Dynamic Synthesis of Local Time Requirement for Service Composition

Tian Huat Tan*, Étienne André†, Jun Sun‡, Yang Liu§, Jin Song Dong¶, Manman Chen¶
*NUS Graduate School for Integrative Sciences and Engineering, Singapore
tianhuat@comp.nus.edu.sg
†Université Paris 13, Sorbonne Paris Cité, LIPN, CNRS, UMR 7030, F-93430, Villetaneuse, France
Etienne.Andre@lipn.univ-paris13.fr
‡Singapore University of Technology and Design, Singapore
sunjun@sutd.edu.sg
§Nanyang Technological University, Singapore
yangliu@ntu.edu.sg
¶National University of Singapore, Singapore
{dongjs, chenman}@comp.nus.edu.sg

***Abstract*—Service composition makes use of existing service-based applications as components to achieve a business goal. In time critical business environments, the response time of a service is crucial, which is also reflected as a clause in service level agreements (SLAs) between service providers and service users. To allow the composite service to fulfill the response time requirement as promised, it is important to find a feasible set of component services, such that their response time could collectively allow the satisfaction of the response time of the composite service. In this work, we propose a fully automated approach to synthesize the response time requirement of component services, in the form of a constraint on the local response times, that guarantees the global response time requirement. Our approach is based on parameter synthesis techniques for real-time systems. It has been implemented and evaluated with real-world case studies.**

I. Introduction

Service-oriented architecture is a paradigm that promotes the building of software applications by using services as basic components. Services make their functionalities available through a set of operations accessible over a network infrastructure. To assemble a set of services to achieve a business goal, service composition such as BPEL (Business Process Execution Language) Orchestration [1] has been proposed. The service that is composed by service composition is called a *composite* service, and services that the composite service makes use of are called *component* services.

In business where timing is critical, a requirement on the service response time is often an important clause in service-level agreements (SLAs), which is the contractual basis between service consumers and service providers on the expected quality of service (QoS) level. Henceforth, we denote the response time requirement of composite services as *global time requirement*; and the set of constraints on the response times of the component services as the *local time requirement*. The response time of a composite service is highly dependent on that of individual component services. It is therefore important to derive the local time requirement from the global time requirement so as to identify component services which could be used to build the composite service while satisfying the response time requirement. Consider an example of a stock indices service, which has an SLA with the subscribed users, such that the stock indices would be returned in two seconds upon request. The stock indices service makes use of three component services, including a paid service, to request for the stock indices. The stock indices service provider would be interested to know the local time requirement of the component services.

BPEL is a service composition language that supports complex timing constructs and control flow structures such as concurrency. Such a combination of timing constructs, concurrent calls to external services, and complex control structures makes it a challenge to synthesize the local time requirement.

In this paper, we present a fully automated technique for the synthesis of the local time requirement in BPEL. The approach works by performing dynamic analysis on the service composition, using techniques of parameter synthesis for real-time systems. For the synthesized local time requirement to be useful, it needs to be as weak as possible, to avoid discarding any service candidates that might be part of a feasible composition. This is particularly important, as often having a faster service would incur higher cost. To synthesize a better constraint that allows larger sets of feasible composition, we provide an extra analysis on the activities, and classify them as or-activities or and-activities. We then extend labeled transition systems (LTS) with and-states and or-states, which we call and/or LTS (AOLTS), for synthesizing the local time requirement. The approach not only avoids bad scenarios in the service composition, but also guarantees the fulfillment of global time requirement.

Our contributions are as follows:

1) Given a composite service, we develop a sound method for synthesizing the local time requirement in the form of a set of constraints on the local service response

978-1-4673-3076-3/13/$31.00 © 2013 IEEE

ICSE 2013, San Francisco, CA, USA

times. The approach is implementation independent, therefore can be applied at the design stage of service composition.

2) We propose a new kind of labeled transition systems called AOLTS, that allows us to exploit the hierarchical structure of composite services for synthesizing a more relaxed local time requirement, which is yet strong enough to guarantee the reachability of good scenarios in *any* instances of the composite services.
3) We develop a fully automated tool to evaluate the proposed method and apply it to real-world case studies.

The synthesized local time requirement has multiple advantages. First, it allows the selection of feasible services from a large pool of services with similar functionalities but different local response times. Second, the designer can avoid over approximations on the local response times. An over approximation may lead the service provider to purchase a service at a higher cost, while a service at a lower cost with a slower response time may be sufficient to guarantee the global time requirement. Third, the local requirements serve as a safe guideline when component services are to be replaced or new services are to be introduced. Last but not least, the requirement synthesized by our method gives a quantitative measure of the *robustness* of the composite services. Indeed, if the global time requirement is satisfied given a local response time of 2 seconds, it may not be the case anymore for a value bigger than, but very close to, 2 seconds (e.g., 2.001 seconds). The constraint synthesized gives a measure of the upper bound until which each local response time can vary.

Outline. The rest of this paper is structured as follows. Section II introduces a timed BPEL running example. Section III provides the necessary definition for terminologies. Section IV introduces our approach to dynamically analyze the BPEL process. Section V presents the synthesis algorithms and their soundness proofs. Section VI reviews related works. Section VII discusses the application of our approach on two case studies. Finally, Section VIII concludes the paper, and outlines future work.

II. A Timed BPEL Example

BPEL is a standard for implementing composition of existing Web services by specifying an executable work flow using predefined activities. In this work, we assume the composite service is specified using the BPEL language. Basic BPEL activities that communicate with component Web services are `<receive>`, `<invoke>`, and `<reply>`, which are used to receive messages, execute component Web services and return values respectively. We denote them as communication activities. The control flow of the service is defined using structural activities such as `<flow>`, `<sequence>`, `<pick>`, `<if>`, etc. In this section, we illustrate a *Stock Market Indices Service* (SMIS) that is used as a running example in this work.

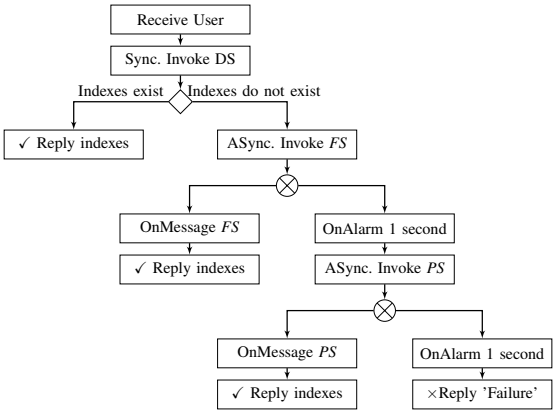

Fig. 1. Stock Market Indices Service

A. Stock Market Indices Service

Assume that the SMIS is a paid service and its goal is to provide updated stock indexes to the subscribed users. It provides service level agreement (SLA) to the subscribed users stating that it always responds within two seconds upon request.

The SMIS has three component Web services: a database service (*DS*), a free news feed service (*FS*) and a paid news feed service (*PS*). The strategy of the SMIS is calling the free service *FS* before calling the paid service *PS* in order to minimize the cost. Upon returning the result to the user, the SMIS would also store the latest results in an external database service provided by *DS* (storage of the results is omitted here). The workflow of the SMIS is sketched in Fig. 1 under the form of a tree. When a request is received from a subscribed customer (`Receive User`), it would synchronously invoke (i.e., invoke and wait for reply) the database service (`Sync. Invoke DS`) to request for stock indexes stored in the last minute. Upon receiving the response from *DS*, the process is followed by an `<if>` branch (denoted by $\diamondsuit$). If the indexes are available (`indexes exists`), then they are returned to the user (`Reply indexes`). Otherwise, *FS* is invoked asynchronously (i.e., the system moves on after the invocation without waiting for the reply). A `<pick>` construct (denoted by $\otimes$) is used here to await incoming response (`<onMessage>`) from previous asynchronous invocation and timeout (`<onAlarm>`) if necessary. If the response from *FS* (`OnMessage` *FS*) is received within one second, then the result is returned to the user (`Reply indexes`). Otherwise, the timeout occurs (`OnAlarm 1 second`), and SMIS stops waiting for the result from *FS* and calls *PS* instead (`ASync. Invoke` *PS*). Similar to *FS*, the result from *PS* is returned to user, if the response from *PS* is received within one second. Otherwise, it would notify the user regarding the failure of getting stock indexes (`Reply 'Failure'`). The state with a ✓ (resp. ×) represents the desired (resp. undesired) end state.

The global time requirement for SMIS is that SMIS should respond within two seconds upon request. It is of particular interest to know the local time requirements for services *PS*,

FS, and *DS*, so as to fulfill the global time requirement. This information could also help to choose a paid service *PS* which is both cheap and responding quickly enough.

B. BPEL Notations

To present BPEL syntax succinctly, we define a set of BPEL notations below:

- $rec(S)$ and $reply(S)$ are used to denote "receive from" and "reply to" a service S;
- $sInv(S)$ (resp. $aInv(S)$) denotes synchronous (resp. asynchronous) invocation of a service S;
- $P \;|||\; Q$ denotes the concurrent execution of BPEL activities P and Q;
- $P \lhd b \rhd Q$ denotes conditional activity, where b is a guard condition. If b is evaluated as true, BPEL activity P is executed, otherwise activity Q is executed.
- $pick(S \Rightarrow P, alrm(a) \Rightarrow Q)$ denotes BPEL `<pick>` activity, which either receives the message from service S within a seconds, where $a \in \mathbb{R}_{\geq 0}$, and subsequently executes BPEL activity P; or timeouts at a seconds, and subsequently executes BPEL activity Q. If the message is arrived at exactly a seconds, then P or Q executes non-deterministically.

III. Formal Model for Parametric Analysis

A composite service S makes use of a finite number of component services to accomplish a task. Let $C = \{s_1, \ldots, s_n\}$ be the set of all component services that are used by S. The response time of a service s_1 is reflected on the time that spends on the communication activities. For example, assume that the only communication activity that communicates with component service s_1 is $sInv(s_1)$. Upon invoking of service s_1, the construct $sInv(s_1)$ waits for the reply. The response time of component service s_1, is equivalent to the waiting time in $sInv(s_1)$, Therefore by analyzing the time spent in $sInv(s_1)$, we can get the response time of component service s_1. Given a composite service S, let $t_i \in \mathbb{R}_{\geq 0}$ be the response time of component service s_i for $i \in \{1, \ldots, n\}$, and let $C_t = \{t_1, \ldots, t_n\}$ be a set of component service response times that fulfill the global time requirement of service S. Because t_i, for $i \in \{1, \ldots, n\}$, is a real number, there are infinitely many possible values, even in a bounded interval (and even if one restricts to rational numbers). A method to tackle this problem is to reason *parametrically*, by considering these response times as unknown constants, or *parameters*. Let $u_i \in \mathbb{R}_{\geq 0}$ be the parametric response time of component service s_i for $i \in \{1, \ldots, n\}$, and let $C_u = \{u_1, \ldots, u_n\}$ be the set of component service parametric response times. Using constraints on C_u, we can represent an infinite number of possible response times symbolically. The local time requirement of composite service S is specified as a constraint over C_u. An example of local time requirement is $(u_1 < 6) \vee (u_2 < 5)$. This local time requirement specifies that, in order for S to satisfy the global time requirement, either service s_1 needs to respond within 6 time units, or service s_2 needs to respond within 5 time units.

We review relevant definitions in the following.

A. Clocks, Parameters, and Constraints

A *clock* is a variable of type $\mathbb{R}_{\geq 0}$. A clock is used to record the time passage of a communication activity. All clocks are progressing at the same rate. $\mathcal{X}$ is defined as a universal set of clocks. Let $X = \{x_1, \ldots, x_H\} \subset \mathcal{X}$ be a finite set of clocks. A *clock valuation* is a function $w : X \to \mathbb{R}_{\geq 0}$, that assigns a non-negative real value to each clock. A *parameter* is an unknown constant. Let $\mathcal{U}$ denote the universal set of parameters, disjoint with $\mathcal{X}$. Given a finite set $U = \{u_1, \ldots, u_M\} \subset \mathcal{U}$, a *parameter valuation* is a function $\pi : U \to \mathbb{R}_{\geq 0}$ assigning a non-negative real value to each parameter. We can identify a valuation π with the point $(\pi(u_1), \ldots, \pi(u_M))$. A *linear term* on $X \cup U$ is an expression of the form $\sum_{1 \leq i \leq N} \alpha_i z_i + d$ with $z_i \in X \cup U$, $\alpha_i \in \mathbb{R}_{\geq 0}$ for $1 \leq i \leq N$, and $d \in \mathbb{R}_{\geq 0}$. Given $X \subset \mathcal{X}$ and $U \subset \mathcal{U}$, an *inequality* over X and U is $e \prec e'$ with $\prec \in \{<, \leq\}$, where e and e' are linear terms on $X \cup U$. A *constraint* is a conjunction of inequalities. We denote by $\mathcal{C}_{X \cup U}$ the set of all constraints over X and U. Henceforth, we use w (resp. π) to denote a clock (resp. parameter) valuation.

Let $C \in \mathcal{C}_{X \cup U}$, $C[\pi]$ denote the constraint over X obtained by replacing in C each $u \in U$ with $\pi(u)$. Similarly, $C[\pi][w]$ denotes the constraint obtained by replacing each clock x in $C[\pi]$ with $w(x)$. We write $(w, \pi) \models C$, if $C[\pi][w]$ evaluates to true. C is *empty*, if there does not exist a parameter valuation π, such that $\pi \models C$; otherwise C is *non-empty*. We define $C^{\uparrow} = \{x + d \mid x \in C \wedge d \in \mathbb{R}_{\geq 0}\}$, as *time elapsing* of C, i.e., the constraint over X and U obtained from C by delaying an arbitrary amount of time. Given two constraints $C_1, C_2 \in \mathcal{C}_{X \cup U}$, C_1 is *included* in C_2, denoted by $C_1 \subseteq C_2$, if $\forall w, \pi : (w, \pi) \models C_1 \Rightarrow (w, \pi) \models C_2$.

B. Labeled Transition Systems

In this work, the semantics of composite service is captured using labeled transition systems (LTSs). The behavior of composite service is affected by the input data. Since the input data is unpredictable, in order to reason on the general behavior of composite service, it is useful to obtain a local time requirement which could guarantee global time requirement for any input data. For example, given conditional expression $P \lhd b \rhd Q$, the execution of activity P or activity Q is based on the valuation of b; but if we choose to abstract from data, either P or Q will be executed non-deterministically. A state of the LTS represents a status of the composite service. Informally, a concrete state is a state that contains data information, and the LTS that contains concrete state is denoted as concrete LTS. An abstract state is a state which abstracts away data information, and the LTS that contains abstract state is denoted as abstract LTS. Our dynamic analysis is based on the abstract LTS. In the following, we provide the formal definition of various terminologies that are used in this work.

Definition 1 (Parameterized Composite Service Model): A *composite service model* $\mathcal{M}$ is a tuple (Var, V_0, U, P_0, C_0), where *Var* is a finite set of variables, V_0 is an initial valuation that maps each variable to its initial value, U is a finite set

of parameters, P_0 is the composite service process, and C_0 is the initial constraint.

Given a service model $\mathcal{M}$ with a parameter set $U = \{u_1, \ldots, u_m\}$, and given a parameter valuation $(\pi(u_1), \ldots, \pi(u_M))$, $\mathcal{M}[\pi]$ denotes the *instantiation* of $\mathcal{M}$ with π, viz., the model (Var, V_0, U, P_0, C), where C is $C_0 \wedge \bigwedge_{i=1}^{M}(u_i = \pi(u_i))$. Note that $\mathcal{M}[\pi]$ is a non-parametric service model.

Definition 2 (Concrete Labeled Transition System): A *concrete labeled transition system* is a tuple $\mathcal{L}_c = (S_c, s_0^c, \Sigma, \delta_c)$, where

- S_c is a set of concrete states,
- $s_0^c \in S_c$ is the initial state,
- Σ is the universal set of actions,
- $\delta_c : S_c \times \Sigma \times S_c$ is a transition relation.

A *concrete state* s_c is represented as (V, P, C, D), where V is a valuation of the variables (i.e., a function that maps a variable name to its value), P is a composite service process, C is a constraint over $\mathcal{C}_X$, and D is the (real-valued) duration from the initial state s_0 to the beginning of the state s.

Definition 3 (Abstract Labeled Transition System): An *abstract labeled transition system* is a tuple $\mathcal{L} = (S, s_0, \Sigma, \delta)$, where

- S is a set of abstract states,
- $s_0 \in S$ is the initial state,
- Σ is the universal set of actions,
- $\delta : S \times \Sigma \times S$ is a transition relation.

An *abstract state* s is represented as (P, C, D), where P is a composite service process, C is a constraint over $\mathcal{C}_{X \cup U}$, and D is the (parametric) duration from the initial state s_0 to the beginning of the state s (i.e., a linear term on the parameters).

Henceforth, we refer to an abstract state (resp. abstract LTS) as a state (resp. LTS), as long as it is clear from the context.

The following definitions could be easily extended to concrete states and concrete LTSs. Given an LTS $\mathcal{L} = (S, s_0, \Sigma, \delta)$, a state $s \in S$ is said to be a *terminal state* if there does not exist a state $s' \in S$ and an action $a \in \Sigma$ such that $(s, a, s') \in \delta$; otherwise, s is said to be a *non-terminal state*. There is a *run* from a state s to state s', where $s, s' \in S$, if there exist a set of states $\{s_1, \ldots, s_n\} \in S$ and a sequence of actions $\langle a_1, \ldots, a_n \rangle$, where $a_i \in \Sigma$, such that $s_1 = s$, $s_n = s'$, and $\forall\, i \in \{1, \ldots, n-1\}, (s_i, a_i, s_{i+1}) \in \delta$. A *complete run* is a run that starts in the initial state s_0 and ends in a terminal state. Given a state $s \in S$, we use $Enable(s)$ to denote the set of states reachable from s; formally, $Enable(s) = \{s' \mid s' \in S \wedge a \in \Sigma \wedge (s, a, s') \in \delta\}$. Given a state $s = (P, C, D)$, we use the notation $s.P$ to denote the component P of s, and similarly for $s.C$ and $s.D$.

Given $\mathcal{M} = (Var, U, P_0, C_0)$, the *global time requirement* for $\mathcal{M}$ requires that, for every state (P, C, D) reachable from the initial state $(P_0, C_0, 0)$ in the LTS, the constraint $D \leq T_G$ is satisfied, where $T_G \in \mathbb{R}_{\geq 0}$ is the *global time constraint*. The *local time requirement* requires that if the response times of all component services of $\mathcal{M}$ satisfy the *local time constraint* $C_L \in \mathcal{C}_U$, then the service S satisfies the global time requirement.

$Act(A(S), x)$	$= A(S)_x$	A1
$Act(mpick, x)$	$= mpick_x$	A2
$Act(A(S)_{x'}, x)$	$= A(S)_{x'}$	A3
$Act(mpick_{x'}, x)$	$= mpick_{x'}$	A4
$Act(P \oplus Q, x)$	$= Act(P, x) \oplus Act(Q, x)$	A5
$Act(P \,{}_9^{\circ}\, Q, x)$	$= Act(P, x) \,{}_9^{\circ}\, Q$	A6

Fig. 2. Activation Function, where $A \in \{rec, sInv, aInv, reply\}$, $\oplus \in \{|||, \lhd b \rhd\}$, and $mpick = pick(S \Rightarrow P_t, alrm(a) \Rightarrow P_a)$

IV. Dynamic Analysis with LTS

Our approach for synthesizing local time constraint for component services is based on the dynamic analysis of the constraint of each state in the LTS of the composite service $\mathcal{M}$. In this section, we present how we analyze the LTS with real-time semantics using parametric techniques. In order to analyze the LTS with real-time semantics, we use *clocks* to record the elapsing of time. The clock formalism has been used to record the time elapsing in several formalisms, in particular in Timed Automata (TA) [2]. In TAs, the clocks are defined during the modeling phase, and once created, are never discarded. It is known the the state space of the system could grow exponentially with the number of clocks. An alternative approach is to create clocks on the fly when necessary, and have them pruned when no longer needed. This allows smaller state space compared to the explicit clock approach; we refer to this second approach as *implicit clock approach*. We use the implicit clock approach [3], [4] when analyzing the BPEL model with LTSs.

A. Clock Activation

Clocks are implicitly associated with timed processes. For instance, given an action $sInv(s)$, a clock starts ticking once the process becomes activated. To introduce clocks on the fly, we define an activation function *Act* in Fig. 2. Given a process P, we denote by P_x the corresponding process that has been associated with clock x. The activation function will be called when a new state s is reached to assign a new clock for each newly activated communication activity. Rules A1 and A2 state that a new clock is associated with BPEL communication activity if it is newly activated. Rules A3 and A4 state that if a BPEL communication activity has already been assigned a clock, it will be never reassigned. Rules A5 and A6 state that function *Act* is applied recursively for activated child activities for BPEL structural activities.

B. Idling Function

Given a state s, we define function *idle* to calculate how long the activity A can idle in state s. The result is a constraint over the clocks and the parameters. Rule I1 states the situation that the communication required to wait for the response of component services S, and states the value of clock x must not be larger than the response time parameter t_S of the service. Rule I2 states that the situation that no waiting is required. Rules I3 to I5 state that the function *idle* is applied recursively for activated child activities of a BPEL structural activity.

$$
\begin{array}{lll}
idle(A(S)_x) & = x \leq t_S & \text{I1} \\
idle(B(S)_x) & = (x = 0) & \text{I2} \\
idle(P \oplus Q) & = idle(P) \wedge idle(Q) & \text{I3} \\
idle(P \,_{\S}Q) & = idle(P) & \text{I4} \\
idle(mpick_x) & = x \leq t_S \wedge x \leq a & \text{I5}
\end{array}
$$

Fig. 3. Idling Function, where $A \in \{rec, sInv\}$, $B \in \{aInv, reply\}$, $\oplus \in \{|||, \lhd b \rhd\}$, $mpick = pick(S \Rightarrow P_t, alrm(a) \Rightarrow P_a)$, and t_S is the parametric response time of service S

We define an undesired scenario that the user does not want to end in. In case of the SMIS example, it corresponds to a situation where both component services *FS* and *PS* fail to response within one second. Given a BPEL service $\mathcal{M}$, in order to be able to classify the desired and undesired scenarios, we allow the user to annotate a BPEL activity A as a bad activity, denoted by $[A]_{bad}$. The annotation can be achieved, for example, by using extension attribute of BPEL activities. The execution of activity $[A]_{bad}$ will result the LTS of $\mathcal{M}$ to end in an undesired terminal state, which we denote as a *bad state*. The terminal state which is not a bad state is called a *good state*. The synthesized local time constraint needs to guarantee the avoidance of all bad states and the reachability of at least one good state.

C. State Space Exploration

Let $Y = \langle x_0, x_1, \cdots \rangle$ be a sequence of clocks. Starting from the initial state $s_0 = (P_0, C_0, 0)$, we iteratively construct successor states as follows. Given a state (P, C, D), a clock x which is not currently associated with P is picked from Y. The state (P, C, D) is transformed into $(Act(P, x), C \wedge x = 0, D)$, i.e., timed processes which just become activated are associated with x and C is conjuncted with $x = 0$. Then, a firing rule is applied to get a target state (P', C', D'), such that C' be satisfiable (otherwise, the transition is infeasible). Note that, for the sake of conciseness, firing rules are not introduced here; they encode the semantics of BPEL, and are straightforward.

Lastly, clocks which do not appear within P' are pruned from C'. Observe that one clock is introduced and zero or more clocks may be pruned during a transition. Actually, a clock is introduced only if necessary; if the activation function does not activate any subprocess, this new clock is not created.

D. Application to an Example

Consider composite service $\mathcal{M}$, which is a fraction of the SMIS of Section II. The LTS of $\mathcal{M}$ is shown in Fig. 4.

- At state s_0, activation function assigns clock x to record time elapsing of pick activity *mpick* and the tuple becomes $(mpick_x, true, 0)$.
- From state s_0, it could evolve into state s_1 if the constraint $c_1 = (x = t_{PS} \wedge idle(mpick_x))$ where $idle(mpick_x) = (x \leq t_{PS} \wedge x \leq 1)$, is satisfiable. Intuitively, c_1 denotes the constraint where t_{PS} time units elapsed since clock x has started. In fact, constraint c_1 is satisfiable (for example with $t_{PS} = 0.5$ and $x = 0.5$). Therefore, it could evolve into state $s_1 = (r_{good}, x = t_{PS} \wedge x \leq 1, t_{PS})$. Since clock x is not used anymore in $s_1.P$ which is r_{good}, it is eliminated using variable elimination techniques such as Fourier-Motzkin [5]. After elimination of clock variable x and simplification of the expression, state s_1 becomes $(r_{good}, t_{PS} \leq 1, t_{PS})$.
- From state s_0, it could also evolve into state s_2, if the constraint $c_2 = (x = 1 \wedge idle(mpick_x))$, where $idle(mpick_x) = (x \leq t_{PS} \wedge x \leq 1)$, is satisfiable. It is easy to see that c_2 is satisfiable; we now have state $s_2 = (r_{bad}, t_{PS} \geq 1, 1)$.
- From state s_1, activation function assigns clock x' for reply activity r_{good}, and the reply activity becomes $(r_{good})_{x'}$. It could evolve to state s_3, if the constraint $c_3 = (t_{PS} \leq 1 \wedge idle((r_{good})_{x'}))$, where $idle((r_{good})_{x'}) = (x' = 0)$, is satisfiable. In fact it is, with the similar reason as previous cases, and it evolves into terminal state $s_3 = (stop, t_{PS} \leq 1, t_{PS})$, where *stop* is an activity that does nothing. Since the terminal state is due to a good activity, s_3 is considered as a good state, denoted by ✓.
- From state s_2, it could also evolve into terminal state $s_4 = (stop, t_{PS} \geq 1, 1)$. Since the terminal state is due to the bad activity, it is considered as a bad state, denoted by $\times$.

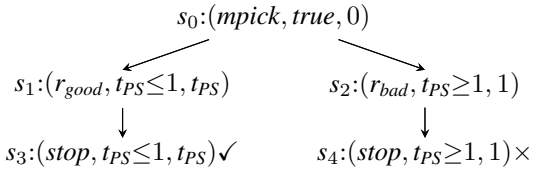

Fig. 4. LTS of service $\mathcal{M}$, where $mpick = pick(PS \Rightarrow r_{good}, alrm(1) \Rightarrow r_{bad})$, $r_{good} = reply(User)$, $r_{bad} = [reply(User)]_{bad}$, and t_{PS} is the parametric response time of service *PS*

V. Constraint Synthesis using AOLTS

In this section, given a global time constraint T_G for service S, we present an approach to synthesize local time constraint C_L based on the LTS. We show that if response times of all component services of S satisfy the local time requirement, the service S would end at a good state regardless of the values input by the user.

A. And-Activity and Or-Activity

To synthesize a better constraint that allows larger sets of feasible composition, we first characterize the BPEL structural activities. We divide the structural activities into and-activities and or-activities. An and-activity is non-deterministic in the abstract LTS but deterministic in the concrete LTS; an example is the `<if>` activity.

Consider the abstract LTS of a service $S = (Var, V_0, U, P_0, C_0)$ that has $Var = \{a\}$, $V_0 = \{a \mapsto 1\}$, $U = \{t\}$ and $P_0 = A \lhd a = 1 \rhd B$ where A and B are BPEL processes and $C_0 = (true)$. The global time requirement for service S requires that the service must complete within four

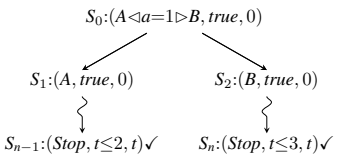

Fig. 5. Abstract LTS

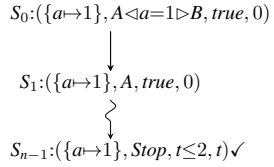

Fig. 6. Concrete LTS

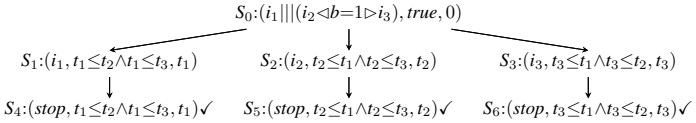

Fig. 7. LTS of composite service S

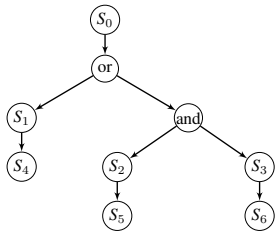

Fig. 8. AOLTS of service S

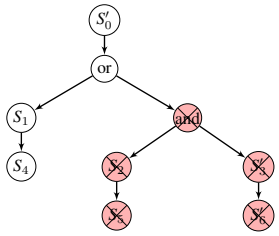

Fig. 9. Bad state propagation

seconds. The abstract LTS of service S is shown in Fig. 5. The initial state s_0 has two branches due to the non-determinism of the abstract LTS for the conditional activity. Assume s_{n-1} and s_n are the (only) two good states that could be reached from execution of s_1 and s_2 respectively. Given constraint $c_1 = (s_{n-1}.C \wedge s_{n-1}.D \leq 4) = t \leq 2$ and constraint $c_2 = (s_n.C \wedge s_n.D \leq 4) = t \leq 3$, the result $c_1 \vee c_2$ (viz., $t \leq 3$) will guarantee the reachability of a good state within four seconds in the abstract LTS. Nevertheless, in the concrete LTS as shown in Fig. 6, the result would be incorrect, since for example, $t = 3$ would not satisfy the constraint of the (only) good state s_{n-1} in the concrete LTS. The reason is that the transition caused by the and-activity in the abstract LTS could be missing in the concrete LTS; therefore we must conjunct the constraint to guarantee it could reach state s_{n-1} in the concrete LTS. In our example, the correct local time requirement would be $c_1 \wedge c_2$, which is $t \leq 2$. If a structural activity is not an and-activity, it is classified as an or-activity. The transition caused by an or-activity in the abstract LTS could not be missing in the concrete LTS. Examples of or-activities include `<flow>`, `<pick>`, and `<sequence>`.

B. And/Or LTS

Since a BPEL composite service contains a hierarchy of and-activities and or-activities, we extend the LTS with And/Or states to facilitate the conjunction and disjunction of constraints in the hierarchical activities. We denote the LTS that has been extended with And/Or state as *And/Or labeled transition system (AOLTS)*.

Definition 4 (And/Or labeled transition system): An *And/Or labeled transition system (AOLTS)* is a tuple $\mathcal{L}_{AO} = (S_{AO}, s_0, \Sigma, \delta_{AO})$ where $S_{AO} = S \cup \{and, or\}$, S is a set of abstract states, $s_0 \in S$ is the *initial state*, Σ_τ is a *set of symbols*, and $\delta_{AO} : S_{AO} \times \Sigma_\tau \times S_{AO}$ is a *labeled transition relation*.

We call a state $s \in S$ a normal state, to differentiate from and-states and or-states. A non-terminal normal state is said to be an *inter-state*. Fig. 7 shows the LTS of a composite service S that contains an and-activity $a_1 = (i_2 \lhd b = 1 \rhd [i_3]_{bad})$ inside an or-activity $a_2 = (i_1 \ ||| \ a_1)$, where i_j denotes $sInv(s_j)$, such that s_j is a component service with parametric response time t_j, for $j \in \{1, 2, 3\}$.

The AOLTS of composite service S is shown in Fig. 8. Given the global time constraint $T_G = 3$, let $c_i = s_i.C \wedge s_i.D \leq 3$, for $i \in \{4, 5, 6\}$, the local time requirement for composite service s is $c_4 \vee (c_5 \wedge c_6)$.

C. Bad State Propagation

Consider now we change the definition of i_3 in Fig. 7 as $[sInv(s_3)]_{bad}$, and we have state s'_0, s'_3 and s'_6 where state s'_6 is a bad state. Let $c_i = s_i.C \wedge s_i.D \leq T_G$, for $i \in \{4, 5\}$, a plausible synthesis solution on the local time requirement for composite service S would be $(c_4 \vee c_5) \wedge \neg s'_6.C$. Given $\pi = \{t_1 \mapsto 2, t_2 \mapsto 1, t_3 \mapsto 3\}$, and variable $b = 2$, this might be resulting in a situation where no good state is reachable. In order to avoid this situation, we apply *bad state propagation*. It works by propagating the bad status to the parent, as described below.

- If the current state is a terminal bad state, propagate the bad status to the parent;
- If the current state is an or-state s_{or}, and all states in $Enable(s_{or})$ have a bad status, then marks s_{or} with a bad status, and propagate the bad status to the parent state, otherwise do nothing;
- If the current state is an and-state s_{and}, and if there exists a state in $Enable(s_{and})$ that has a bad status, then mark s_{and} with a bad status, and propagate the bad status to the parent state, otherwise do nothing;
- If the current state is an inter-state, and if the unique successor state $Succ(s)$ has a bad status, then mark s with a bad status, and propagate the bad status to the parent state, otherwise do nothing.

Consider the example in Fig. 9. The bad state status of s'_6 would propagate up to the and-state. Therefore, the current constraint is $c_4 \wedge \neg (s_2.C \vee s'_3.C)$.

D. The Algorithm for Synthesis of Local Time Requirement

Algorithm 1 presents the main algorithm for synthesizing the local time constraint. Given a state s, $synConsAOLTS(s)$ returns a constraint tuple c_s which is a tuple (b, c) where $b \in \mathbb{B}$ and $c \in \mathcal{C}_U$. Given constraint tuple c_s, we use $c_s.First$ to refer the first component of c_s, and $c_s.Second$ to refer the second component of c_s. In the following, we define notions that are used in Algorithm 1. Given $A = \{c_1, \ldots, c_n\} \subset \mathcal{C}_U$, we denote

Algorithm 1: Algorithm *synConsAOLTS*(s)

input : State s of LTS
output: The constraint tuple for LTS that starts at s

1 **if** *s is good state* **then**
2 **return** *(true, $s.C \wedge (s.D \leq T_G)$)*;
3 **if** *s is bad state* **then**
4 $K_{bad} = K_{bad} \wedge \neg (s.C)$;
5 **return** $(false, \varnothing)$;
6 $CS \leftarrow \{synConsAOLTS(s') \mid s' \in Enable(s)\}$;
7 **if** *s is or-state* **then**
8 **if** $true \in D(\!|CS|\!)$ **then**
9 **return** $(true, \bigvee\{c.Second \mid c \in CS \wedge c.First = true\})$;
10 **else**
11 **return** $(false, \varnothing)$;
12 **if** *s is and-state* **then**
13 **if** $false \in D(\!|CS|\!)$ **then**
14 **return** $(false, \varnothing)$;
15 **else**
16 **return** $(true, \bigwedge\{c \mid c \in R(\!|CS|\!)\})$;
17 **if** *s is inter-state* **then**
18 **if** $CS[0].First = true$ **then**
19 **return** $(true, CS[0].Second)$;
20 **else**
21 $K_{bad} = K_{bad} \wedge \neg (s.C)$;
22 **return** $(false, \varnothing)$;

Algorithm 2: Algorithm *LocalTimeConstraint*(s_0)

input : Initial state s_0
output: The local time constraint C_L

1 $S \leftarrow synConsAOLTS(s_0)$;
2 **if** *S.First=true* **then**
3 **return** $K_{bad} \wedge S.Second$;
4 **else**
5 **return** *false*;

by $\bigwedge A$ the conjunction of constraints in A, i.e., $c_1 \wedge \ldots \wedge c_n$. Similarly, we denote by $\bigvee A$ the disjunction of constraints in A, i.e., $c_1 \vee \ldots \vee c_n$. $D(\!|CS|\!)$ denotes the set of elements in the domain of CS; formally, $\{b : \mathbb{B} \mid \exists\, c \in \mathcal{C}_U, (b,c) \in CS\}$. $R(\!|CS|\!)$ denotes the set of element in the range of CS; formally, $\{c : \mathcal{C}_U \mid \exists\, b \in \mathbb{B}, (b,c) \in CS\}$. If a set CS is known to contain exactly an element cs_1, we use $CS[0]$ to denote that particular element cs_1.

Furthermore, K_{bad} is a static variable of type $\mathcal{C}_U$, that is used to collect the negation of the constraint associated to states that are marked with a bad status (lines 4 and 21). Finally, T_G is the given global time constraint of a service S.

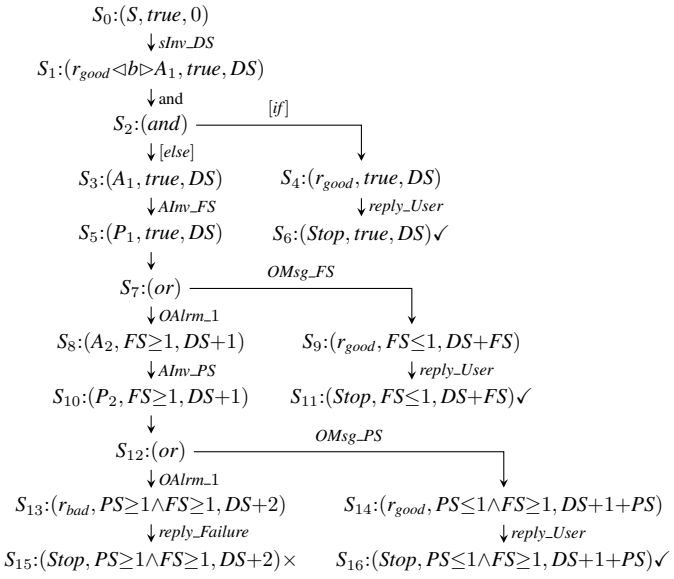

Fig. 10. LTS of the SMIS, where $S{=}(sInv(DS)\,_{9}^{\wp}\, r_{good} \lhd b \rhd A_1)$, $A_1{=}(aInv(FS)\,_{9}^{\wp}\, P_1)$, $P_1{=}(pick(FS{\Rightarrow}r_{good}, alrm(1){\Rightarrow}A_2))$, $A_2{=}(aInv(PS)\,_{9}^{\wp}\, P_2)$, $P_2{=}(pick(PS{\Rightarrow}r_{good}, alrm(1){\Rightarrow}r_{bad}))$, $r_{good}{=}(reply(User))$, $r_{bad}{=}([reply(User)]_{bad})$

In Algorithm 1, lines 5, 11, 14, and 22 are used for the purpose of bad state propagation as described in Section V-C. Lines 2, 9, 16, and 19 are used for synthesizing the local time constraint with conjunction and disjunction operation. Given a constraint tuple $c_s = synConsAOLTS(s)$, if state s is marked with a bad status during bad state propagation then $c_s.First = false$, otherwise $c_s.First = true$. Algorithm 2 is the entry algorithm that returns the local time constraint given initial state s_0 of a service S. If s_0 is not marked with a bad status, then it returns the local time constraint, otherwise, it returns false, i.e., there is no parameter valuation for component services that could possibly satisfy the local time requirement of service S.

E. Application to the Running Example

Fig. 10 shows the AOLTS of the running example introduced in Section II. Algorithm *LocalTimeConstraint* is used to synthesize the local time constraint for *SMIS* based on the AOLTS. The local time constraint is simplified and convert to disjunctive normal form (DNF) by using Z3 [6], and the local time constraint of the running example is:

$$\begin{array}{l} \quad (t_{FS} < 1 \wedge t_{DS} \leq 3 \wedge t_{DS}{+}t_{FS} \leq 3) \\ \vee\ (t_{PS} < 1 \wedge t_{DS} \leq 3 \wedge t_{FS} \leq 1 \wedge t_{DS}{+}t_{FS} \leq 3) \\ \vee\ (t_{PS} < 1 \wedge t_{DS} \leq 3 \wedge t_{FS} \geq 1 \wedge t_{DS}{+}t_{PS} \leq 2) \end{array}$$

This result, as a whole, provides us useful information regarding how the component services collectively satisfy the global time constraint. That is of most importance for selecting component services. By observing the conjuncts of DNF, we can extract useful information as well. For example, $(t_{PS} < 1 \wedge t_{DS} \leq 3 \wedge t_{FS} \geq 1 \wedge t_{DS}{+}t_{PS} \leq 2)$ conveys to us that, if the free service *FS* takes more than one second to

response and if the database service *DS* is fast enough (e.g., it is a service over local network), then we need a paid service *PS* that responds in less than one second in order for the SMIS to fulfill the global time constraint.

F. Soundness

In the following, we assume that all loops have a bound on the number of iterations and the execution time (see Section V-G for discussion).

Lemma 1: Given $\mathcal{M}$ be an abstract service model, the LTS of $\mathcal{M}$ is acyclic.

Proof: This holds due to the assumption on the loop activities such that the upper bound on the number of iterations and the time of execution is known, and there is no recursive activities in BPEL. ■

Lemma 2: Let $s_0, \ldots, s_n$ be a complete run in LTS, and π be a parameter valuation, if $\pi \models s_n.C$ then $\pi \models s_i.C$, for all $i \in \{1, \ldots, n-1\}$.

Proof: Since there is only conjunction of new constraints (refer to Section IV-C) when moving one step to the other. Therefore $s_i.C \subseteq s_{i-1}.C$ for $i \in \{1, \ldots, n\}$, and the result holds. ■

Theorem 3 (Soundness of LocalTimeConstraint): Given service $\mathcal{M} = (Var, V_0, U, P_0, C_0)$, $K = LocalTimeConstraint(\mathcal{M})$. Let S_{good} be the collection of good states in S and S_{bad} be the collection of all bad states in S, and. If K is non-empty,

1) There is at least a good state $s_g \in S_{good}$ that is reachable in any concrete LTS of $\mathcal{M}[\pi]$;
2) There is no bad state $s_b \in S_{bad}$ that is reachable in any concrete LTS of $\mathcal{M}[\pi]$.

Proof: Before bad state propagation, by definition, there will have at least an edge from non-terminal state. We show that after bad state propagation, all kind of non-terminal states (and-states, or-states, inter-states) that are not marked with a bad status will have at least an edge that leads to a good state in the abstract LTS of $\mathcal{M}[\pi]$. For and-states and inter-states, if there exists one child that is marked with a bad status, then the and-states and normal-states should be marked with a bad status (line 14 and line 22). This means that the and-states that are not marked with a bad status have all their transitions preserved in the abstract LTS. Therefore, regardless which transitions are removed in the concrete LTS, there is at least one transition left (the concrete LTS will execute at least one transition). For the or-states, if all the child states are marked with a bad status, then the or-states will be marked with a bad status, therefore if they are not marked with a bad status, they would have at least one transition left in the abstract state (line 11). Since any concrete LTS does not remove the transition from an or-state, therefore all transitions of an or-state remains. By Lemma 1, since the LTS is finite and acyclic, it will have at least one of the transitions that ends in the terminal state. Let state s_g be a good state, and T_G be the global time constraint, by Lemma 2, $\pi \models s_g.C \wedge s_g \leq T_G$ implies that $\pi \models s'.C$, where s' is any non-terminal state in the complete runs between initial state and good state s_g. Therefore item 1 holds. Since the result is of the form $(K_1 \vee K_2 \vee \ldots \vee K_n) \wedge K_{bad}$, this can be rewritten as $((K_1 \wedge K_{bad}) \vee (K_2 \wedge K_{bad}) \vee \ldots \vee (K_n \wedge K_{bad}))$. Since K_{bad} contains negation of all such reachable bad states, therefore all the bad-states are non-reachable, and item 2 holds. ■

G. Discussion

We discuss our method and show its advantages and limitations in this section.

Termination. Our method is guaranteed to terminate. This is due to the fact that BPEL composite services do not support recursion, as well as our assumption on the loop activities such that the upper bound on the number of iterations and the time of execution is known. We discuss how to enforce such assumption if the loop activities exist. For the upper bound on the number of iterations, it could be either provided by the user, or it could be inferred by using loop bound analysis tool, e.g., [7]. For the maximum time of loop executions, this could in fact be enforced by using proper timeout mechanism in BPEL.

Time for internal operations. For simplicity, we do not account for the time taken for the internal operations of the system. In reality, the time taken by the internal operations might be significant especially when the process is large. In order to provide a more accurate synthesis of time constraint, an additional constraint $t_{overhead} \leq b$, where $t_{overhead} \in \mathbb{R}_{\geq 0}$ is a time overhead for an internal operation, and $b \in \mathbb{R}_{\geq 0}$ is a machine dependent upper bound for $t_{overhead}$, could be included for more precise analysis. The method in estimation of b is not the focus of this work; the interested reader may refer to, e.g., [8].

Completeness. A limitation of the method is that it is incomplete, i.e., it does not include all parameter valuations that could give a solution to the problem of the local time requirement. The method could also return an empty constraint, although there exists parameter valuation that satisfies the solution. This can be seen as a tradeoff of making the synthesized local time requirement more general, i.e., to hold in any composite service instances. Consider the conditional case $A \lhd 1{=}1 \rhd B$, where activity A is always executed, but activity B is never executed. Since our method is completely abstracted from data, the constraint of executing activity A, as well as activity B, are both considered; but, in fact, only activity A needs to be considered, therefore our method has discarded some feasible parameter valuation in this case. In practice, such situation could be mitigated by proper program analysis method.

VI. Related Work

This work shares common techniques with work for constraint synthesis for scheduling problems. The use of models such as Parametric Timed Automata (PTA) [9] and Parametric Time Petri Nets (TPNs) [10] for solving such problems has received recent attention. In particular, in [11], [12], [13], parametric constraints are inferred, guaranteeing the feasibility

of a schedule using PTAs with stopwatches. In [4], we extended the "inverse method" (see, e.g., [14]) to the synthesis of parameters in a parametric, timed extension of CSP. Although PTAs or TPNs might have been used to encode (part of) BPEL language, our work is specifically adapted and optimized for synthesizing local timing constraint in the area of service composition. The quantitative measure of the robustness of concurrent timed systems has been tackled in different papers (see [15] for a survey). However, most approaches consider a single dimension ϵ: transitions can usually be taken at most ϵ (before or after) units of time from their original firing time. This can be seen as a "ball" in $\mid U \mid$ dimensions of radius ϵ. In contrast, our approach quantifies robustness for all parameter dimensions, in the form of a polyhedron in $\mid U \mid$ dimensions.

Our method is related to using LTS for analysis purpose in Web services. In [16], the author proposes an approach to obtain behavioral interfaces in the form of LTS of external services by decomposing the global interface specification. It also has been used in the model checking the safety and liveness properties of BPEL services. For example, Foster et al. [17], [18] transform BPEL process into FSP, subsequently using a tool named as WS-Engineer for checking safety and liveness properties. Simmonds et al. [19] proposes a user-guided recovery framework for Web services based on LTS. Our work uses LTS in synthesizing local time requirement dynamically.

Our method is related to the finding of a suitable quality of service (QoS) for the system [20]. The authors of [20] propose two models for the QoS-based service composition problem [21] model the service composition problem as a mixed integer linear problem where constraints of global and local component serviced can be specified. The difference with our work is that, in their work, the local constraint has been specified, whereas for ours, the local constraints is to be synthesized. An approach of decomposing the global QoS to local QoS has been proposed in [22]. It uses the mixed integer programming (MIP) to find optimal decomposition of QoS constraint. However, the approach only concerns for simplistic sequential composition of Web services method call, without considering complex control flow and timing requirement.

Our method is related to response time estimation. In [23], the authors propose to use linear regression method and a maximum likelihood technique for estimating the service demands of requests based on their response times. [24] has also discussed the impact of slow services on the overall response time on a transaction that use several services concurrently. Our work is focused on decomposing the global requirement to local requirement, which is orthogonal to these works.

VII. Evaluation

Here we report about the evaluation of our approach using two case studies. Each case study consists of a service composition in the form of a BPEL process. The experiment data were obtained on a system using Intel Core I5 2410M CPU with 4GB RAM.

A. Computer Purchasing Services

The goal of the computer purchasing service (CPS) (e.g., Dell.com) is to allow a user to purchase the computer system online using credit cards. CPS makes use of five component services, namely Shipper Service (SS), Logistic Service (LS), Inventory Service (IS), Manufacture Service (MS), and Billing Service (BS). The global time requirement of the CPS is to response within three seconds. CPS starts upon receiving the purchase request from the client with credit card information, and CPS spawns three workflows (viz., shipping workflow, inventory workflow, and billing workflow) concurrently. In the shipping workflow, the shipping service provider is invoked synchronously for the shipping service on computer systems. Upon receiving the reply, LS which is a service provided by internal logistic department is invoked synchronously to record the shipping schedule. In the manufacture workflow, IS is invoked synchronously to check for the availability of the goods. Subsequently, MS is invoked asynchronously to update the manufacture department regarding the current inventory stock. In the billing workflow, the billing service which is offered by third party merchant, is invoked synchronously for billing the customer with credit card information. The AOLTS of this system contains 457 states and 6355 transitions. The time taken for the synthesis with the simplification to DNF takes 2 seconds. The local time constraint for CPS contains a disjunction of 12 conjuncts in the DNF, that are given below.

$$
\begin{array}{l}
\quad (t_{SS}{+}t_{LS}{\le}t_{IS}){\wedge}(t_{IS}{\le}t_{BS}){\wedge}(t_{SS}{+}t_{LS}{+}t_{IS}{+}t_{BS}{\le}3)\\
\vee\ (t_{SS}{+}t_{LS}{\le}t_{BS}){\wedge}(t_{BS}{\le}t_{IS}){\wedge}(t_{SS}{+}t_{LS}{+}t_{IS}{+}t_{BS}{\le}3)\\
\vee\ (t_{SS}{+}t_{LS}{\le}t_{BS}){\wedge}(t_{SS}{\le}t_{IS}){\wedge}(t_{IS}{\le}t_{SS}{+}t_{LS}){\wedge}(t_{SS}{+}t_{LS}{+}t_{IS}{+}t_{BS}{\le}3)\\
\vee\ (t_{SS}{\le}t_{IS}){\wedge}(t_{IS}{\le}t_{BS}){\wedge}(t_{BS}{\le}t_{SS}{+}t_{LS}){\wedge}(t_{SS}{+}t_{LS}{+}t_{IS}{+}t_{BS}{\le}3)\\
\vee\ (t_{SS}{+}t_{LS}{\le}t_{IS}){\wedge}(t_{SS}{\le}t_{BS}){\wedge}(t_{BS}{\le}t_{SS}{+}t_{LS}){\wedge}(t_{SS}{+}t_{LS}{+}t_{IS}{+}t_{BS}{\le}3)\\
\vee\ (t_{SS}{\le}t_{BS}){\wedge}(t_{BS}{\le}t_{IS}){\wedge}(t_{IS}{\le}t_{SS}{+}t_{LS}){\wedge}(t_{SS}{+}t_{LS}{+}t_{IS}{+}t_{BS}{\le}3)\\
\vee\ (t_{SS}{+}t_{LS}{\le}t_{BS}){\wedge}(t_{IS}{\le}t_{SS}){\wedge}(t_{SS}{+}t_{LS}{+}t_{IS}{+}t_{BS}{\le}3)\\
\vee\ (t_{SS}{\le}t_{BS}){\wedge}(t_{IS}{\le}t_{BS}){\wedge}(t_{BS}{\le}t_{SS}{+}t_{LS}){\wedge}(t_{SS}{+}t_{LS}{+}t_{IS}{+}t_{BS}{\le}3)\\
\vee\ (t_{IS}{\le}t_{BS}){\wedge}(t_{BS}{\le}t_{SS}){\wedge}(t_{SS}{+}t_{LS}{+}t_{IS}{+}t_{BS}{\le}3)\\
\vee\ (t_{SS}{+}t_{LS}{\le}t_{IS}){\wedge}(t_{BS}{\le}t_{SS}){\wedge}(t_{SS}{+}t_{LS}{+}t_{IS}{+}t_{BS}{\le}3)\\
\vee\ (t_{SS}{\le}t_{IS}){\wedge}(t_{BS}{\le}t_{SS}){\wedge}(t_{IS}{\le}t_{SS}{+}t_{LS}){\wedge}(t_{SS}{+}t_{LS}{+}t_{IS}{+}t_{BS}{\le}3)\\
\vee\ (t_{IS}{\le}t_{SS}){\wedge}(t_{BS}{\le}t_{IS}){\wedge}(t_{SS}{+}t_{LS}{+}t_{IS}{+}t_{BS}{\le}3)
\end{array}
$$

Note that t_{MS} does not appear in the local time constraint for CPS. The reason is that MS is invoked asynchronously without expecting a response; therefore its response time is irrelevant to the global time requirement of CPS. Given each service with their maximum response time as a constraint, e.g., $c_1 = t_{IS} \le 0.2 \wedge t_{LS} \le 0.2 \wedge t_{SS} \le 0.5 \wedge t_{BS} \le 0.5$, and the local time constraint C_L, it is useful to know whether c_1 is included in C_L. This can be checked automatically using SMT solver such as Z3 [6], and concluded that c_1 is included in C_L.

B. Travel Booking Service

The goal of a travel booking service (TBS) (such as Booking.com) is to provide a combined flight and hotel booking service by integrating two independent existing services. TBS provides an SLA for its subscribed users, saying that it must respond within three seconds upon request. The travel booking

system has five component services, user validation service (VS), flight service (FS), backup flight service (FS_{bak}), hotel service (HS) and backup hotel service (HS_{bak}). Upon receiving the request from users, TBS spawns two workflows (viz., a flight request workflow, and a hotel request workflow) concurrently. In flight request workflow, it starts by invoking FS, which is a service provided by a flight service booking agent. If service FS does not respond within two seconds, then FS is abandoned, and another backup flight service FS_{bak} is invoked. If FS_{bak} returns within one second, then the workflow is completed; otherwise it is considered as a failure for the flight request workflow. The hotel request workflow shares the same process as the flight request workflow, by replacing FS with HS and FS_{bak} with HS_{bak}. The resulting AOLTS has 705 states with 3412 transitions, and it takes 1.5 seconds for synthesizing the constraints, and simplify the constrains into DNF. The local time constraint for TBS contains a disjunction of 12 conjuncts in the DNF, that are shown below.

$$
\begin{array}{l}
\quad (t_{HSbak}\leq 1)\wedge(t_{FSbak}\leq 1)\wedge(t_{FS}\leq 1)\wedge(t_{HS}\geq 2)\wedge(t_{FS}+t_{HSbak}\leq 1)\\
\vee\ (2t_{HS}\geq t_{FS})\wedge(t_{HSbak}\leq 1)\wedge(t_{FSbak}\leq 1)\wedge(t_{FS}+t_{HS}\leq 3)\wedge(t_{HS}\leq 2)\\
\vee\ (t_{HSbak}\leq 1)\wedge(t_{FSbak}\leq 1)\wedge(t_{HS}\leq 1)\wedge(t_{FS}\geq 2)\wedge(t_{FSbak}+t_{HS}\leq 1)\\
\vee\ (t_{HS}\leq 2t_{FS})\wedge(t_{HSbak}\leq 1)\wedge(t_{FSbak}\leq 1)\wedge(t_{FS}+t_{HS}\leq 3)\wedge(t_{FS}\leq 2)\\
\vee\ (t_{HSbak}\leq 1)\wedge(t_{FS}\leq 1)\wedge(t_{HS}\geq 2)\wedge(t_{FS}+t_{HSbak}\leq 1)\\
\vee\ (t_{FS}\leq 2t_{HS})\wedge(t_{HSbak}\leq 1)\wedge(t_{FS}\leq 2)\wedge(t_{FS}+t_{HS}\leq 3)\wedge(t_{HS}\leq 2)\\
\vee\ (t_{HS}\leq 2t_{FS})\wedge(t_{HSbak}\leq 1)\wedge(t_{FS}\leq 2)\wedge(t_{FS}+t_{HS}\leq 3)\\
\vee\ (t_{FS}\leq 2t_{HS})\wedge(t_{HS}\leq 2)\wedge(t_{FSbak}\leq 1)\wedge(t_{FS}+t_{HS}\leq 3)\\
\vee\ (t_{HS}\leq 1)\wedge(t_{FSbak}\leq 1)\wedge(t_{FS}\geq 2)\wedge(t_{FSbak}+t_{HS}\leq 1)\\
\vee\ (t_{HS}\leq 2t_{FS})\wedge(t_{HS}\leq 2)\wedge(t_{FSbak}\leq 1)\wedge(t_{FS}+t_{HS}\leq 3)\wedge(t_{FS}\leq 2)\\
\vee\ (t_{FS}\leq 2t_{HS})\wedge(t_{HS}\leq 2)\wedge(t_{FS}\leq 2)\wedge(t_{FS}+t_{HS}\leq 3)\\
\vee\ (t_{HS}\leq 2t_{FS})\wedge(t_{HS}\leq 2)\wedge(t_{FS}\leq 2)\wedge(t_{FS}+t_{HS}\leq 3)
\end{array}
$$

Note that the constraint contains inequalities such as $t_{HS} \geq 2$. This could be counterintuitive, as it asserts the response time of hotel service be larger than 2. This inequality is introduced by, for example, the `<pick>` activity (refer to state s_4 in *Fig*. 4). But one should view the local time constraint as whole, i.e., view it as all possible sets of parameter valuations that could satisfy the constraint.

VIII. Conclusion and Future Work

We have presented a novel technique for synthesizing the local time requirement for the component services of a composite service *S*, knowing its global time requirement.

Our approach is based on the dynamic analysis of the AOLTS of a composite service by making use of parameterized timed techniques. The synthesis algorithm conjuncts and disjuncts the constraint of all good states in the AOLTS to synthesize a local time constraint for component services. We have implemented the approach in the PAT tool [25] and applied it to two case studies.

We plan to further improve and develop the technique presented in this paper. First, we will consider various heuristics that could be used to reduce the number of states and transitions. Second, we will investigate the combination of our approach with other approaches such as the "inverse method" [4] to evaluate the possibility of synthesizing a better local time requirement. Last, we could extend our current approach to other domains that share similar problems, for example sensor networks.

References

[1] A. Alves, A. Arkin, S. Askary, C. Barreto, B. Bloch, F. Curbera, M. Ford, Y. Goland, A. Guízar, N. Kartha, C. K. Liu, R. Khalaf, D. König, M. Marin, IBM, V. Mehta, S. Thatte, D. van der Rijn, P. Yendluri, and A. Yiu, *Web Services Business Process Execution Language Version, version 2.0*, April 2007.

[2] R. Alur and D. Dill, "A theory of timed automata," *Theoretical computer science*, vol. 126, no. 2, pp. 183–235, 1994.

[3] J. Sun, Y. Liu, J. S. Dong, and X. Zhang, "Verifying stateful timed CSP using implicit clocks and zone abstraction," in *ICFEM 2009*. Springer, 2009, pp. 581–600.

[4] É. André, Y. Liu, J. Sun, and J.-S. Dong, "Parameter synthesis for hierarchical concurrent real-time systems," in *ICECCS 2012*. IEEE Computer Society, 2012, pp. 253–262.

[5] A. Schrijver, *Theory of linear and integer programming*. John Wiley and Sons, 1986.

[6] L. M. de Moura and N. Bjørner, "Z3: An efficient SMT solver," in *TACAS 2008*, ser. LNCS. Springer, 2008, pp. 337–340.

[7] A. Ermedahl, C. Sandberg, J. Gustafsson, S. Bygde, and B. Lisper, "Loop bound analysis based on a combination of program slicing, abstract interpretation, and invariant analysis," in *WCET 2007*, 2007.

[8] O. Moser, F. Rosenberg, and S. Dustdar, "Non-intrusive monitoring and service adaptation for WS-BPEL," in *WWW 2008*, 2008, pp. 815–824.

[9] R. Alur, T. A. Henzinger, and M. Y. Vardi, "Parametric real-time reasoning," in *STOC 1993*. ACM, 1993, pp. 592–601.

[10] L.-M. Traonouez, D. Lime, and O. H. Roux, "Parametric model-checking of stopwatch Petri nets," *Journal of Universal Computer Science*, vol. 15, no. 17, pp. 3273–3304, 2009.

[11] A. Cimatti, L. Palopoli, and Y. Ramadian, "Symbolic computation of schedulability regions using parametric timed automata," in *RTSS 2008*. IEEE Computer Society, 2008, pp. 80–89.

[12] T. T. H. Le, L. Palopoli, R. Passerone, Y. Ramadian, and A. Cimatti, "Parametric analysis of distributed firm real-time systems: A case study," in *ETFA 2010*. IEEE, 2010, pp. 1–8.

[13] L. Fribourg, D. Lesens, P. Moro, and R. Soulat, "Robustness analysis for scheduling problems using the inverse method," in *TIME 2012*. IEEE Computer Society Press, 2012, pp. 73–80.

[14] É. André and R. Soulat, *The Inverse Method*. ISTE Ltd and John Wiley & Sons Inc., 2013.

[15] N. Markey, "Robustness in real-time systems," in *SIES 2011*. IEEE, 2011, pp. 28–34.

[16] D. Bianculli, D. Giannakopoulou, and C. S. Pasareanu, "Interface decomposition for service compositions," in *ICSE*, 2011, pp. 501–510.

[17] H. Foster, "A Rigorous Approach To Engineering Web Service Compositions," Ph.D. dissertation, Imperial College of London, 2006.

[18] H. Foster, S. Uchitel, J. Magee, and J. Kramer, "LTSA-WS: a tool for model-based verification of Web service compositions and choreography," in *ICSE 2006*, 2006, pp. 771–774.

[19] J. Simmonds, S. Ben-David, and M. Chechik, "Guided recovery for Web service applications," in *SIGSOFT FSE 2010*, 2010, pp. 247–256.

[20] T. Yu, Y. Zhang, and K.-J. Lin, "Efficient algorithms for Web services selection with end-to-end QoS constraints," *TWEB*, vol. 1, no. 1, 2007.

[21] D. Ardagna and B. Pernici, "Global and local QoS guarantee in Web service selection." in *Business Process Management Workshops*, 2005.

[22] M. Alrifai and T. Risse, "Combining global optimization with local selection for efficient qos-aware service composition." in *WWW 2009*. ACM, 2009, pp. 881–890.

[23] S. Kraft, S. Pacheco-Sanchez, G. Casale, and S. Dawson, "Estimating service resource consumption from response time measurements," in *VALUETOOLS*, 2009, p. 48.

[24] D. A. Menascé, "Response-time analysis of composite Web services," *IEEE Internet Computing*, vol. 8, no. 1, pp. 90–92, 2004.

[25] J. Sun, Y. Liu, J. S. Dong, and J. Pang, "PAT: Towards flexible verification under fairness," ser. Lecture Notes in Computer Science, vol. 5643. Springer, 2009, pp. 709–714.