal specifications. We firstly introduce a simple real-time system, called 'Railroad Crossing System' (abbreviated as RCS), as an example. Then pick up some different types of data as examples based on different behaviours of this system.

3.1 Railroad Crossing System

Railroad Crossing System (or RCS) (shown in Fig. 2) is a dynamic scheduling system at a crossing. There are two agents: a train and a gate. The gate is at the crossing road where there is a railroad in east-west direction and a road in

[1] Some early examples of data-driven languages are the text-processing languages sed and AWK [15], where the data is a sequence of lines in an input stream ?C these are thus also known as line-oriented languages ?C and pattern matching is primarily done via regular expressions or line numbers.

south-north direction. The train communicates with the gate dynamically when attending to pass the crossing. The gate keeps opened when no train coming, and has to close itself if there is a train coming. We summarize the system scenario and list the behaviours of the train and the gate respectively as below:

RCS System Scenario:

- Train passes through a crossroad which installs a gate for safety.
- Normally, the gate is open so that cars and people may pass the railroad. But, if a train passes through this crossroad then the gate must be closed.
- This control between gate and train will be completed through sending and getting messages each other.

Train Behaviour:

- Train sends a message **App** to gate at the location **Lapp** and then goes to gate;
- At location **Lpass**, if the train gets the message **Cross** from gate then it passes through the gate and sends message **leaving** to gate at the location **Lleav**.
- Otherwise, the train must stop at the location **Lstop** and waits for the message **Cross** there;

Gate Behaviour:

- The gate is open for cars and people to pass through railroad.
- If the gate gets the message **App** then closes;
- If the gate is closed then sends a message **Cross** to train;
- When the gate gets a message **leaving** and does not get the message **App** then it opens.

Next we extract part of their behaviours and represent them as different types of data.

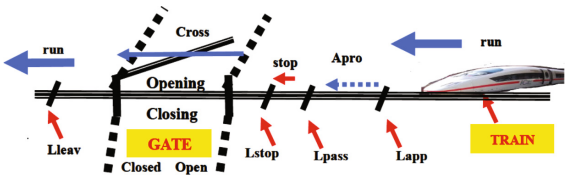

Fig. 2. Railroad Crossing System

3.2 The Data

Untimed Data. The untimed data only captures logical relationships between actions.

For example, suppose we capture the action 'open' of the gate of RCS system as a sequences, using the data defined as follows:

$$Data_{open} = \{Tr \mid Tr = \langle open \rangle^*\}$$

Tr is a trace. It is a regular languages, so the mechanism from it to its formal model is determined. Fig. 3 shows its corresponding automata.

Similarly, we can capture and generate the behaviours of sequences of 'open-close' of the gate, like:

$$Data_{OC} = \{Tr \mid Tr = (\langle open \rangle \langle close \rangle)^*\}$$

Fig. 4 shows its corresponding automata.

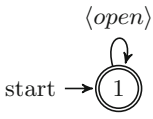

Fig. 3. Automata accepting the trace 'door open'

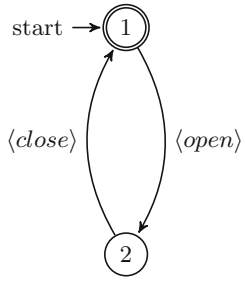

Fig. 4. Automata accepting the trace 'door open-close'

Timed Data. Timed data contain information about time concerning the point at which actions happen or how long actions would take through. For example, we now capture the sequences of actions of train 'stop-run' with a two-tuple arrays of data:

$$Data_{SRT} = \{Tr \mid Tr = (\langle stop, \text{ duration}= 1m \rangle \langle run, \text{ duration}= 5m \rangle)^*\}$$

We suppose that the train 'stop' for 1 minutes, and keep 'running' for 5 minutes...(this behaviour may not true for the scenario described above in Section 3.1, it is just an example! The same in the rest of paper). $Data_{SRT}$ contains a domain representing the duration of one action. Though it is not a formal timed language for timed automata, we can translate it into a timed language:

$$L = \{trace \mid trace = \langle stop', 1 \rangle \langle run', 6 \rangle \langle stop', 7 \rangle \langle run', 13 \rangle ...\}$$

Where we take $stop'$ as an instant action, meaning the finishing of the action $stop$, the same as run'. We can proof that such timed language L is a timed regular expression so we generate a timed automata shown as Fig. 5, we translate the 'duration' to a clock 'x' in timed automata.

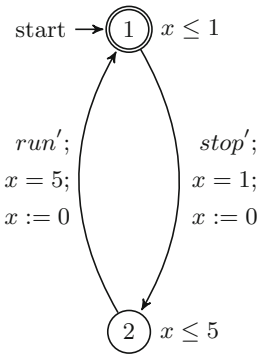

Fig. 5. Timed Automata accepting the trace 'train stop-run'

Spatio-Time Data. Spatio-time data not only have time issues, but also space issues. They record behaviours that concern both time and locations.

Suppose that the train in RCS system approach the gate at time 14:34, location 'Lapp', taking 2 minutes, then arrive at location 'Lpass' at time 14:36. It then take 8 minutes to stop at time 14:44. After waiting for 2 minutes, the train passes the gate at time 14:46, taking 2 minutes. We give the corresponding data as follows:

'Train Passing' spatio-time data:

$$Data_{TP} = \{Tr\},$$

where

$$Tr = \langle\ appr, \text{location}= Lapp, \text{time}= 14:34, \text{duration}= 2m\rangle\ \langle\ stop, \text{location}= Lpass, \text{time}= 14:36, \text{duration}= 8m\rangle\ \langle\ wait, \text{location}= Lstop, \text{time}= 14:44, \text{duration}= 2m\rangle\ \langle\ pass, \text{location}= Lstop, \text{time}= 14:46, \text{duration}= 2m\rangle$$

See that $Data_{TP}$ is not in a form that can be accepted by any kinds of automata. So we need to make a translation, we translate data $Data_{TS}$ into a timed trace and a propositional trace of timed automata as follows:

$$trace = \langle\ appr, 14:34\ \rangle\ \langle\ stop, 14:36\rangle\ \langle\ wait, 14:44\rangle\ \langle\ pass, 14:46\rangle\ \langle\ \emptyset, 14:48\rangle$$
$$proptrace = \langle\emptyset\rangle\ \langle Lapp\rangle\ \langle\ Lpass\ \rangle\ \langle\ Lstop\ \rangle\ \langle\ Lstop\ \rangle\ \langle\ \emptyset\ \rangle$$

we now can generate timed automata according to them by adding propositions on each state representing the locations. Fig. 6 is the corresponding timed automata, the data 'location' is translated to atomic propositions in timed automata.

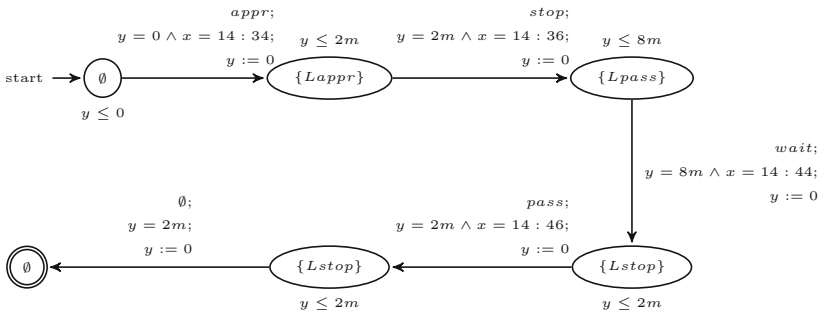

Fig. 6. Timed Automata accepting the trace 'Train Passing'

Hybrid Data. Hybrid data describes the behaviours with time issues and the change rate of variables.

Suppose that after the gate receive the message 'App' from the train, it will process for 2 seconds, then it will close itself for 30 seconds, with the rate of 3 degrees per second. After getting the message 'Leave' from the train for 2 seconds, it will open itself again for 30 seconds, with the same rate. We give the data as follows:

'Gate Close-Open' hybrid data:

$$Data_{GCO} = \{Tr \mid Tr = (\langle App?, \text{duration}= 2s\rangle\ \langle close, \text{angle0}= 90d, \text{angle1}= 0, \text{rate}= -1\rangle\ \langle open, \text{angle0}= 0d, \text{angle1}= 90d, \text{rate}= 1\rangle\ \langle Leave?, \text{duration}= 2s\rangle\ )^*\}$$

We can translate the data into a hybrid trace and a differential equation trace as follows:

$$trace = \langle App'?, a = 90, t = 0\rangle\langle close', a = 90, t = 2\rangle\langle\emptyset, a = 90, t = 30\rangle\langle Leave'?, a = 0, t = 30\rangle\langle open', a = 0, t = 2\rangle\langle\emptyset a = 90, t = 30\rangle...$$
$$vftrace = \langle\dot{a} = 0, \dot{t} = 1\rangle\langle\dot{a} = 0, \dot{t} = 1\rangle\langle\dot{a} = -1, \dot{t} = 1\rangle\langle\dot{a} = 0, \dot{t} = 1\rangle\langle\dot{a} = 0, \dot{t} = 1\rangle\langle\dot{a} = 1, \dot{t} = 1\rangle\langle\dot{a} = 0, \dot{t} = 1\rangle...$$

Variable t records time eclapse and d records the angle value. Fig. 7 shows the generated automata.

4 Case Study—Data-Driven Method for Spatial-Temporal System

In this section, we give an example of building a formal specification for a special type of system—spatial temporal system based on a type of spatial-time data. We propose a formal model called 'spatial-hybrid automata for the specification.

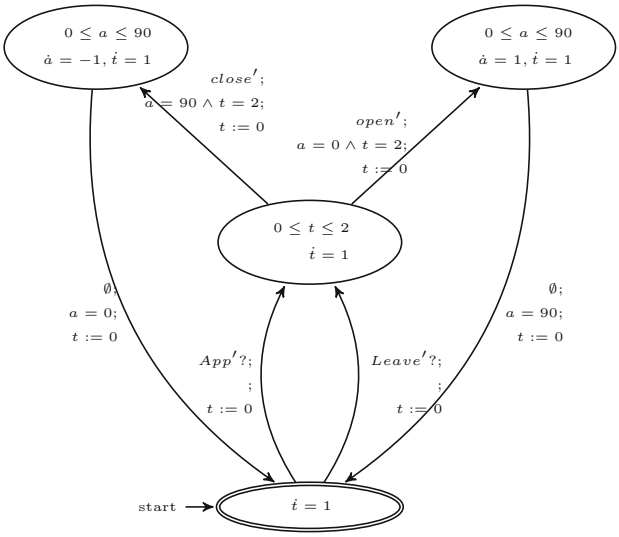

Fig. 7. The Hybrid Automata of 'Gate Close-Open'

4.1 A Counter Example

From Section 3, we show some examples of data which can generate an automata. Now consider an example of data which can not (or say not suitable) generate an automata. We change $Data_{TP}$ into a new form as:

$$Data'_{TP} = \{Tr\}$$

where

$$Tr = \langle\ appr,\ \text{l}= 1km,\ \text{time}= 14:34,\ \text{duration}= 2m\rangle\ \langle\ stop,\ \text{l}= 0.8km,\ \text{time}= 14:36,\ \text{duration}= 8m\rangle\ \langle\ wait,\ \text{l}= 0.3km,\ \text{time}= 14:44,\ \text{duration}= 2m\rangle\ \langle\ pass,\ \text{l}= 0.3km,\ \text{time}= 14:46,\ \text{duration}= 2m\rangle$$

We replace the locations of each word in $Data'_{TS}$ with an exact number l. So the data turns into a hybrid data we can translate it into a hybrid trace as follows:

$$trace = \langle\ appr,\ T = 14:34, l = 1, d = 2\ \rangle\ \langle\ stop,\ T = 14:36, l = 0.8, d = 8\rangle\ \langle\ wait,\ T = 14:44, l = 0.3, d = 2\rangle\ \langle\ pass,\ T = 14:46, l = 0.3, d = 2\rangle\ \langle\ \emptyset,\ T = 14:48, l = 0.3, d = 2\rangle$$

$$trace = \langle stop, T = 14:34, l = 10, d = 2\rangle\langle run, T = 14:36, l = 10, d = \delta_1\rangle\langle stop, T = 15:10, l = 25, d = 2\rangle\langle run, T = 15:12, d = \delta_2\rangle\langle stop, T = 16:15, l = 50, d = 2\rangle ...$$

Now the problem rises, since the data does not specify the change rate of l, we can not generate a hybrid automata for it. It tells us we need to introduce

an 'abstract' automata that only give the conditions of each edge but leave the different equations free unknown. We call it 'spatial hybrid automata' because it leave locations as a condition whose domain varies from different types of data we choose for it.

Indeed, we are interested in a special type of systems called 'spatial hybrid systems, as introduced below.

4.2 Spatial Temporal System

A type of software systems is called spatial hybrid system, the examples of this kind of systems are like Internet of Things and Cyber-Physical Systems. In their behaviours, both the time point at which an action occurs and the location at which an action stays count. Consider the 'Railroad Crossing System' (RCS) in Section 3, in such a system, the train speed depends on the environment (for example, the weather, the traffic condition or the driver), considerations which are partially ignored in a high-level model. However, the train is expected to communicate with the gate at some specific locations on the track. So is the gate, which should respond before the train reaches a given point, if to avoid any accident and have an 'optimal' use. For this type of systems, we propose a hybrid automata in a special domain—'space' domain based on hybrid automata to capture their behaviours.

4.3 Spatial Hybrid Automata

As an abstraction of hybrid automata, spatial hybrid automata add a set of special conditional triggers called 'locations', and reduce the expressive power to be weaker than hybrid automata. Such automata are good for abstracting the spatial-temporal features of spatial-temporal systems, and on the other hand though it loses the power of hybrid automata, but can be easily refined to be concrete hybrid automata.

In spatial hybrid automata, we stress that the trigger of each edge is depending on not only the conditions of variables, but also the conditions in the special triggers we define—'locations'. We take differential equations in each state as parameters and thus they are not certain. In other words, we only give the conditions ('location') when each edge can be triggered and do not tell the exact form of differential equations for each state to reach those conditions. We remain the time variables and continuous variables the same in hybrid automata, but adding our set—'locations' as part of guard conditions. The formal definition is as follows:

Definition 41. *A Spatial hybrid automaton* sHa *is a collection* $(Q, X, L, M, \Sigma, F, Init, D, E, G, R)$, $(q, x) \in Q \times X$ *as the state of* sHa *denoted as* St, *where*

1. $Q = \{q_1, q_2, \ldots\}$ *is a set of discrete states;*
2. $X = \mathbf{R}^n$ *is a set of continuous states;*
3. $L = \{l_1, l_2, \ldots, l_n\}$ *is a finite set of locations,* $l_i \in P(X)$ *for any* $1 \leq i \leq n$ *(*$P(X)$ *denotes the power set of* X*).*

4. $M = \{m_1, m_2, \ldots\}$ *is a set of messages;*
5. $\Sigma \subseteq \{ch?, ch!\} \times M \cup Act$ *is the set of events. Act is normal events.* $\{ch?, ch!\} \times M$ *is the synchronizing events.*
6. $Init \subseteq Q \times X \times P(F)$ *is a set of initial states;*
7. $D(\cdot) : Q \to P(X)$ *is a domain;*
8. $E \subseteq Q \times Q$ *is a set of edges;*
9. $G(\cdot) : E \to L \times \Sigma \times P(X)$ *is a guard condition;*
10. $R(\cdot, \cdot) : E \times X \to P(X)$ *is a reset map for a jump.*
11. $F : Q \to P(Q \times X \to R^n)$ *is the set of all vector fields for each state that satisfy all locations and guards of edges starting from this state (where* $f(\cdot, \cdot) : Q \times X \to R^n$ *is a vector field). For any* $q_1 \in Q$, $F(q_1) = \{f \mid (\forall (q_1, q_2) \in E)(\forall (l, a, g) \in G((q_1, q_2)))(\exists t \in \mathbb{R}^+).(\dot{x} = f(q_1, x) \wedge x(t) \in l \cap g\}$;

The different from the definition of hybrid automata is that we define a set of differential equations for each state that satisfy all guards of each edge from this state. We also add location set L as a field of locations. A location can be any physical locations like $\mathbb{R}$, $\mathbb{R}^2$, etc.

We say a differential equation $\dot{x} = f(q, x)$ satisfies a guard g, for example, we mean that there exists a time point t such that $x(t) \in g$. For each state q, there exists a set of differential equations $F(q)$ that satisfy all its guard conditions ($F(q)$ might be $\emptyset$ if no guards can be satisfied). This shows that in our spatial-hybrid automata we do not care the exact differential equation that trigger each edge. We do not care how to achieve the edge conditions, but the edge conditions itself.

4.4 Semantics of Spatial Hybrid Automata

Like hybrid automata, we give the transition semantics of spatial hybrid automata. The transition relation of spatial hybrid automata, as hybrid automata, constitutes of the labelled transition relation and the continuous transition relation.

Labelled Transition Relation. A triple $\to \subseteq St \times \Sigma \times St$ is called a labelled transition relation. We write $s \xrightarrow{a} s'$ if $(s, a, s') \in \to$.

Continuous Transition Relation. A triple $\Longrightarrow \subseteq St \times \mathbb{R}^+ \times St$ is called a continuous transition relation. We write $s \xRightarrow{\delta} s'$ if $(s, \delta, s') \in \Longrightarrow$.

Definition 42 (Transition Semantics). *Let* $sHa = (Q, X, L, M, \Sigma, F, Init, D, E, G, R)$ *be a spatial hybrid automata, the transition relation is given as follows:*

i. ***State****. For any* (q_0, x_0), $(q_0, x_0) \in St$ *iff* $x_0 \in D(q)$ *and* $F(q_0) = \{f \mid (\forall (q_0, q) \in E)(\forall (l, a, g) \in G((q_0, q)))(\exists t \in \mathbb{R}^+).(\dot{x} = f(q_0, x) \wedge x(t) \in l \cap g\}$.

ii. ***Labelled Trans****. For any* $s = (q_0, x_0)$, $s' = (q_1, x_1) \in St$, $a \in \Sigma$, $s \xrightarrow{a} s'$ *iff* $(q_0, q_1) \in E$, $(x_0, a, x_0) \in G((q_0, q_1))$ *and* $x_1 \in R((q_0, q_1), x_0)$.

iii. ***Continuous Trans****. For any* $s = (q, x_0), s' = (q, x_1) \in St$, $\delta \in \mathbb{R}^+$ *and* $f \in F(q)$, $s \xRightarrow{\delta} s'$ *iff* $x_1 = x_0 + \int_{t_0}^{t_0+\delta} f(q, x)dt$ *for some* t_0.

The spatial hybrid automata allow all continuous transitions with the differential equations ranging in set $F(q)$ for each state q. While the definition of state and labelled transitions remain the same as hybrid automata.

4.5 Formal Model of $Data'_{TS}$

Given the data $Data'_{TP}$ in Section 4.1, we can build a spatial-hybrid automata sHa_{TP} shown in Fig. 8.

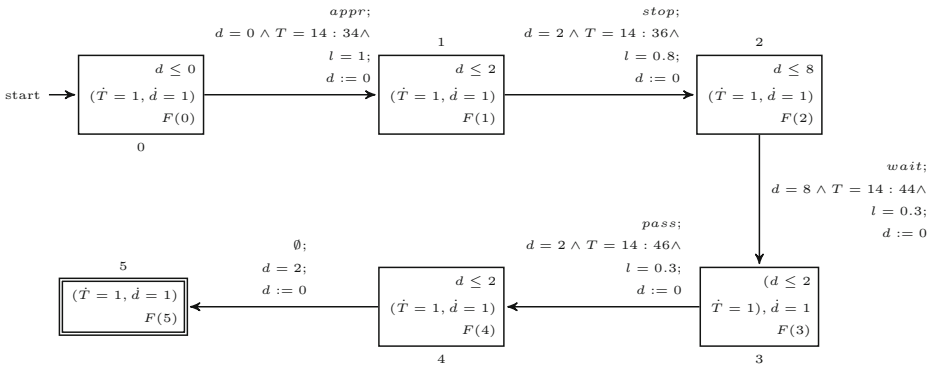

Fig. 8. Spatial-hybrid automata for $Data'_{TP}$

$sHa_{TP} = (Q, X, F, L, M, \Sigma, f, Init, D, E, G, R)$, where

- $Q = \{0, 1, 2, 3, 4, 5\}$;
- $\mathbf{X} = \mathbb{R}^3$, since there are 3 variables: T, d, l;
- $L = \{l = 1, l = 0.8, l = 0.3\}$ are the location conditions;
- $M = \emptyset$;
- $\Sigma = \{appr, stop, wait, pass\}$ is the set of alphabets;
- $Init = \{(0, x)\}$, where $x = [T = 14 : 34, d = 0, l = 0]$;
- D is the domain of invariants. e.g, $D(0) = (d \le 0)$, $D(2) = (d \le 8)$;
- $E = \{(0, 1), (1, 2), (2, 3), (3, 4), (4, 5)\}$ is the set of edges;
- G is guard condition. e.g, $G((2, 3)) = (l = 0.3, wait, d = 8 \wedge T = 14 : 44)$;
- R is a reset map. e.g, $R((2, 3), d) = 0$;
- F is the set of vector fields for each state. e.g, $F(0) = F_0 = \{f \mid (\exists t \in \mathbb{R}^+).(\dot{T} = f(0, T) = 1 \wedge \dot{d} = f(0, d) = 1 \wedge \dot{l} = f(0, l) \wedge d(t) = 0 \wedge T(t) = 14 : 34 \wedge l(t) = 1)\}$;

Compared with the automata of $Data_{TP}$ (Fig. 6), in sHa_{TP}, each variable is not only driven by one differential equation in each state, but a set of differential equations. Each edge is triggered by not only guards but also locations (L).

4.6 Condition-Triggered Transition System

In Section 4.3 we propose a 'location triggered' hybrid automata, it is an abstract automata where in each state, a set of differential equations (not only one) satisfy all guards and locations of edges out of this state. The differential equations that satisfy the guards are depending on the domain of 'locations', just as the $Data_{TS}$ and $Data'_{TS}$ shown above. More generally, we can define an abstract transition system called 'condition-triggered transition system', where each edge is triggered by one or several conditions with specific domains. See the next definition:

Definition 43. *A condition-triggered transition system* $ctTS$ *is a collection* $(Q, X, C, \Sigma, F, Init, D, E, G, R)$, $(q, x) \in Q \times X$ *as the state of* $ctTS$, *where*

1. $Q = \{q_1, q_2, \ldots\}$ *is a set of discrete states;*
2. $X = \mathbf{R}^n$ *is a set of continuous states;*
3. $C = \{c_1, c_2, \ldots\}$ *is a finite set of conditions;*
4. Σ *is set of alphabets.*
5. $Init \subseteq Q \times X \times P(F)$ *is a set of initial states;*
6. $Dom(\cdot) : Q \to P(X)$ *is a domain;* ($P(X)$ *denotes the power set of* X)
7. $E \subseteq Q \times Q$ *is a set of edges;*
8. $G(\cdot) : E \to C \times \Sigma \times P(X)$ *is a guard condition;*
9. $R(\cdot, \cdot) : E \times X \to P(X)$ *is a reset map for a jump.*
10. $F : Q \to P(Q \times X \to R^n)$ *is the set of all differential equations that satisfy all guards and conditions in* C. *For any state* $q_1 \in Q$, $F(q_1) = \{f \mid (\forall (q_1, q_2) \in E)(\forall (c, a, g) \in G((q_1, q_2)))(\exists t \in \mathbb{R}^+).(\dot{x} = f(q_1, x) \wedge x(t) \in c \cap g\}$;

It is almost like a hybrid automata except that we take each condition c in C as a variable, it determines the set of differential equations $F(q)$ for each state q. We omit the discussion about it.

5 Conclusion

In this paper, we are trying to give a new developing method for software engineering by which somehow to solve the crucial problems existed in the field of nowadays software engineering. We mainly propose a data-driven framework and give some examples to explain our idea. We also give a case study of building the formal model of a special type of systems—spatial-temporal systems using our proposed framework.

The core idea of the data-driven framework that distinguishes it from other methods is that we want to design models through the data, since data is the most intuitive for human beings and somehow easy to acquire. In this paper we only show some example of how data and their relative models (might) look like, and choose a formal model for a special type of data. We have no idea about the way of generating model from data. It is a complex problem, and various from one another

according to different types of data, different user requirements and some other factors. For the future work, maybe we will focus on building some concrete systems (under well-defined circumstances and clear users' requirements) and trying to find concrete algorithms to deal with the generating procedure.

Acknowledgment. This work is supported by the National Basic Research Program of China (No. 2011CB302802), the National Natural Science Foundation of China (No.61370100 and No. 61321064), Shanghai Knowledge Service Platform Project (No.ZF1213) and Shanghai Municipal Science and Technology Commission Project (No.14511100400).

References

1. Rumbaugh, J., Jacobson, I., Booch, G.: Unified Modeling Language Reference Manual, 2nd edn. Pearson Higher Education (2004)
2. Data-Driven: https://en.wikipedia.org/wiki/Data-driven_programming
3. Wirfs-Brock, R., Wilkerson, B.: Object-oriented design: a responsibility-driven approach. In: The Proceeding OOPSLA 1989 Conference Proceedings on Object-oriented Programming Systems, Languages and Applications, pp. 71–75. ACM, New York (1989)
4. Widom, J., Ceri, S. (eds.): Active Database Systems: Triggers and Rules for Advanced Database Processing. Morgan Kaufmann, San Francisco (1994)
5. Paton, N.W. (ed.): Active Rules in Database Systems. Springer, New York (1999)
6. Helmer, S., Poulovassilis, A., Xhafa, F.: Reasoning in Event-Based Distributed Systems. Springer, Heidelberg (2011)
7. Roussos, G.: Networked RFID: Systems, Software and Services. Springer, London (2008)
8. Samek, M.: State Machines for Event-Driven Systems (2003). http://www.barrgroup.com/Embedded-Systems/How-To/State-Machines-Event-Driven-Systems
9. Nuseibeh, B., Easterbrook, S.: Requirements engineering: a roadmap (PDF). In: Proceedings of the Conference on the Future of Software Engineering, ICSE 2000, pp. 35–46 (2000). doi:10.1145/336512.336523
10. Chemuturi, M.: Requirements Engineering and Management for Software Development Projects (2013). doi:10.1007/978-1-4614-5377-2. ISBN: 978-1-4614-5376-5
11. Kotonya, G., Sommerville, I.: Requirements Engineering: Processes and Techniques. John Wiley & Sons, September 1998. ISBN 0-471-97208-8
12. Thayer, R.H., Dorfman, M. (eds.): Software Requirements Engineering, 2nd edn. IEEE Computer Society Press (1997). ISBN 0-8186-7738-4
13. Mu, K., Kiu, W., Jin, Z., Lu, R., Yue, A., Bell, D.: Handing Incosistency in Distributed Software Requirements Specification Based on Prioritized Merging. Fundamenta Informaticae **91**(3–4), 631–670 (2009)
14. Chen, Y.: Stec: A location-triggered specification language for real-time systems. In: ISORC Workshops, pp. 1–6. IEEE (2012)
15. Stutz, M.: Get started with GAWK: AWK language fundamentals. developerWorks. IBM (retrieved October 23 2010)
16. Hopcroft, J.E., Motwani, R., Ullman, J.D.: Introduction to Automata Theory, Languages, and Computation, 3rd edn. Addison-Wesley Longman Publishing Co., Inc., Boston (2006)

17. Rabin, M.O., Scott, D.: Finite automata and their decision problems. IBM J. Res. Dev. **3**(2), 114–125 (1959)
18. Alur, R., Dill, D.L.: A theory of timed automata. Theor. Comput. Sci. **126**(2), 183–235 (1994)
19. Asarin, E., Caspi, P., Maler, O.: A kleene theorem for timed automata. In: LICS. IEEE Computer Society, pp. 160–171 (1997)
20. Alur, R., Courcoubetis, C., Henzinger, T.A., Ho, P.-H.: Hybrid automata: an algorithmic approach to the specification and verification of hybrid systems. In: Hybrid Systems, pp. 209–229. Springer, London (1993)