Modeling Web Services by Iterative Reformulation of Functional and Non-functional Requirements

Jyotishman Pathak, Samik Basu, and Vasant Honavar

Department of Computer Science
Iowa State University
Ames, IA 50011-1040, USA
{jpathak, sbasu, honavar}@cs.iastate.edu

Abstract. We propose an approach for incremental modeling of composite Web services. The technique takes into consideration both the functional and non-functional requirements of the composition. While the functional requirements are described using symbolic transition systems—transition systems augmented with state variables, function invocations, and guards; non-functional requirements are quantified using thresholds. The approach allows users to specify an abstract and possibly incomplete specification of the desired service (goal) that can be realized by selecting and composing a set of pre-existing services. In the event that such a composition is unrealizable, i.e. the composition is not functionally equivalent to the goal or the non-functional requirements are violated, our system provides the user with the causes for the failure, that can be used to appropriately reformulate the functional and/or non-functional requirements of the goal specification.

1 Introduction

With the recent advances in networking, computation grids and WWW, automatic Web service composition has emerged as an active area of research in both academia and industry (see [1,2] for a survey). The main objective of these approaches is to build and deploy new, value-added applications from existing ones in various domains such as e-Science, e-Business and e-Government.

Typically, automation in service composition relies on developers to formally describe a complete specification of the desired service (goal). In most situations, however, the task of developing such a complete functional description of a complex Web service is difficult and error prone as the developer is faced with the cognitive burden to deal with a large set of available components and their possible compositions. Furthermore, the existing techniques adopt a "single-step request-response" paradigm to service composition—that is, if the goal specification provided to a composition analyzer cannot be realized using the available component services, the entire process fails. Thus, it becomes the responsibility of the developer to identify the cause(s) for the failure of composition, which becomes a non-trivial task when modeling complex Web services. Additionally, baring a few approaches, most of the techniques for service composition focus only on the functional aspects of the composition. In practice, since there might be multiple component services that can provide the same functionality, it

A. Dan and W. Lamersdorf (Eds.): ICSOC 2006, LNCS 4294, pp. 314–326, 2006.

is of interest to explore the non-functional properties of the components to reduce the search space for determining compositions efficiently.

Towards this end, we introduce a framework for Modeling Service Composition and Execution (MoSCoE). Our approach allows the developer to start with an abstract, and perhaps incomplete specification of the goal (composite) service. In the event that the goal service is not realizable using the existing component services, the technique identifies the cause(s) for the failure of composition to help guide the developer in reformulating the goal specification and iterating the above process. In previous work, we modeled such a formalism for the iterative development of services in the context of sequential [3] and parallel [4] compositions of service functionalities. In this paper, we focus our attention on incorporating specification of non-functional requirements (e.g., Quality of Service) to the modeling of composite Web services in MoSCoE.

Specifically, given the component services $\texttt{STS}_1, \texttt{STS}_2, \ldots, \texttt{STS}_n$ and a goal service $\texttt{STS}_g$ as Symbolic Transition Systems (STSs), the objective is to generate a composition strategy $[\texttt{STS}_{i_1}, \texttt{STS}_{i_2}, \ldots, \texttt{STS}_{i_m}]$ (where $\texttt{STS}_{i_j}$ is deployed before $\texttt{STS}_{i_{j+1}}$), that satisfies both the functional and non-functional requirements. The non-functional requirements are quantified using *thresholds*, where a composition is said to conform to a non-functional requirement if it is below or above the corresponding threshold, as the case may be. For example, for a non-functional requirement involving the `cost` of a service composition, the threshold may provide an *upper-bound* (maximum allowable `cost`) while for requirements involving `reliability`, the threshold usually describes a *lower-bound* (minimum tolerable `reliability`). If more than one composition strategy meets the goal specifications, our algorithm generates all such strategies and ranks them. Strategies with higher rank are better than those with the lower rank in terms of meeting the non-functional requirements. For example, given two valid composition strategies A and B, if the `cost` of A is more than B, then A is ranked lower than B. It is left to the user's discretion to select the best strategy according to the requirements. Note that it is desirable to identify all the strategies, not just the best one, since the strategies are likely to be used multiple times in future to realize the goal service, and the component services that are part of the best strategy may become unavailable at the time of execution. In such situations, the user can select an alternate strategy from the generated set of alternative composition strategies.

The contributions of our work are:

1. A framework for incrementally composing complex Web services taking into consideration both the functional and non-functional requirements.
2. An algorithm for validating the conformance of composition to non-functional requirements. At its core, the algorithm relies on recursive forward-backward exploration of the search-space (various possible service compositions) to identify all possible compositions that meet the specified functional and non-functional requirements.
3. An approach to determine the causes of failures (due to violation of functional and/or non-functional requirements) of composition to assist the user in reformulation of functional and non-functional requirements of the goal specification.

The rest of the paper is organized as follows: Section 2 introduces an illustrative example used to explain the salient aspects our work, Sections 3 and 4 present a logical formalism and our algorithm for determining feasible composition strategies that meet the functional and non-functional requirements, Section 5 illustrates our approach for identifying the cause(s) for failure of composition, Section 6 briefly discusses related work, and finally Section 7 concludes with future avenues for research.

2 Illustrative Example

We present a simple example where a service developer is assigned to model a new Web service, `LoanApproval`, that allows potential clients to determine whether an amount of loan requested will be approved or not. The service takes as input the requested loan

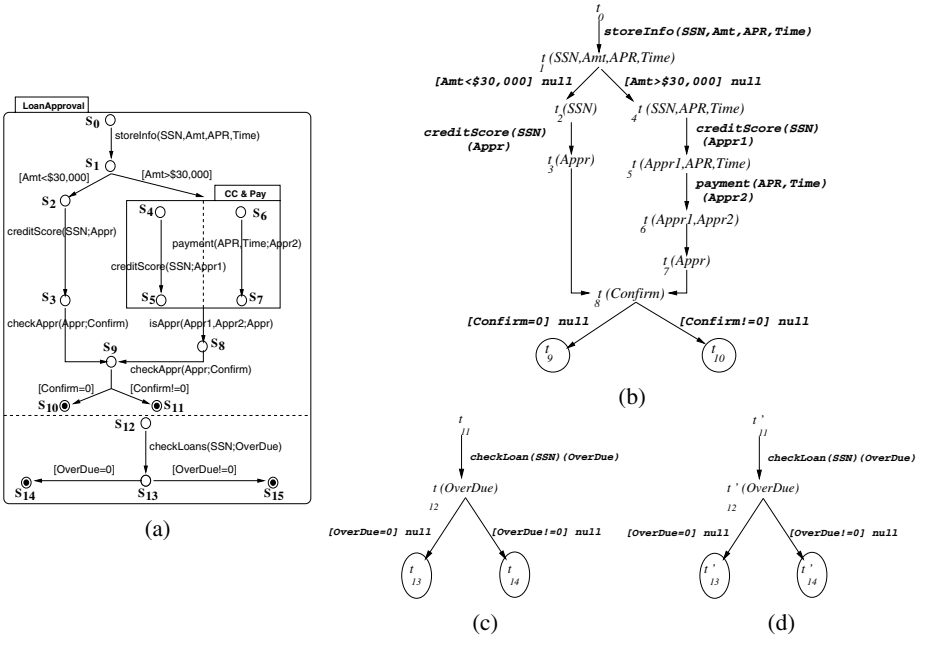

Fig. 1. (a) State machine representation of `LoanApproval` (b) STS representation of `Approver`, `cost`=$100 (c) STS representation of `Checker`, `cost`=$50 (d) STS representation of `Checker'`, `cost`=$200

amount and social security number (SSN) of the client along with the annual percentage rate (APR) and payment duration (in months) for approval of the loan. Additionally, to assist in the decision-making process, the service also checks payment overdues of the client for his/her existing loans (if any). Figure 1(a) shows the state-machine representation of such a service. Each transition labeled by functions along with their input

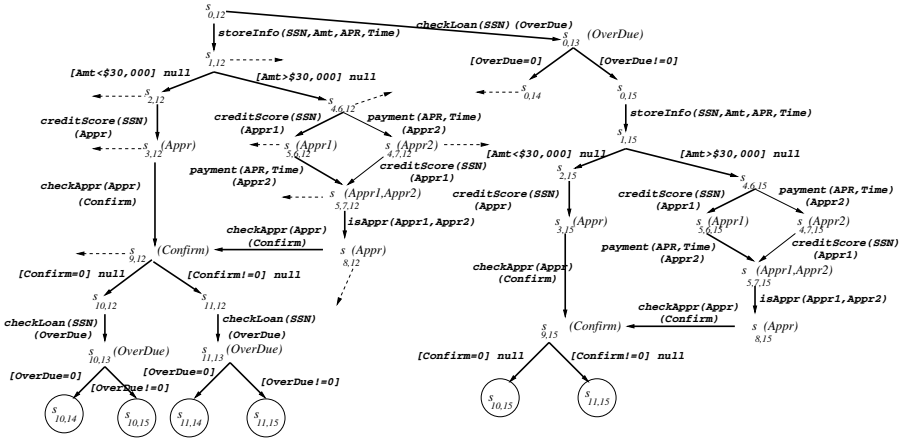

Fig. 2. Partial view of `LoanApproval` transition system

and output parameters (separated by ";"). Guards on transitions are enclosed in "[...]" and denote the conditions under which the transition is enabled. State machine representation is desirable because it allows the developer to present the functionality in a modularized and hierarchical fashion. For example, in Figure 1(a) the user modularizes the design of the desired service using composite states in the state machines; e.g. `CC & Pay` is present inside `LoanApproval`, and there are "and-" states where each partition is separated by dotted lines[1].

In MoSCoE, this goal state machine is internally translated into Symbolic Transition Systems (see Definition 1); the corresponding transition system of `LoanApproval` is presented in Figure 2. Transitions with no function invocation makes a call to a dummy function `null` and the dotted lines represent sequence of transitions (not shown) originating due to various interleaving choices of transitions in the and-partition (s_{12} in this case). Furthermore, the component services published by the service providers are also represented using STSs[2]. Figures 1(b) and 1(c) show the corresponding STS for component services `Approver` and `Checker`, respectively.

Our aim is to compose these component services to realize the goal service, thereby providing the desired capability. However, in reality, there might be multiple component services which provide the same functionality, but have different non-functional characteristics (e.g., `cost`). For example, consider services `Checker` (Figure 1(c)) and `Checker'` (Figure 1(d)), which provide the same functionality, but have different `cost` associated to their usage. Accordingly, depending on the user need, a valid composition is one which satisfies both functional and non-functional requirements. We formally describe present our approach to model such compositions in the remainder of this paper.

[1] An and-state represents the behavior where the transitions in its partition can interleave in any order.

[2] These specifications can be obtained from service descriptions provided in high-level languages such as BPEL or OWL-S by applying translators similar to those proposed in [5,6].

3 Composition Based on Functional Requirements

3.1 Modeling Services Using Transition Systems

State machines have emerged as a promising approach for modeling Web services [5,6,7,8] specifically because they possess formal semantics, have well-established modeling notations, and are intuitive and widely used in industrial software development. As mentioned earlier, our approach also relies on providing the goal specification in the form of a state machine (Figure 1(a)). In our framework, the state-machine representation is automatically translated to corresponding *Symbolic Transition Systems* (STS) [3,4]. The STS-model is used to apply the existing formalisms on transition-system "equivalence" which will be the basis for automatically identifying a valid composition strategy. Formally, an STS can be defined as:

Definition 1 (Symbolic Transition System [9]). *A symbolic transition system is a tuple* $(S, \longrightarrow, s0, S^F, A)$ *where* S *is a set of states represented by terms,* $s0 \in S$ *is the start state,* $S^F \subseteq S$ *is the set of final states and* $\longrightarrow$ *is the set of transition relations where* $s \xrightarrow{\gamma,\alpha,\rho} t$ *is such that*

1. γ *is the guard where,* $vars(\gamma) \subseteq vars(s)$
2. α *is a term representing service-functions of the form* $a(\boldsymbol{x})(y)$ *where* $\boldsymbol{x}$ *represents the input parameters and* y *denotes the return valuations*
3. ρ *relates* $vars(s)$ *to* $vars(t)$

Finally, A *is a set of non-functional attributes and the respective values corresponding to the service whose behavior is represented by the STS.*

Here, we assume that the values for non-functional attributes can be obtained from the "profiles" of the services [10] and can be mapped to a scale between 0 & 1 by applying standard mathematical maximization and minimization formulas depending on whether the attribute is *positive* or *negative*. For example, the values for attributes such as `latency` and `fault rate` need to be minimized, whereas `availability` need to be maximized. Figure 3 shows example STSs.

Semantics of STS. The semantics of STS is given with respect to substitutions of variables present in the system. A state represented by the term s is interpreted under substitution σ over the state variables ($s\sigma$). A transition $s \xrightarrow{\gamma,\alpha,\rho} t$ is said to be *enabled* from $s\sigma$ if and only if $\gamma\sigma = \texttt{true}$ and $\gamma \Rightarrow \rho$. The semantics of the transition under substitution σ is $s\sigma \xrightarrow{\alpha\sigma} t\sigma$.

3.2 Composition of Symbolic Transition Systems

A sequential composition of $\texttt{STS}_i$ and $\texttt{STS}_j$, denoted by $\texttt{STS}_i \circ \texttt{STS}_j$, is obtained by merging the final states of $\texttt{STS}_i$ with the start state of $\texttt{STS}_j$, i.e., every out-going transition of start state of $\texttt{STS}_j$ is also the out-going transition of each final state of $\texttt{STS}_i$. We say that given a goal service representation $\texttt{STS}_g$ and a set of component representations $\texttt{STS}_{1\ldots n}$, the former is said to be (partially) realizable from the latter

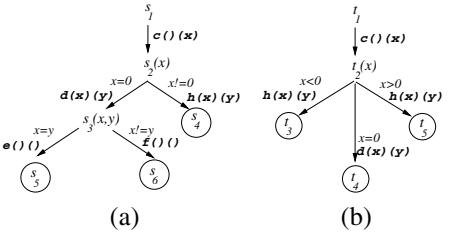

Fig. 3. Example Symbolic Transition Systems. (a) $\texttt{STS}_g$ (b) $\texttt{STS}_1$.

if there exists a composition of components such that $\texttt{STS}_g$ *simulates* $\texttt{STS}_i \circ \texttt{STS}_j \circ \ldots \texttt{STS}_k$. In essence, simulation relation ensures that the composition can 'mimic' the goal service functionality. We present the definition of simulation in the context of STSs.

Definition 2 (Late Simulation). *Given an STS $\mathcal{S} = (S, \longrightarrow, s0, S^F)$, late simulation relation with respect to substitution θ, denoted by $\mathcal{R}^{\theta}$, is a subset of $S \times S$ such that*

$$s_1 \, \mathcal{R}^{\theta} s_2 \Rightarrow (\forall s_1\theta \xrightarrow{\alpha_1} t_1\theta . \exists s_2\theta \xrightarrow{\alpha_2} t_2\theta . \forall\sigma . \alpha_1\theta\sigma = \alpha_2\theta\sigma \wedge \; t_1 \, \mathcal{R}^{\theta\sigma} t_2)$$

Two states, under the substitution θ over state variables, are equivalent with respect to simulation if they are related by the *largest* similarity relation $\mathcal{R}^{\theta}$. We say that an $\texttt{STS}_i$ is simulated by $\texttt{STS}_j$, denoted by $(\texttt{STS}_i \, \mathcal{R}^{\theta} \texttt{STS}_j)$ $\texttt{iff}$ $(s0_i \, \mathcal{R}^{\theta} s0_j)$.

For example, consider the STSs in Figure 3(a) and 3(b). If state t_1 is simulated by s_1, then $t_2(x)$ is simulated by $s_2(x)$ for all possible valuations of x. The above can be represented using logical expressions as follows:

$$\begin{aligned} t_1 \, \mathcal{R}^{\texttt{true}} \, s_1 &\Rightarrow \forall x . (t_2(x) \; \mathcal{R}^{[x]} \; s_2(x)) \\ &\Rightarrow (t_2(x) \; \mathcal{R}^{[x>0]} \; s_2(x)) \; (t_2(x) \; \mathcal{R}^{[x<0]} \; s_2(x)) \; \wedge \; (t_2(x) \; \mathcal{R}^{[x!=0]} \; s_2(x)) \\ &\Rightarrow (\forall y . (t_3 \; \mathcal{R}^{[x>0,y]} s_4)) \; \wedge \; t_4 \; \mathcal{R}^{[x=0,y]} s_3(x,y))) \; \wedge \; t_3 \; \mathcal{R}^{[x<0,y]} s_4)) \end{aligned} \tag{1}$$

Note that the simulation of the $\texttt{STS}_1$ by $\texttt{STS}_g$ leads the latter to the states $s_3(x, y)$ and s_4 with the constraints $x = 0$ and $x > 0 \vee x < 0$, respectively. In terms of sequential composition, it can be stated that a selected component (simulated by the goal) drives the goal to some specific states and the start state of the next component in the composition must be simulated by these goal states. These goal states and the corresponding constraints can be identified easily by expanding the simulation relation to also record the constraints and the goal-states that simulates the final state of the component under consideration (see [3] for details). We will use $\texttt{STS}_i \; \mathcal{R}^{\theta}_{[S_g \Delta]} \; \texttt{STS}_g$ to denote $\texttt{STS}_i$ being simulated by $\texttt{STS}_g$ under the constraint θ, which leads to the simulation of the final states of $\texttt{STS}_i$ by the goal states s_g under the constraint δ such $s_g\delta \in S_g\Delta$.

Therefore, the composition $[\texttt{STS}_1 \circ \texttt{STS}_2 \circ \ldots \circ \texttt{STS}_n]$ is said to (partially) replicate the goal $\texttt{STS}_g$ if and only if:

$$[\texttt{STS}_1 \circ \texttt{STS}_2 \circ \ldots \circ \texttt{STS}_n] \; \mathcal{R}^{\texttt{true}}_{S_g \Delta} \; \texttt{STS}_g \text{ such that } S_g \subseteq S_g^F$$

Proceeding further, we can state that:

$$\forall s_1\theta_1 \in S_1\Theta_1 : \texttt{STS}_1 \; \mathcal{R}^{\texttt{true}}_{S_1\Theta_1} \; \texttt{STS}_g \; \wedge \; [\texttt{STS}_2 \circ \texttt{STS}_3 \circ \ldots \circ \texttt{STS}_n] \; \mathcal{R}^{\theta}_{S_g\Delta} \; s_1$$

$$\Leftrightarrow \quad \forall s_1\theta \in S_1\Theta_1 : \texttt{STS}_1 \; \mathcal{R}^{\texttt{true}}_{S_1\Theta_1} \; \texttt{STS}_g \; \wedge \; \forall s_2\theta \in S_2\Theta_2 : \texttt{STS}_2 \; \mathcal{R}^{\theta_1}_{S_2\Theta_2} \; s_1 \; \wedge$$

$$[\texttt{STS}_3 \circ \ldots \circ \texttt{STS}_n] \; \mathcal{R}^{\theta_2}_{S_g\Delta} \; s_2$$

$$\text{such that } S_g \subseteq S_g^F$$

4 Composition Based on Non-functional Requirements

In order to determine a suitable ordering of the available components, it is necessary to select the appropriate components from the pool of candidates. This becomes challenge with the increasing size of the search space of available component services. Hence, we consider non-functional aspects (e.g., QoS) to winnow components (thereby reducing the search space) and composition strategies which violate the requirements desired by the user. We assume the existence of a shared controlled vocabulary [10] which is needed to specify the non-functional attributes of the component services. These attributes can be either domain-dependent or domain-independent, and are used to compose a quality matrix comprising of a set of quality attribute-values, such that each row of the matrix corresponds to the value of a particular QoS attribute and each column corresponds to a particular component service. Next, we describe an algorithm that considers both the functional and non-functional user requirements to determine feasible composition strategies as illustrated in Section 3.

4.1 Algorithm for Service Composition

The algorithm for determining feasible composition strategies (Figure 4) that are "equivalent" to the goal service works as follows: the procedure `forwardSearch([], v, g, F, OP)` is invoked by providing the threshold value `v` of a desired non-functional attribute[3] (e.g., `cost` of using the composite service should be less than $150), the start state `g` of the goal STS, an optimization function `F` that corresponds to maximization/minimization[4] of the non-functional attribute under consideration, and the composition operator `OP` for the specific optimization function. The initial composition strategy `prefixStrat` is incrementally built as the algorithm proceeds recursively by performing forward-backward traversals: The forward traversal tries to identify a feasible composition (by applying the simulation relation, Definition 2) that comply to the user-specified non-functional requirements; the backward traversal tries to explore alternate compositions (if any). If multiple compositions are identified, then it is left at user's discretion to select one amongst them.

More specifically, given a state g of the goal STS as input, the procedure `forward Search` selects a component from the available set of components `CompSet`, such

[3] In this paper, we consider only one non-functional attribute at a time for determining feasible compositions; considering multiple attributes simultaneously is part of our on-going work.

[4] Assuming that `F` is a minimization function, `F(x,y)` = 1, if `x` < `y`.

```
/*
  v0 is the user-defined threshold value, prefixStrat is the valid strategy obtained so far
  v is non-functional attribute-value prefixStrat, g is the current goal state,
  F is optimization function, OP is the composition operation for a specific optimization,
  CompSet is the set of components
*/
 1: proc forwardSearch(prefixStrat, v, g, F, OP) {
 2:   if (g is not final goal-state) {
 3:     select c_i ∈ CompSet s.t. ∀ c_j ∈ CompSet, j ≠ i : F(v_ci, v_cj) = 1 and
 4:           c_i is simulated by g to reach g';
 5:     /* update the attribute-value */
 6:     v = v OP v_ci;
 7:     if (F(v,v0)≠ 1) { backwardSearch(prefixStrat, F, OP); return; }
 8:     /* add the component with the goal-state */
 9:     prefixStrat = prefixStrat + (c_i,g);
10:     forwardSearch(prefixStrat, v, g', F, OP);
11:   }
12:   else /* if the goal state is final, strategy is achieved */ {
13:     assertPath(prefixStrat projection on components, v);
14:     backwardSearch(prefixStrat, F, OP);
15:   }
16: }

17: proc backwardSearch(prefixStrat, F, OP) {
18:   if (prefixStrat = ∅) return;
19:   prefixStrat = [(c1, g1), (c2, g2), ..., (cn, gn)];
20:   select c_i ∈ CompSet s.t. ∀ c_j ∈ CompSet, j ≠ i, F(v_cj, v_cn) ≠ 1 : F(v_ci, v_cj) = 1 and
21:         c_i is simulated by g_n to reach g';
22:   /* update the value and begin the forward search */
23:   v = v0 OP v_c1 OP v_c2 OP ... OP v_ci;
24:   if (F(v_ci,v) = 1)
25:     forwardSearch([(c1, g1), (c2, g2), ..., (ck, gi)], v, g', F, OP);
26:   backwardSearch([(c1, g1), (c2, g2), ..., (cn−1, gn−1)], F, OP)
27: }
```

Fig. 4. Algorithm for Identifying Compositions Satisfying Functional & Non-Functional User Requirements

that its value for the non-functional attribute under consideration (e.g., `cost`) is maximum/minimum (depending on `F`) and verifies whether the component state is "simulation equivalent" to the goal state g (`line 3`). If no such component exists, then an exception causing a failure of composition is raised and the user is notified (Section 5). On the other hand, if such a component is available, the value of the desired non-functional property is appropriately updated (`line 6`) and the component that simulates the goal state is added to the composition strategy (`line 9`). The procedure is recursively invoked with the updated non-functional attribute-value and a new goal state g', until the final state of the goal STS is reached; after which the corresponding composition strategy is stored with the associated non-functional attribute-value of the composition (`line 13`). This value will be either below or above the threshold v_0 (`line 7`), depending on composition optimization function `F`. Once this step is executed, the algorithm backtracks to determine alternate composition strategies that are feasible. Note that, the `forwardSearch` procedure is a local greedy approach for finding out a feasible composition and essentially identifies at least one path (beginning at the start and ending at the final state of the goal STS) in the composition graph that can be realized using the available component services.

When there are multiple components that provide the same functionality, it is possible to generate more than one composition strategy to realize the composite service. `backwardSearch` achieves this by replacing one component at a time (beginning with the last component in the composition strategy, `line 20`) with an "equivalent" component and then invokes `forwardSearch` to determine if the replacement violates the non-functional requirement under consideration (`line 24--25`). If there is no such violation, then the derived strategy is stored with its associated value of the corresponding non-functional attribute. The procedure proceeds further by recursively backtracking (`line 26`) until all feasible composition strategies are determined. However, if the replacement violates the non-functional requirements, then the replaced component as well as the corresponding composition strategy are disregarded. Thus, eliminating composition strategies (and components) that violate non-functional requirements yield significant reduction in the size of the search space.

4.2 Modeling `LoanApproval` Composite Service

We now show how to model the `LoanApproval` composite service introduced in Section 2 using the algorithm and the formalisms described above. Figures 2, 1(b) & 1(c) show the transition system of the goal (`LoanApproval`) and the component services (`Approver` and `Checker`, respectively). To determine whether `LoanApproval` can be realized from `Approver` and `Checker` services, we need to find out if $\texttt{STS}_{\texttt{LA}}$ simulates the composition of $\texttt{STS}_{\texttt{App}}$ and $\texttt{STS}_{\texttt{Chck}}$ as well as whether the non-functional requirements are met or not. Assume that the non-functional attribute we are interested is `cost`, and we want that the `cost` of the composite service is less than or equal to \$150 (i.e., minimization of `cost`).

From Figure 4, if the algorithm selects component `Approver` first, it can be seen that $\texttt{STS}_{\texttt{App}}$ is *late*-simulated by $\texttt{STS}_{\texttt{LA}}$. The path starting from $s_{0,12}$ in `LoanApproval` simulates the paths in `Approver` such that $s_{10,12}$ and $s_{11,12}$ are the states in $\texttt{STS}_{\texttt{LA}}$ that are simulation equivalent to final states t_9 and t_{10} of $\texttt{STS}_{\texttt{App}}$ under a `true` constraint. Also, the `cost` of $\texttt{STS}_{\texttt{App}}$ is less than the threshold value of \$150. Thus, this component is added to the composition strategy being constructed. Proceeding further, $\texttt{STS}_{\texttt{Chck}}$ is also simulated by states $s_{10,12}$ and $s_{11,12}$ of $\texttt{STS}_{\texttt{LA}}$, and the corresponding `cost` of $\texttt{STS}_{\texttt{Chck}}$ is \$50, which makes the total `cost` of the composition strategy equal to the threshold value. Thus, the composition of $\texttt{STS}_{\texttt{App}} \circ \texttt{STS}_{\texttt{Chck}}$ realizes the goal service $\texttt{STS}_{\texttt{LA}}$. Note that, there is another solution to the above problem where `Checker` service is followed by `Approver` service.

Once this strategy is identified, the backward traversal procedure is invoked. Here, we try to replace $\texttt{STS}_{\texttt{Chck}}$ with $\texttt{STS}_{\texttt{Chck}'}$ (Figure 1(d)) since it can also be simulated by $\texttt{STS}_{\texttt{LA}}$. However, replacing $\texttt{STS}_{\texttt{Chck}}$ by $\texttt{STS}_{\texttt{Chck}'}$ violates the `cost` requirement.

5 Reformulation of Goal Specification

The composition of a goal service from available component services using the process outlined above will fail when some aspect of the goal specification cannot be realized

using the available component services. When this happens, our approach seeks to provide to the user, information concerning the cause of the failure in a form that can be used to further refine the goal specification. In our framework, the reason for failure of an attempted composition to simulate a component by single or multiple states in the goal is obtained by examining the simulation relation $\mathcal{R}$:

$$s_1 \; \overline{R}^{\theta} \; s_2 \; \Leftarrow \; \exists s_1\theta \xrightarrow{\alpha_1} t_1\theta . \forall s_2\theta \xrightarrow{\alpha_2} t_2\theta . \exists\sigma . (\alpha_1\theta\sigma = \alpha_2\theta\sigma) \Rightarrow t_1 \; \overline{R}^{\theta\sigma} \; t_2 \qquad (1)$$

Two states are said to be *not* simulation equivalent if they are related by the least solution of $\overline{\mathcal{R}}$. We say that $\texttt{STS}_i \; \overline{\mathcal{R}}^{\theta} \; \texttt{STS}_j$ `iff` $s_i^0 \; \overline{\mathcal{R}}^{\theta} \; s_j^0$. From Equation 1, the cause of the state s_1 *not* simulated by s_2 can be due to:

1. $\exists\sigma.\alpha_1\theta\sigma \neq \alpha_2\theta\sigma$ (i.e., that actions do not match), or
2. $\exists\sigma.\alpha_1\theta\sigma = \alpha_2\theta\sigma$ and the subsequent states are related by $\overline{\mathcal{R}}^{\theta\sigma}$, or
3. $\exists s_1\theta \xrightarrow{\alpha_1} t_1\theta$, but there is no transition enabled from s_2 under the substitution θ.

For example, consider the `STS`s in Figure 5(a) & 5(b). The component `STS` is not simulated by the first $\texttt{STS}_g$ (rooted at s_1) as there exists a transition from $t_2(x)$ to t_4 when the x is not equal to zero, which is absent from the corresponding state $s_2(x)$ in the goal. The state t_1 is also not simulated by state s_4 of $\texttt{STS}_{g'}$ as the state $t_2(x)$ is not simulated by $s_5(y)$. This is because x and y may not be unified as the former is generated from the output of a transition while the latter is generated at the state. In fact, a state which generates a variable is not simulated by any state if there is a guard on the generated variable. Such generated variables at the states are *local* to that transition system and hence, cannot be 'mimicked' by another transition system. In our example, $t_2(x)$ is not simulated by $s_5(y)$.

Note that in some cases, the failure to realize a feasible composition can also be due to non-compliance of non-functional requirements specified by the user. In essence, this refers to argument `prefixStrat` of procedure `forwardSearch` (Figure 4) being `null` after exploring all possible composition strategies. When such a situation arises, the framework identifies the particular non-functional requirement (v_0) that cannot be met using the available components, and provides this information to the service developer such that it can be used for appropriate refinement of the requirement.

Failure-Cause Analysis for `LoanApproval`. Returning to our example from Section 2, assume that we replace the `Checker` component service (Figure 1(c)) with another service for determining client payment overdues (`NewChecker` Figure 5(c)), which

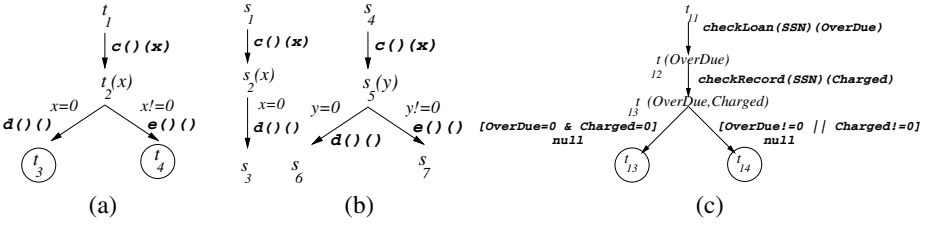

Fig. 5. (a) Component `STS` (b) Goal STSs $\texttt{STS}_g$ & $\texttt{STS}_{g'}$ (c) STS for `NewChecker`

functions exactly like `Checker`, but additionally checks the criminal record of the client. The service first checks whether the client has payment overdues for the existing loans (if any) and then determines if the client has been previously charged for a criminal act. Since, the 'additional' criminal verification transition is not present in $\mathtt{STS}_{\mathrm{LA}}$,, the component start state t_{11} is not simulated by states $s_{10,12}$, $s_{11,12}$ or $s_{0,12}$. On the other hand, assuming that we replace `Checker` with `Checker'` (Figure 1(d)), even though the functional requirements are satisfied (due to equivalence), the non-functional requirements are violated. Note that such failure-cause information can be provided to the user which can be used for refining the goal specification in an iterative manner. In this case the user can add the criminal verification transition (with appropriate parameters) to the goal specification or change the threshold value of the non-functional attribute `cost` and try to determine a feasible composition strategy. These steps can be iterated until such a strategy is found or the user decides to abort.

6 Related Work

A number of approaches have been proposed in the literature which adopt a transition-system based framework to service composition. Fu et al. [11] model Web services as automata extended with a queue, and communicate by exchanging sequence of asynchronous messages, which are used to synthesize a composition for a given specification. Their approach is extended in Colombo [8] which deals with infinite data values and atomic processes. Colombo models services as labeled transition systems and define composition semantics via message passing, where the problem of determining a feasible composition is reduced to satisfiability of a deterministic propositional dynamic logic formula. Pistore et al. [5,6] represent Web services using non-deterministic state transition systems, which also communicate through messaging. Their approach relies on symbolic model checking techniques to determine a parallel composition of all the available component services and then generates a plan that controls the services, based on user-specified functional requirements.

Several techniques have also been developed which consider non-functional requirements for service composition. Cardoso et al. [12] describe a model that allows prediction of quality of service for workflows based on individual QoS attributes for the component services. Their technique allows compensation of composition deficiency if many services with compatible functions exist. Benatallah et al. [7,13] consider service selection task as a global optimization problem and apply linear programming to find solution that represents service composition optimizing a target function, where the function is defined as a combination of multiple non-functional parameters. Yu and Lin [14] modeled the service selection as a complex multi-choice multi-dimension 0-1 knapsack problem, which takes into consideration difference in QoS parameters offered by multiple services by assigning weights.

The proposed framework, MoSCoE, is inspired by and builds on the above mentioned approaches. One of the unique features of MoSCoE is its ability to work with abstract (and possibly incomplete) goal service specifications for realizing composite services, and in the event of failure of composition, determining the cause(s) for the failure. In addition, provides an approach to consider both functional and non-functional

characteristics of services simultaneously to determine feasible composition strategies. We believe that such a technique holds the promise of efficiently modeling complex composite Web services; empirically verifying this claim is part of our current work.

7 Summary and Discussion

We introduce a novel approach for automatically developing composite services by the applying the techniques of abstraction, composition and reformulation in an incremental fashion. The framework provides a goal-directed approach to service composition and adopts a symbolic transition system-based approach for computing feasible composition strategies. Our formalism allows us to identify and validate all possible composition strategies that meet the user-specified functional and non-functional requirements. However, in the event that a composition cannot be realized using the existing set of candidate services, the technique determines the cause(s) for the failure (due to violation of functional and/or non-functional requirements), and assists the user in reformulation of those requirements in the goal specification.

Our on-going work is aimed at developing heuristics for hierarchically arranging failure-causes to reduce the number of refinement steps typically performed by the user to realize a feasible composition. We also plan to explore approaches to reducing the number of candidate compositions that need to be examined e.g., by exploiting domain specific information to impose a partial order over the available services. Other work in progress is aimed at automatic translation of the composition strategy into BPEL process flow code that can be executed to realize the composite service. Of particular interest to us is a systematic evaluation of scalability and efficiency of the proposed approach on a broad class of benchmark service composition problems [15]. More details about our framework can be obtained from `http://www.moscoe.org`.

Acknowledgment. This research has been supported in part by the NSF-ITR grant 0219699 to Vasant Honavar and NSF grant 0509340 to Samik Basu. The authors would like to thank Robyn Lutz for her help in preparing this manuscript.

References

1. Hull, R., Su, J.: Tools for Composite Web Services: A Short Overview. SIGMOD Record **34**(2) (2005) 86–95
2. Dustdar, S., Schreiner, W.: A Survey on Web Services Composition. International Journal on Web and Grid Services **1**(1) (2005) 1–30
3. Pathak, J., Basu, S., Lutz, R., Honavar, V.: Selecting and Composing Web Services through Iterative Reformulation of Functional Specifications. In: 18th IEEE International Conference on Tools with Artificial Intelligence. (2006)
4. Pathak, J., Basu, S., Lutz, R., Honavar, V.: Parallel Web Service Composition in MoSCoE: A Choreography-based Approach. In: 4th IEEE European Conference on Web Services. (2006)
5. Pistore, M., Traverso, P., Bertoli, P.: Automated Composition of Web Services by Planning in Asynchronous Domains. In: 15th Intl. Conference on Automated Planning and Scheduling. (2005) 2–11

6. Traverso, P., Pistore, M.: Automated Composition of Semantic Web Services into Executable Processes. In: 3rd Intl. Semantic Web Conference, Springer-Verlag (2004) 380–394
7. Benatallah, B., Sheng, Q., Dumas, M.: The Self-Serv Environment for Web Services Composition. IEEE Internet Computing **7**(1) (2003) 40–48
8. Berardi, D., Calvanese, D., Giuseppe, D.G., Hull, R., Mecella, M.: Automatic Composition of Transition-based Semantic Web Services with Messaging. In: 31st Intl. Conference on Very Large Databases. (2005) 613–624
9. Basu, S., Mukund, M., Ramakrishnan, C.R., Ramakrishnan, I.V., Verma, R.M.: Local and Symbolic Bisimulation Using Tabled Constraint Logic Programming. In: Intl. Conference on Logic Programming. Volume 2237., Springer-Verlag (2001) 166–180
10. Pathak, J., Koul, N., Caragea, D., Honavar, V.: A Framework for Semantic Web Services Discovery. In: 7th ACM Intl. Workshop on Web Information and Data Management, ACM press (2005) 45–50
11. Fu, X., Bultan, T., Su, J.: Analysis of Interacting BPEL Web Services. In: 13th Intl. conference on World Wide Web, ACM Press (2004) 621–630
12. Cardoso, J., Sheth, A., Miller, J., et al.: Quality of Service for Workflows and Web Service Processes. Journal of Web Semantics **1**(3) (2004) 281–309
13. Zeng, L., Benatallah, B.: QoS-Aware Middleware for Web Services Composition. IEEE Transactions on Software Engineering **30**(5) (2004) 311–327
14. Yu, T., Lin, K.: Service Selection Algorithms for Composing Complex Services with Multiple QoS Constraints. In: International Conference on Service Oriented Computing, LNCS 3826 (2005) 130–143
15. Oh, S.C., Kil, H., and, D.L.: WSBen: A Web Services Discovery and Composition Benchmark. In: 4th International Conference on Web Services, IEEE Press (2006) 239–246