International Journal of u- and e- Service, Science and Technology
Vol.9, No. 7 (2016), pp.335-348
http://dx.doi.org/10.14257/ijunesst.2016.9.7.34

Timed Model Checking Service-Oriented Product Lines

Hongxia Zhang[1] and Fei Wang[2]

[1]*College of Computer & Communication Engineering, China University of Petroleum,Qingdao, P. R. China*
[2]*School of Economics & Management,China University of Petroleum Qingdao, P. R. China*
[1]*zhanghx@upc.edu.cn*, [2]*dyqwzzb917@163.com*

Abstract

Modeling and verification of behavioral variability in service-oriented product line descriptions is very important. Time is a pivotal parameter in representing behaviors of services, which is difficult for scalable modeling and efficient automated checking. In this paper, we describe an approach for modeling and efficient verification of service-oriented product lines. Firstly, we propose feature timed automata, formalism designed to describe the combined behavior of a timed service family, which leverages on establishing relationships between features and transitions. Then, we present feature computation tree logic to model timed properties of service families, which extend computation tree logic to take into account the products constraints. Finally, using our methodology, we analyze a travel agency service family using the model checker UPPAAL. The experiment shows that our approach helps to verify timed service families effectively in the early phases of development.

Keywords*-model checking, service-orientated product lines, timed automata, behavioral modeling, Computation Tree Logic*

1. Introduction

Business environments command innovation, increasingly shorter time-to-market and efficiency. Web services are the main pillar of the Service Oriented Computing (SOC) paradigm which enables the application integration within and across business organizations [1]. Different consumers often have different requirements, which will lead to an exponential number of services. It is unrealistic to model and verify the behavior of each service individually. For this reason, services in SOC often need to be designed to serve a wide variety of service requirements.

For example, a travel reservation service is used by a multitude of travel agencies, as well as numerous portal sites for displaying information. In many cases the requirements are slightly different from one user to another. To facilitate the development, deployment and consumption of services in such situations, variability needs to be systematically identified and managed.

For modeling and verifying such variability, recent research initiatives identified a promising synergy in a combined use of Software Product Lines (SPLs) and Service Oriented Architectures (SOAs), namely Service-Oriented Product Lines (SOPLs) [2-3], which enable organizations to produce service-oriented systems faster, cheaper and customizable to specific customers. Stemmed from the definition of software variability, service variability was defined as “the ability of a Web service to be efficiently extended, changed, customized, or configured

ISSN: 2005-4246 IJUNESST

for use in a particular context" [4]. Service variability can be considered as the specialization of software variability in the domain of service computing. The various service variants can be derived from the basic service family, which creates the opportunity to reuse and differentiate on products in the family.

In addition to inherited characteristics from SPLs, SOPLs have strong time constraints in many cases as mentioned above. It is important to ensure the correctness of services with time constraints which are related to some serious problems, such as bottlenecks and deadlocks [5]. However, time constraints have not yet been considered in SOPLs, hence the problem of analyzing the variability of timed services in SOPLs is still open. To overcome such limitations, in this paper, when analyzing services, the time constraints are taken fully account for and corresponding temporal properties is set up to verify the model effectively.

On the other side, it is commonly agreed that the variability of service behaviors is reflected in execution of transitions. There are some works that tried to consider variability in transitions through transitions classification [6] or variant points [4]. Actually, the differences among the products of a service family are typically expressed in terms of features, which are first-class abstractions that shape the reasoning of the engineers and other stakeholders. The existing works ignore the relationship between features and transitions of a service, which can support the mapping of service variants to the corresponding transitions in behavioral models. Therefore, they capture different behaviors, but offer little to products and their behavioral descriptions. Moreover, none of the proposals provide methods for model checking the behaviors rely on features or products.

To address these challenges, we recently launched a research effort to, on the one hand, investigate the most promising existing modeling structures that allow (behavioral) variability to be described with time constraints, on the other hand, survey a proper temporal logic that can express interesting properties especially time properties over service families. Our aim is the development of rigorous modeling techniques as well as analysis and verification approach for SOPLs.

In this paper, we propose feature Timed Automata (FTA) for specifying variable behaviors in service families based on Timed Automata, which has been already used in a series of papers. To express timed properties, we rely on clocks as defined in standard timed automata. Besides, FTA has a parameterized semantics that permits to obtain the behavior of each service of the SOPL through features. The priorities of transitions are also considered. Then, product CTL (pCTL) was defined based on Computation Tree Logic (CTL), which allows to model checking of temporal properties both on families and products. pCTL makes products dimension available. To express products, the encoding set of products is introduced. Finally, we apply our approach to model checking variability in behavioral descriptions of a particular service family. In order to verify the soundness of method, we apply it to a travel agency service family and compare with other methods based on the model checker UPPAAL [7] which has been used widely to analyze timed web services.

The reminder of the paper is organized as follows. Section II presents a travel agency case study that we use to show the related issues of the proposed approach. We formally define FTAs in section III. In section IV, we introduce pCTL and show how it can be used to define service properties and manage variability. Sections V discuss how to implement modeling and model checking a family of services we propose by UPPAAL. In section VI, we discuss related work. Finally, in section VII we conclude.

2. Running Example: Travel Agency

A. Service Requirements Descriptions

As a motivating example, consider a software company developed a package on sale for those interested in starting a travel agency service (*e.g.* on the web). This company provides a choice among travel agency services of a family with different features. All products provide the features including train reservation, flight reservation and payment. These products can be enhanced in these ways:

1) Some travel agency services may provide hotel reservations. Travelers can only reserve train, flight or hotel also can reserve hotel according to the information provided by train or flight reservation.

2) Some travel agency services access by cell phones, computers, or both of them.

3) Some travel agency services provide payments via credit cards or mobile phones which requires to accessing by a mobile phone.

This list of requirements contains functional constraints (part of 1, 2 and 3) defining the differences between products, as well as temporal ordering (part of 1, and 3).Besides the functional requirements, these products must satisfy time constraints as below:

1) Reservation orders should be confirmed within 20 time units;

2) Confirmed orders must be paid in 60 time units, otherwise will be canceled.

B. 2.2 Requirements Definition in Feature Model

Feature Diagrams are the first level abstraction of requirements, which provide help for understanding users' requirements effectively. Skipping the details, we informally recall the definition of feature diagrams (FDs) as below.

Definition 1 (Feature Diagrams) A feature diagram is a 3-tuple $FD = (F, r, DE)$, where F is a set of features, r is the root, and r is a mandatory feature, $DE \subseteq F \times F$ is the set of decomposition edges between features.

The semantics of an FD d, denoted $\llbracket d \rrbracket_{FD}$, is its valid feature groups, named products, *i.e.*, a set of sets of features: $\llbracket d \rrbracket_{FD} \subseteq \mathrm{P}(F)$.

A complete formal definition of FDs can be found in [8].

Based on the function constraints, we acquire the feature model of a family of travel agency services by a feature diagram (FD) is shown in Figure 1. Basically, this FD formally describes the set of travel agency variants. The semantics of the travel agency FD (shown in Figure 1) is shown in Table 1, which contains 10 different products. These products correspond to 4 different behaviors according to temporal ordering. The number of behavior models is less than the number of products, because in travel agencies, the business process does not relate to which kind of media they use, so the behavior of using mobile phones is the same as using computers. But in some cases, the variant of functions is lead to the variant of behaviors directly. This means that the behavior model of this minor example such as this would already require some identical, behavioral descriptions. For realistic cases, this number will be so high that it is outright impossible to verify, l*et al*one model, each single service individually.

Therefore, it is necessarily to provide a unified method for modeling and checking these service behaviors as FD for functional requirement description, which will reduce service orchestration complexity and improve detection rate effectively

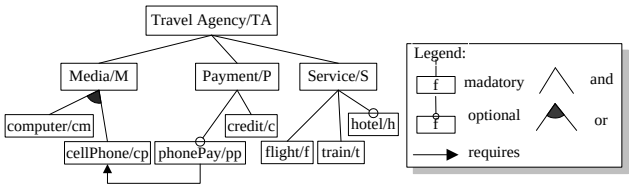

Figure 1. FD for Family of Travel Agency Services

3. Feature Timed Automata

Timed automata (TA) are now an accepted formal model for defining the timed behavior of web services. In order to model the behavior of each product in one model concisely, we draw upon existing approaches that create a single parameterized model to satisfy the service family. We will formalize such variability by means of combining an associated set of logical formulae with actions in a TA, named feature timed automata (FTA), and product Computation Tree Logic (pCTL) will be presented in Section IV.

Table 2. Products for Family of Travel Agency Services Figure 1

					products (*pid*)									
feature			*fid*	variability	1	2	3	4	5	6	7	8	9	10
Travel Agency **/TA**	Service/S	train reservation **/t**	5	mandatory	√	√	√	√	√	√	√	√	√	√
		flight reservation **/f**	4	mandatory	√	√	√	√	√	√	√	√	√	√
		hotel reservation **/h**	6	optional						√	√	√	√	√
	Media/M	cell phone **/cp**	1	alternative	√		√	√	√	√		√	√	√
		computer **/cm**	0			√	√		√		√	√		√
	Payment/P	phone pay **/pp**	2	optional				√	√				√	√
		credit pay **/c**	3	mandatory	√	√	√	√	√	√	√	√	√	√

A. Timed Automata

In this paper, timed behaviors of individual products are represented with Timed Automata, which is a transition system extended with conditions over and resets of non-negative real-valued clock variables. The extension makes it easier for modeling the time aspects of systems, although timed automata are not more expressive than transition systems.

To capture the timed properties when modeling web services, we rely on clocks as defined in standard timed automata [9] C is a set of clocks and $\theta(C)$ is the set of conjunctions over simple conditions of the form x⋈c or x-y⋈c, where $x, y \in C$, $c \in \mathbb{N}$ and $\bowtie \in \{<, \leq, =, \neq, >, \geq\}$.

Formally we have the following definition on timed automata.

Definition 2 (Timed Automation) A timed automaton is a 6-tuple (L, l_0, C, A, E, I) ,where L is a set of locations, $l_0 \in L$ is the initial location, C is the set of clocks, A is a set of actions, $E \subseteq L \times A \times \theta(C) \times 2^C \times L$ is a set of edges between locations with an action, a guard and a set of clocks to be reset, and $I : L \to \theta(C)$ assigns invariants to locations.

We now define the semantics of a timed automaton. A clock valuation is a function $u : C \to \mathbb{R}^+$ from the set of clocks to the non-negative reals. Let $\mathbb{R}^C$ be the set of all clock valuations. Let $u_0(x) = 0$ for all $x \in C$. We will abuse the notation by considering guards and invariants as sets of clock valuations, writing $u \in I(l)$ to mean that u satisfies $I(l)$.

Definition 3 (Semantics of TA)

Let $TA = (L, l_0, C, A, E, I)$ be a timed automaton. The semantics is defined as a labeled transition system $(S, s_0, \to)$, where $S \subseteq L \times \mathbb{R}^C$ is the set of states, $s_0 = (l_0, u_0)$ is the initial state, and $\to \subseteq S \times (\mathbb{R}^+ \cup A) \times S$ is the transition relation such that:

$(l,u) \xrightarrow{d} (l, u+d)$ *if* $\forall d' : 0 \le d' \le d \Rightarrow u + d' \in I(l)$, and

$$(l,u) \xrightarrow{a} (l',u') \text{ if there exists } e = (l,a,g,r,l') \in E \text{ s.t. } u \in g, u' = [r \mapsto 0]u, \text{ and } u' \in I(l')$$

where for $d \in \mathbb{R}^+$, $u + d$ maps each clock x in C to the value $u(x) + d$, and $[r \mapsto 0]u$ denotes the clock valuation which maps each clock in r to 0 and agrees with u over $C \setminus r$.

B. Feature Timed Automata

The purpose of FTAs is to model the behaviors of families of Service-oriented System. Think that features are often used for compact representations of a family's products. To model such service family representations as TAs, a "translation" from features to actions is needed. Features and actions are not necessarily a one-to-one mapping, for actions can be enabled by the feature logic formulas:

Let F be a set of features in a FD and the set of feature logic formulas over F denoted $\mu(F)$, is defined as follows:

$$\mu ::= f \mid \neg\mu \mid \mu_1 \wedge \mu_2 \mid \mu_1 \vee \mu_2 ,$$

where $f \in F$ is a feature.

For example, the condition of paying by cell phone in the family of travel agency services is both of *cellPhone* and *phonePay* are selected, *i.e.*, $cellPhone \wedge phonePay$.

Products of a service family are considered to be different with regard to the actions they can perform in any given state of the FTA. This means that the FTA of a product family has to accommodate all the possibilities desired for each derivable product, predicating on the choices of features that make a product belong to that family. Therefore, our approach models a service family that contains all features, as well as annotations that indicate relationships between feature logic formulas and actions.

The behavior of a service family may become complicated for accommodating all the possibilities. This will cause conflicts between actions. For this reason, we define action priorities which offer an intuitive way to model cases in which one transition overrides the behavior of another. FTAs model constraints by explicitly referring to feature formulas and priorities.

Definition 4 (Feature Timed Automata) A featured timed automaton is a tuple $FTA = (TA, FD, \gamma, \succ)$,where $TA = (L, l_0, C, A, E, I)$ be a timed automaton, $FD = (F, r, DE)$ is a feature diagram, $\gamma : A \to \mu(F)$ is a total function, labeling actions with feature logic formulas, $\succ \subseteq E \times E$ is a partial order, defining priorities among actions.

The FTA of travel agency is depicted in Figure 2, with the feature formula label of an action shown next to its action label (separated by the solidus (/)), features are expressed by abbreviation in Figure 1.

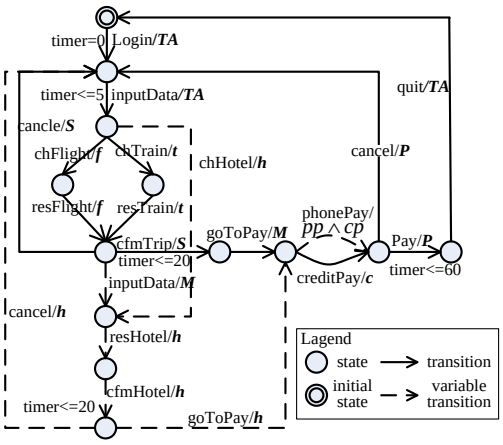

Figure 2. FTA for Family of Travel Agency Services

Definition 5 (semantics of an FTA)

Let $FTA = (TA, FD, \gamma, \succ)$ be a feature timed automaton. $\llbracket d \rrbracket_{FD}$ is the products. The semantics is defined as a labeled transition system $(S, s_0, \rightarrow)$, where $S \subseteq L \times \mathbb{R}^{C}$ is the set of states, $s_0 = (l_0, u_0)$ is the initial state, and $\rightarrow \subseteq S \times (\mathbb{R}^{+} \cup A) \times S$ is the transition relation such that:

$(l,u) \xrightarrow{d} (l,u+d)$ *if* $\forall d' : 0 \le d' \le d \Rightarrow u + d' \in I(l)$, and

$(l,u) \xrightarrow{a} (l',u')$ *if* $\exists e = (l,a,g,r,l') \in E$
$s.t.\ u \in g, u' = [r \mapsto 0]u, u' \in I(l')$ *and* $\gamma(\alpha) \in \llbracket d \rrbracket_{FD}$
$\wedge \neg \exists \alpha' \gamma(\alpha') \in \llbracket d \rrbracket_{FD} \wedge \alpha' \succ \alpha$

Based on FTAs, the behavior of a particular product of the service product family is obtained through projection. This operation removes all actions of the FTA whose feature logic formula is not satisfied by the product. The result of a projection on a FTA over one product is a TA. Formally, we have the following definition.

Definition 6 (projection in FTA)

The projection of an FTA $fta = (L, l_0, C, A, E, I, FD, \gamma, \succ)$ to a product $p \in \llbracket d \rrbracket_{FD}$, noted $fta|_p$, is the TA $ta = (L, l_0, C, A, E', I)$ where

$E' = \{e \in E \mid \gamma(e)(\mu(p)) = 1 \wedge \neg \exists e' \in E \llbracket \gamma(e')(\mu(p)) = 1 \wedge e' \succ e\}$.

Through the projection, one can obtain the variants behavior of a particular product. All the behaviors of product families can be obtained through FTA. For example, the behavior of the product (pid=2 in Table 1) which only provide flight and train reservation through internet is shown in Figure. 3(a); the behavior of the product (pid=9 in Tab. 1) which provide flight, train and hotel reservation through *cellPhone* and both of *phonePay* and *creditPay* is shown in Figure 3(b).

In that way, a FTA is thus the union of the behaviors of all projections. More generally, we have the following definition.

Definition 7 $\llbracket fta \rrbracket_{FTA} = \bigcup_{p \in \llbracket d \rrbracket_{FD}} \llbracket fta|_p \rrbracket_{TA}$

Therefore, the semantics of FTA is the union of the TA semantics for all possible products. The behavior of service families can be formally modeled by FTA, how we can know whether it is satisfied by the specification? The existing temporal logic formula is checked over the

FTA of all products as a whole, which does not aim at a specific product. The product for which a property holds is missing in the logic. We need to propose a temporal logic formula which can also specific product. Next, we will discuss it in detail.

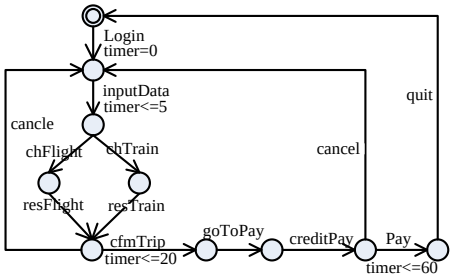

Figure 3(a). TA for the Product Pid=2

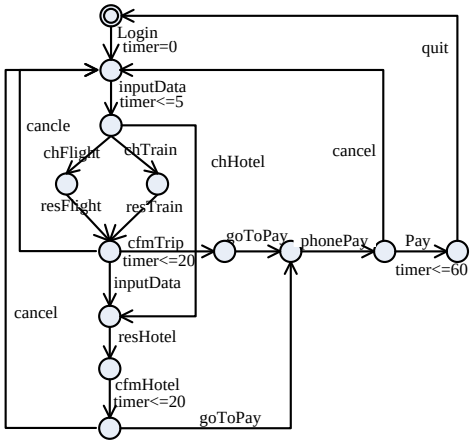

Figure 3(b). TA for the Product Pid=9

Figure 3. TAs Projected from FTA for Certain Products

4. The Temporal Logics for FTA

In this section, we begin our study of the model checking problem for service-oriented product lines. Our objective is to verify temporal properties for all the products of a service family. We first discuss how to encode sets of products in order to define the property for certain products. Then we extend the CTL into feature-based CTL with a semantics that is specifically well suited for capturing the aforementioned variability constraints.

A .Computation Tree Logic (CTL)

Before introducing our temporal logic we recall the syntax and semantic of CTL. CTL does not allow nesting of path formulae and consists of path formulae and state formulae. State formulae describe individual states, whereas path formulae quantify over paths or traces of the model. Path formulae can be classified into reachability, safety and liveness.

Definition 8 (Syntax of CTL) Formulae in CTL are either state or path formulae. TCTL state formulae over the set AP of atomic propositions and the set C of clocks are formed according to the following grammar:

$\beta ::= true \mid a \mid \neg \beta \mid \beta_1 \wedge \beta_2$

$\varphi ::= \forall \square \beta \parallel \exists \Diamond \beta$

where a is an atomic formula being either an atomic clock (or data) constraint or a location (l_i at l). Atomic clock (data) constrains are either inter bounds on individual clock (data) variables (*e.g.*,$1 \le x \le 5$) or integer bounds on differences of two clock (data) variables (*e.g.*, $3 \le x - y \le 7$).

Intuitively, $\forall$ (resp. $\exists$) means at least one path satisfies (resp. all paths satisfies) the following path operator; $\Box$ (resp. $\Diamond$) means the reachable state always (resp. eventually) satisfied.

B. product CTL (pCTL)

A CTL algorithm [10] computes the set of states that do satisfy the property being checked. When checking the behavior model of service families, the CTL formula is only evaluated over the timed computation tree of the FTA, that is, the behavior over all products (not a certain product) in the SOPL. However, model checking on FTA is not only on the whole process, but also on the behavior of certain products. Additional dimension which expresses the products for a property holds is missing in the logic.

Hence, a first challenge is to find an efficient way to represent sets of products. Here, we adopt the encoding method in [11], define a Boolean function χ_{px} to represent the features of product.

Definition 8 (encoding sets of products) A Boolean function $\chi_{px}(f_1,\ldots,f_n)$ over the set of features F, $\chi_{px} : \{0,1\}^{|F|} \to \{0,1\}$, is a symbolic encoding of a set of products $px \in \Box d\Box_{FD}$ such that:

$$\Box\chi_{px}\Box\Box \{p \in \Box d\Box_{FD} \chi_{px}(x_1,\ldots,x_n) = 1\} \; with \; x_i = 1 \Leftrightarrow f_i \in p \; for \; i = [1..n]\}$$

The Boolean function is the characteristic function of the set, return 1 for elements in the set, and otherwise return 0 in order to facilitate implementation of feature sets in UPPAAL.

Based on the encoding of products, we propose product CTL, an extension of CTL that makes products dimension available, which can evaluate not only properties over all products, but also properties over certain products. The syntax of pCTL is given as follows:

Definition 9 (product CTL) A feature CTL formula φ is an expression

$$\beta ::= true \mid \alpha \mid \neg\beta \mid \beta_1 \wedge \beta_2$$

$$\varphi ::= [\chi_{px}]\forall\Box\beta \mid [\chi_{px}]\exists\Diamond\beta \mid [\chi_{px}]\beta_1 \hat{I} \; \beta_2$$

where $\hat{I}$ denote the transitive closure of the delay-transition and action-transition relations between states.

To interpret an pCTL formula on an execution path, we introduce the notion of a execution path. Let $s = (l,u) \in S \subseteq L \times R^C$ be a FTA state and $\pi(s^0) = s^0 \xrightarrow{d_0} s^0 + d_0 \xrightarrow{a_0} s^1 \xrightarrow{d_1} s^1 + d_1 \xrightarrow{a_1} \cdots\cdots$ be an execution path (starting from s^0), $px \subseteq \Box d\Box_{FD}$ be a set of products. The formal semantics of pCTL over a FTA $fta = (TA, FD, \gamma, \succ)$ is given by the satisfaction relation ‘ defined as follows:

$$s, px \; ‘ \; \varphi \Leftrightarrow \forall p \in px \Box s, p \; ‘ \; \varphi,$$

$$s \text{ ‘ } true$$

$$s, p \text{ ‘ } \alpha \quad iff \quad \alpha \in L(l) \times I(l)$$

$$s, p \text{ ‘ } \neg\beta \quad iff \quad \neg(s, p) \text{ ‘ } \beta$$

$$s, p \text{ ‘ } \beta_1 \wedge \beta_2 \quad iff \quad s, p \text{ ‘ } \beta_1 \; and \; s, p \text{ ‘ } \beta_2$$

$$s, p \text{ ‘ } [\chi_{px}]\forall\Box\beta$$

where $iff \quad p \in \Box\chi_{px}\Box, \forall \pi(s) \in \Box fta \mid_p \Box_{FTA}, \forall s' \in \pi(s), s', p \text{ ‘ } \beta$

$$s, p \text{ ‘ } [\chi_{px}]\exists\Diamond\beta$$
$$iff \quad p \in \Box\chi_{px}\Box, \exists \pi(s) \in \Box fta \mid_p \Box_{FTA}, \exists s' \in \pi(s), s', p \text{ ‘ } \beta$$

$$s, p \text{ ‘ } [\chi_{px}]\beta_1 \hat{I} \quad \beta_2$$
$$iff \quad p \in \Box\chi_{px}\Box, \exists \pi(s) \in \Box fta \mid_p \Box_{FTA},$$
$$s, p \text{ ‘ } \beta_1 \Rightarrow \exists s' \in \pi(s), s', p \text{ ‘ } \beta_2$$

χ_{px} is the product constraint in pCTL, which is used to specify the property is evaluated over certain products. It can be ignored when the formula is evaluated over all products.

For example, in Travel Agency service family, all products requires confirming and paying for orders within 60 unit times , which is expressed by the pCTL formula: $Agency.payOK \wedge (timer \leq 60)$;

For products which only provide reservation by Computer, orders only can be paid by credit Card. The condition of features is creditPay should be selected and phonePay not, that is, the logic of product constraint is $creditPay \wedge \neg phonePay$, the property which is expressed by pCTL formula is:

$[creditPay \wedge \neg phonePay]\forall\Box\neg Agency.creditPay$.

In general, an FTA fta satisfies an pCTL formula if the formula is satisfied by all fta's initial states for all products.

In the following section, we will perform the method on checking whether a fta satisfies pCTL formulas in practice.

5. Model Checking a Family of Services

The previous sections provide a mathematical foundation for the model checking of FTAs. To verify our approach and make it available to engineers. We propose an UPPAAL model checker based approach. To do so, we first compare the capability of our approach and UPPAAL model checker that aim to adapt the use of the model checker to deal with the family of timed web services

UPPAAL is a model checker for the verification and simulation of real time systems [7]. An UPPAAL model is a set of timed automata, clocks, channels for systems (automata) synchronization, variables and additional elements. Each automaton has one initial state. The conditions associated to states, called invariants, specify that the system can stay in the state while the invariant is satisfiable. Synchronization between different processes can take place using channels. A synchronization label is either on the form Expression! or Expression? or is an empty label. The expression must be side-effect free, evaluate to a channel, and only refer to integers, constants and channels. We can implement the interaction between different entities through synchronization. Variables and clocks can be associated to processes (automata). Conditions on these clocks and variables can be associated to transitions and states of the process. The conditions associated to transitions, called guards, specify that a transition can be fired only if the corresponding guards are satisfiable. It evaluates to a Boolean value and references to clocks, integer variables, constants and arrays of these types). Priorities can

also be assigned to the channels and processes of a system. The UPPAAL properties query language is a simplified version of Computation Tree Logic.

A.Modeling a travel agency service family

As mentioned above, UPPAAL holds the notion of channels to synchronize real systems. By using such property, we can implement the interaction among several entities. For the behavior of travel agent services reality, following the specification in Section 2, the high level flow of the messages exchanged between travelers and the agency is shown in Figure 4. And therefore we define a User time automata and an Agency featured time automata. We can define several channels like login, confirm, and cancel *etc.* to communication between travelers and the agency.

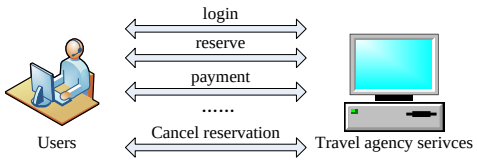

Figure 4. The High Level Flow of the Messages Changed

We propose to build the mapping of features and actions in TAs by means of guards associated to transition. First, a two-dimensional array $px[|px|][|F|]$ is defined in UPPAAL to encode products and corresponding features as definition 4, which realize the feature logic formulas labeling on transitions. Then, a function with a parameter is defined to check whether the parameter (represented a feature logic) equals to 1, which means the feature logic is satisfied to the transition. Priorities on channels and processes in UPPAAL can be used to model the transition priorities defined in FTA.

For this certain service, we only labeled variable features on *Agency*, this will reduce the checking time. The behavior models of *Agency* and User in UPPAAL is shown in Figure 5.

B. Model checking the service family

Next, we will check whether the model defined above is satisfied to all the products. The temporal properties should be defined by pCTL firstly. Here, according to the specification in Section II, *Agency* and *User* are combined as a whole. Based on the definition of pCTL, we formalize properties for our running example.

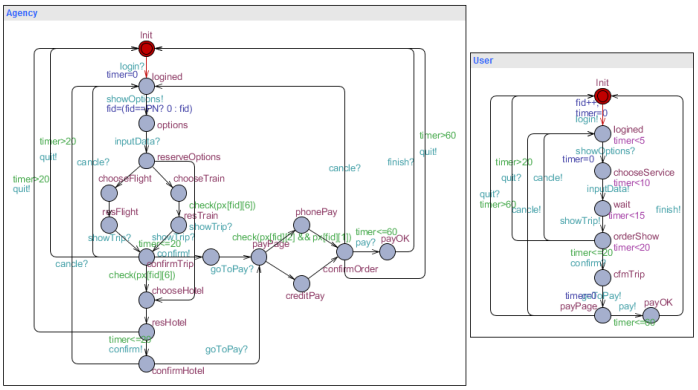

Figure 5. The Behavior Models of Travel Agency Family in UPPAAL

The properties include the following four types [12]:

1) Service Safeness: Safety properties are on the form: "something bad will never happen". When the system is running, it will never reach to error states or bad states, such as systems will not be deadlock forever.

a) It does not allow dead-lock when running systems:

$$\exists\Box\ Agency.Init \wedge User.Init$$

b) Products which only provide reservation by Computer, orders only can be paidby credit Card

$$[creditPay \wedge \neg phonePay]\forall\Box \neg Agency.phonePay$$

c) It does not provide Payment function when timer is more than 60 unit times.

$$\exists\Box(Agency.payOK \wedge timer > 60)$$

2) State reachability: Reachability properties are the simplest form of properties. They ask whether an expected state can be satisfied eventually along the path from an initial state. It can be used to check whether Critical states can be reached.

a) All of travel agency services always provide a selection of reservation.

$$\exists\Diamond Agency.reserveOptions$$

b) Products which can provide hotel reservation always provide such reservation.

$$[hotel]\exists\Diamond Agency.resHotel$$

3) Liveness: Liveness properties are of the form: something will eventually happen.

a) When a user chooses service (flight/train/hotel), then eventually the order should be shown

$$User.chooseService\ \hat{I}\quad User.orderShow$$

4) Time constraints: This kind of property emphasize the execution time of behaviors, it contains a clock variants in the expression.

a) When users login, they must confirm orders within 20 time units.

$$User.logined\ \hat{I}\quad User.cfmTrip \wedge\ timer \leq 20$$

b) Users should pay for orders within 60 unit times.

$$User.cfmTrip\ \hat{I}\quad Agency.payOK\ \wedge\ timer \leq 60$$

We transfer the logics in pCTL into the corresponding symbol in UPPAAL. For product constraints, *pid*(see in Table I is used to express products.

Then we simulate the running of the Travel Agency family through UPPAAL simulator; select the actions to observe the changes of the variables. Traces of each product can be recorded. The changes of the data variables can also be observed. After that, we verified above formulas with UPPAAL verifier and each property is satisfiable, which indicates that the model and its variants are validity.

C. Evaluation

In order to evaluate our approach, in the last section, we report on the analysis of our travel agency service family example. In this section, through comparing the performance of the classical model checking algorithm to our method, we evaluate the model checking performance of ours.

Supposing there are 100 users using different products from the travel agency product family, we should analyze and check behaviors of this product family. All experiments were run on windows 7 with an Intel Core i5 CPU processor and 2GB memory. The time of model checking each property of all products separately with our method (Our.) and the enumerative algorithm (Enum.) is shown in Table 2. These results show that when the more product expressed by a property, the higher verification efficiency achieved, like property 3, 4, 8. When a property expresses only little products, the efficiency is not obvious, or even worse

than the classical method, like property 2, 5. With all properties of this example, our method is on average 5.54 times faster than the enumerative algorithm

Table 2. Comarison of Model Checking Time between SOPL and Classical Method

property	our.	enum.	ratio
1	0.702	3.152	4.49
2	0.437	0.42	0.96
3	0.171	1.87	10.94
4	0.156	1.763	11.30
5	0.156	0.267	1.71
6	0.453	2.509	5.54
7	0.172	1.642	9.55
8	0.172	1.778	10.34
total	2.419	13.401	5.54

6. Related Work

A lot of progresses have been made for behavioral modeling and model checking in SPLs. Fantechi *et al.* [13] propose a variant of Modal Transition Systems to formalize product families firstly. On that basis, Asirelli *et al.* [6] present vaCTL to model checking the behavior models, and define a visit function to acquire the product behavior from the family. Similarly to our approach, these methods allow to verify all possible products at once. Yet, the models lack the notion of feature and priority between transitions. Andreas Classen *et al.* [11] propose featured transition system to combine feature models and behavior models and bring about the corresponding checking algorithm to check all products. Beek *et al.*[14] propose a formal MTS framework to model and analyze variability in product family behavior. The family model is specified in a process-algebraic language with a semantic interpretation as MTS, together with an additional set of variability constraints. And properties to be verified are formalized in v-ACTL. Cordy *et al.* [15] use FTS to formally represent SPL and present abstraction-based model checking for the verification of SPL. However, all these methods are not considering time constraints.

Particularly, in this paper we are interested in the modeling and checking of SOPLs. Then we discuss some approaches used in SOPLs in detail. Mohabbati *et al.* [3] leverage feature modeling and template-based approaches to model variability and configure SOPLs. Templates-based approach enables representing the union of the business processes in all valid template instances. However, they do not study procedures for model checking of templates. Asirelli *et al.* [16] propose a simple action-based branching-time temporal logic over MTS which can manage and verify variability. However, this paper mainly focuses on modeling and verifying the variability of behaviors, ignores the inner relationship between features and behaviors.

7. Conclusion and Future Work

Recognition of the potential synergistic use of SOA and SPLs has motivated the development of several approaches for SOPLs, which provides compact representations of service families.

In this paper, we described an approach for modeling and efficient verification of service-oriented product lines. This step eases the high-level modeling and analysis of web services. Firstly, we propose FTA, feature Timed Automata, a formalism designed to describe the

combined behavior of a timed service family. While allowing modeling very detailed behavioral variations, FTA leverages on establishing relationships between features and transitions, which can reflect variations of features to behavior models directly. Then, we present pCTL to model timed properties of service families, which extend CTL to take into account the products constraints. Thereby, we can use model checking technique verify all the behaviors of a service family at once and detect traces against temporal properties of products. Finally, using our methodology, we have analyzed a travel agency service family using the model checker UPPAAL. As a result, our approach helps to verify timed service families effectively in the early phases of development.

Several issues deserve a further investigation. A major issue is to illustrate the efficiency of our method from quantitative viewpoint through model checking of benchmarks. This is the next step we are going to work. Another crucial issue is study effective model checking algorithms against large service families with many variants and many features.

Acknowledgment

This research is supported by the Foundation for Outstanding Young Scientist in Shandong Province (No. BS2014DX021), the Fundamental Research Funds for the Central Universities (No. 14CX02136A).

References

[1] Y. Wei and M. B. Blake, "Service-Oriented Computing and Cloud Computing: Challenges and Opportunities", IEEE Internet Computing, **(2011)**, pp. 72-75.
[2] N. Gamez, J. E. Haddad and L. Fuentes, "Managing the Variability in the Transactional Services Selection", International Workshop on Variability Modeling of Software-Intensive Systems, **(2015)**, pp. 88.
[3] B. Mohabbati, M. Hatala, D. Gašević, M Asadi and M. Bošković, "Development and configuration of service-oriented systems families", Proceedings of the 2011 ACM Symposium on Applied Computing, **(2011)**, pp. 1606-1613.
[4] T. Nguyen and A. Colman, "Managing service variability: state of the art and open issues," in Proceedings of the 5th Workshop on Variability Modeling of Software-Intensive Systems, **(2011)**, pp. 165-173.
[5] K. Watahiki, F. Ishikawa and K. Hiraishi, "Formal verification of business processes with temporal and resource constraints", Proceedings of the 2011 IEEE International Conference on Systems, Man, and Cybernetics, **(2011)**, pp. 1173-1180.
[6] P. Asirelli, "Design and validation of variability in product lines," in Proceedings of the 2nd International Workshop on Product Line Approaches in Software Engineering, **(2011)**, pp. 25-30.
[7] K. G. Larsen, P. Pettersson and W. Yi, "Uppaal in a nutshell," International Journal on Software Tools for Technology Transfer, **(1997)**.
[8] P. Y.Schobbens, P. Heymans, J. Trigaux and Y. Bontemps, "Generic semantics of feature diagrams", Comput. Netw. vol. 51, no. 2, **(2007)**, pp. 456-479.
[9] R. Alur and D. L. Dill, "A theory of timed automata," Theoretical Computer Science, vol. 126, no. 2, **(1994)**, pp. 183–235.
[10] C. Baier and J. P. Katoen, "Principles of Model Checking," The MIT Press, first edition, **(2008)**.
[11] A. Classen, P. Heymans, P. Schobbens and A. Legay, "Symbolic model checking of software product lines," in Proceeding of the 33rd International Conference on Software Engineering, **(2011)**, pp. 321-330.
[12] L. X. Li, Z. Jin and G. Li, "Modeling and Verifying Services of Internet of Things Based on Timed Automata," Chinese Journal of Computers, vol. 34, no. 8, **(2011)**, pp. 1365-1377.
[13] A. Fantechi and S. Gnesi, "A behavioural model for product families," in Proceedings of the 15th ACM SIGSOFT International Symposium on Foundations of Software Engineering, **(2007)**, pp. 521–524.
[14] M. H. T. Beek,"Modelling and analysing variability in product families: Model checking of modal transition systems with variability constraints." Journal of Logical & Algebraic Methods in Programming, vol. 85, no. 2, **(2015)**, pp. 287-315.
[15] M. Cordy, A. Classen, G. Perrouin, P. Schobbens, P. Heymans and A. Legay, "Simulation-based abstractions for software product-line model checking," in Proceedings of the 34th International Conference on Software Engineering, **(2012)**, pp. 672-682.

[16] P. Asirelli, M. H. T. Beek, A. Fantechi and S. Gnesi, "A model-checking tool for families of services," in Proceedings of the 2011 International Conference on Formal Methods for Open Object-based Distributed Systems, **(2011)**, pp. 44–58.

