

# Encoding Requests to Web Service Compositions as Constraints

Alexander Lazovik<sup>1,2</sup>, Marco Aiello<sup>1</sup>, and Rosella Gennari<sup>2</sup>

<sup>1</sup> DIT, Trento U., via Sommarive 14, 38050 Trento, IT,  
{lazovik,aiello}@dit.unitn.it

<sup>2</sup> ITC-irst, via Sommarive 18, 38050 Trento, IT, gennari@itc.it

**Abstract.** Interacting with a web service enabled marketplace in order to achieve a complex task involves sequencing a set of individual service operations, gathering information from the services, and making choices. We propose to encode the problem of issuing requests to a composition of web services as a constraint-based problem.

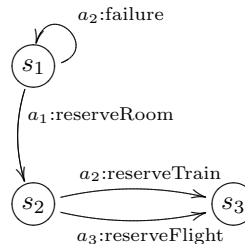
## 1 Introduction

Services are autonomous computational entities which live on a network and interact by asynchronous message passing. Services publish standard interfaces to enable their discovery, binding and invocation. The most prominent example is given by the XML-based standards known as web services, and the most interesting open challenge therein is the *service composition* problem, i.e., aggregating services for achieving complex tasks. Here we concentrate on the problem of enabling a user to express complex requests/goals against a pre-compiled composition of services in the form of a business process/domain (in this paper we use these words interchangeably). What we have is a description of a business domain (e.g., an electronic marketplace) and the user's request (e.g., the cheapest travel offer), which is satisfied by invoking the appropriate domain services. We propose to model the domain and request via constraints. Solving such constraints means finding an executable plan to satisfy the user's request in the business domain. Section 2 introduces the example which runs throughout the paper; definitions of the business domain and request language are in Section 3; the constraint encoding is in Section 4; Section 5 concludes the paper — its extended version is [1], e.g., with the encoding algorithms and related work.

## 2 Organizing a trip

Let us consider a travel marketplace and the organization of a trip. A generic trip organization can be modeled by a complex business process encompassing several actions and states. Moving from one state to another may involve the discovery of information, the choice of which action to take and even *nondeterministic actions* – i.e., their outcome states, hence their effects may be different and not

determined until execution. In [2], we showed a business process for organizing a trip with 36 states (<http://www.opentravel.org>); here we consider a subset of that process. When deciding on a trip, the user may want to book first the hotel of the final destination and then a carrier to reach the hotel location. The figure represents this business process snippet as a state transition system. The first action is the hotel reservation ( $a_1$  leaving state  $s_1$ ). This may result in the room reservation (state  $s_2$ ) or in a failure (back to  $s_1$ ); “which is which” is unknown until execution. Finally, there are two ways to reach  $s_3$ , the state in which a carrier for the hotel is booked: i.e., by either flying or taking a train. This means choosing either the `reserveTrain` action or the `reserveFlight` one. Given this, a user may also want to have a hotel reserved, prefer flying to taking a train and optionally wish to spend no more than 100 euros.



### 3 Web service interactions

Interacting with a web service enabled marketplace to achieve a complex request involves sequencing a set of individual service operations, gathering information from the services and making choices. The complex request of the user is similar to a planning goal, while the business process describing the possible behaviors of the marketplace is similar to a planning domain. Here we propose to model the business domain and the user’s request via constraints. The business domain is a state-transition system with one characterizing peculiarity: *non-deterministic actions*. Formally, the domain is a tuple of states, actions, variables, failure states, and a transition function; we refer to [2] for the definition. Here we only note that the transition function maps a state and an action into a *set of states and an individual state*. The rationale is that of all the states an action reaches, one is the action’s *normal* outcome, while the others are the action’s *failure* states. The *request language* definition is derived from [2]: basic requests are **vital**  $p$ , **atomic**  $p$ , **vital-maint**  $p$ , **atomic-maint**  $p$ , with  $p$  a proposition. A request  $g$  is a basic request or of the form **achieve-all**  $g$ , **optional**  $g$ , **before-then**  $g$ , **prefer-to**  $g$ . Having an initial state and the user’s request  $g$ , a *plan* is given by sequences of actions (of the business domain) that leave from the initial state and satisfy the user’s request. In Section 4, we obtain a plan by encoding domains and requests as numeric constraints.

Let us reconsider our example of Section 2 and the figure therein: the set of states  $\mathcal{S}$  is  $\{s_1, s_2, s_3\}$ , the set of actions  $\mathcal{A}$  is  $\{\text{bookHotel } a_1, \text{reserveTrain } a_2, \text{reserveFlight } a_3\}$ , and the set of variables is  $\{\text{price}, \text{hotelBooked}, \text{trainBooked}, \text{flightBooked}\}$ . The first variable ranges over natural numbers while all the other variables are Boolean. As for the transition function, `bookHotel`  $a_1$  brings the system nondeterministically into  $\langle\{s_1, s_2\}, s_2\rangle$ , which means that  $s_2$  is the normal state, whereas  $s_1$  is the failure state. As for the actions’ effects on variables, we have: the normal `bookHotel` action  $a'_1$  increases *price* and sets *hotelBooked*





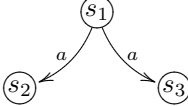
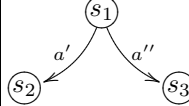
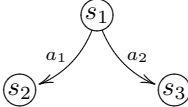
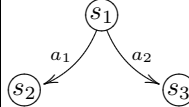
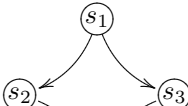
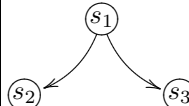
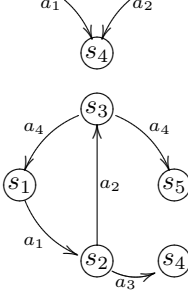
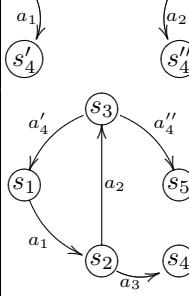
to 1 (i.e., true); the failure `bookHotel` action  $a_1''$  has no effect on the variables; the `reserveTrain` action  $a_2$  increases `price` and sets `trainBooked` to 1; the `reserveFlight` action  $a_3$  increases `price` and sets `flightBooked` to 1. The request in Section 2 is now **achieve-all** (**vital** `hotelBooked = 1`; **atomic-maint** `price < 100`; **prefer** (**vital** `flightBooked = 1` to **vital** `trainBooked = 1`)).

## 4 Constraint-based encoding of the business domain

Services offer a set of independently invocable operations. The operations act on a number of variables whose values may depend on a single service invocation or, more generally, on a number of invocations on several independent services. Here, constraints model how the values of a variable spanning across such services may change. Additionally, the user has requests and preferences in achieving complex tasks. We model these via additional constraints on the service domain. In particular there are two types of Boolean variables: *controlled* variables, denoted by  $\beta_i$ , and *non-controlled* variables, denoted by  $\xi_i$ . The rationale is that the constraint system may not choose values for non-controlled variables, and then a solution to the problem is such regardless of their assignments. We also assume that, once executed, a nondeterministic action has always its first execution outcome. The constraints of our encoding have the form  $[\forall \xi_i :] \bar{c}_v \bowtie value$  where: *value* is a value from the domain of the variable  $v$ ;  $\bar{c}_v$  is a vector of expressions of the form  $\sum \beta_i [\xi_i] a_{i,k}$  (with  $\beta_i, \xi_i \in \{0, 1\}$ ), the  $\xi_i$  are non-controlled variables,  $a_{i,k}$  is the effect of the action  $a_i$  for the outcome  $k$ ,  $\bowtie$  is in  $\{<, >, \geq, \leq, =\}$ , and  $[\cdot]$  denotes that the enclosed expression may not occur in the constraint. Formally, a *service constraint problem* is a tuple  $\mathcal{CP} = \langle \beta, \mathcal{N}, \xi, \mathcal{C} \rangle$ , where  $\beta$  is a set of *controlled* Boolean variables,  $\mathcal{N}$  is a set of *controlled* variables over  $\mathbb{N}$ ,  $\xi$  is a set of *non-controlled* Boolean variables,  $\mathcal{C}$  is a set of constraints as above, in which a non-controlled variable is (i) either universally quantified over, (ii) or a value is available and substituted for it. A *solution* to a service constraint problem is an assignment to controlled variables such that all the problem constraints are satisfied. The encoding of the service interaction problem is split in the domain encoding (*phase 1*), and the request encoding (*phase 2*).

*Phase 1 (domain encoding)*: given a business domain and an initial state  $s$ , the domain-encoding returns a set of constraints  $c_v$  as above. In what follows,  $n$  represents the number of times a cycle is followed, while  $a_i$  represents not only the action but also its effects. The following table briefly illustrates such encoding.

*Phase 2 (request encoding)*: the user's request is added and encoded as follows. **vital**  $v \bowtie v_0$ : if the request is **vital** with respect to  $v$  constrained by  $\bowtie$  on  $v_0$ , the  $v$  encoding in the constraint vector  $c$  (denoted by  $c_v$ ) is considered and it is added to the  $c_v \bowtie v_0$  constraint set. Also all the  $\xi_v$  variables associated with  $c_v$  are set to  $\xi_v^0$ , that is, the normal execution must be followed. **atomic**  $v \bowtie v_0$ : as above, but all nondeterministic executions are considered, thus all non-controlled variables  $\xi$  get universally quantified over.

Business domain		Constraint encoding	
	No action		$\emptyset$
	Single action		$\beta a$
	Nondeterministic action		$\beta(\xi_1 a' + \xi_2 a'')$ $\xi_1 + \xi_2 = 1$
	Diverging actions		$\beta_1 a_1 + \beta_2 a_2$ $\beta_1 + \beta_2 \leq 1$
	Converging actions		$\emptyset$
	Loop with a nondeterministic action		$n\xi(a_1 + a_2 + a'_4)$

**vital-maint**  $v \bowtie v_0$ : all the states visited during execution are considered. One quantifies over the execution steps, repeating the constraints as in the vital case above for each step.

**atomic-maint**  $v \bowtie v_0$ : as above, but all nondeterministic executions are considered, so all non-controlled variables  $\xi$  get universally quantified over.

**achieve-all**  $g_1 \dots, g_n$ : all sub-requests  $g_1, \dots, g_n$  are recursively executed; all basic requests coming from these are thus considered. If during the execution some choices are made for the same branch point among different sub-requests, then these choices are forced to be always the same by introducing a controlled variable  $u$ . E.g., suppose that  $w^j, j \in \{1, 2\}$ , denotes the branch chosen for trying to satisfy the  $j$ -th request;  $w^j = 0$  expresses that no choices were made; then  $w^1 \neq 0 \wedge w^2 \neq 0 \Rightarrow w^1 = w^2$  is added as constraint.

**before**  $g_1$  **then**  $g_2$ : as above, but one tries to satisfy first  $g_1$  then  $g_2$  is.

**prefer**  $g_1$  **to**  $g_2$ : the request variables are instantiated along a certain order. Optional requests are prefer-to request with  $g_2$  equal to true.

*The travel example encoded.* Let us spell out part of the encoding of the example from Section 2. Its domain and initial state  $s_1$  give the constraint  $\beta_1(\xi_1 n a_1^{fail} +$

$\xi_2(a_1^{ok} + \beta_2 a_2 + \beta_3 a_3)$ ) which represents the paths from state  $s_1$  to  $s_3$  with  $n$  being the number of times the cycle is followed. When requests are encoded, for each basic request a new set of variables is introduced. The first sub-request to be parsed is **vital** *hotelBooked* = 1. Only the  $a_1^{ok}$  outcome affects the *hotelBooked* variable, thus the constraint is  $\beta_1' \xi_2 a_1^{ok} = 1$  and the non-controlled variables are assigned to normal executions, i.e.,  $\xi_2 = 1, \xi_1 = 0$ . The other **vital** requests are treated similarly. The request of preferring flying to taking the train gives the assignment  $\beta_i''' = 1$  and  $\beta_i^{iv} = 0$  as first, for all  $i \in \{1, \dots, 3\}$ . A solution is  $\beta_1^{(j)} = 1, \beta_2^{(j)} = 0, \beta_3^{(j)} = 1$ , for all  $j \in \{1, \dots, 4\}$ . This corresponds to booking the hotel (**bookHotel**) and reserving a flight (**reserveFlight**), assuming that the total price is less than 100. However, if the flight price is 200, the above is no longer a solution; but the preference constraint allows for an assignment which is a solution, that is, by taking the train (**reserveTrain**) instead of the plane (if the total cost is less than 100).

## 5 Concluding remarks

We propose to model business domains and users' requests via numeric constraints. Pivotal properties of the encoding are its dealing with nondeterministic actions, its being unbounded, its capability of representing the possible executions of domain actions; these are relevant features in a web service enabled marketplaces, and make the encoding a major improvement with respect to [2]. In particular, here we deal with numeric requests without encoding them into Boolean properties. Moreover, we also handle users' preference requests. A number of issues remain open. Most notably, we have not yet assessed the efficiency of the proposed algorithms with respect to the minimality of the encoding. We have not considered the framework in the context of interleaving planning and execution, nor with respect to run-time information gathering. The last is a crucial issue in a web service scenario. However, we have preliminary results in extending the presented work in this direction.

*Acknowledgments:* R. Gennari is supported by the project grant *Automated Reasoning by Constraint Satisfaction* from the Province of Trento.

## References

1. A. Lazovik, M. Aiello, and R. Gennari. Encoding requests to web service compositions as constraints. Technical Report DIT-05-40, Univ. of Trento, 2005. <http://www.dit.unitn.it/~aiellom/publications/DIT-05-40.pdf>.
2. A. Lazovik, M. Aiello, and M. Papazoglou. Planning and monitoring the execution of web service requests. *Journal on Digital Libraries*, 2005. To appear.