Encoding Requests to Web Service Compositions as Constraints

Alexander Lazovik[1,2], Marco Aiello[1], and Rosella Gennari[2]

[1] DIT – University of Trento
Via Sommarive, 14
38050 Trento
Italy
{lazovik,aiellom}@dit.unitn.it

[2] ITC-irst
Via Sommarive, 18
38050 Trento
Italy
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 programming problem. In particular, we provide algorithms to encode nondeterministic web service business processes and user goals given in an expressive request language.

1 Introduction

Services are autonomous computational entities which live on a network and interact by asynchronous message passing. Services publish standard interfaces to enable the discovery, binding and invocation of their services. The most prominent example of services are the XML based set of standards known as web services. The increasing size of the Internet and the availability of more and more on-line services, opens a myriad of opportunities for developing loosely coupled applications based on third party services.

The most interesting open challenge in the field today is the problem of aggregating several services in order to achieve a complex task, this problem is known as the *service composition* problem. There are many dimensions to this problem which span from selecting services to achieve a goal, considering non-functional properties of the composition, performing automatic composition based on semantic descriptions, providing tools for designing compositions, etc.

We concentrate on the problem of enabling a user to express complex requests against a pre-compiled composition of services in the form of a business process. If the latter is generic enough to describe all possible interactions with those web services, what we have is a description of a domain, e.g., of an electronic

marketplace. Then the user request amounts to invoking the appropriate services following the constraints given by the business process and by the goal of the user. In this paper, we propose to encode the service domain and the user goal as constraint programming. In particular, we provide algorithms to move from service domain representations and goals to a set of constraints. The request language can express achievement of goals, sequencing of goals, maintaining of properties, and preferences. Solving of the constraints represents finding an executable plan to satisfy the user's request.

The remainder of the paper is organized as follows. Section 2 introduces a trip organization example which runs throughout the paper. Definitions of the goal language and planning are provided in Section 3. The algorithms to perform the encoding of the problem of interacting with web services as constraint programming are given and explained in Section 4. Related work in the fields of constraints, AI and service-oriented computing is surveyed in Section 5. Finally, Section 6 contains concluding remarks.

2 Organizing a trip

Nowadays standard business and terminology descriptions are given in XML schemas, e.g., for the automotive industry, tourism industry, chemical industry and so on (e.g., `http://xml.coverpages.org/xmlApplications.html`). In the near future, it is expected that abstract definitions of such business process will be given in BPEL [2] or similar service choreography languages. 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 steps. Moving from one step to another may involve the discovery of information, the choice of which step to take or even the nondeterminism involved in a step of which the outcome is not known until execution. In [9], we used a business process for organizing a trip consisting of 36 states and based on a OTA specification (`http://www.opentravel.org`). Here we look at a subset of that business process which will help to ground the intuitions behind the proposed encoding.
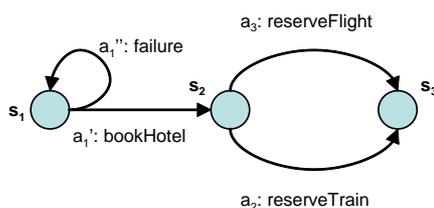

Fig. 1. A component of a travel business process

When deciding on a trip, the user may first want to book the hotel of the final destination and then book a carrier to reach the location of the hotel. We

represent this business process snippet as a state transition system in Figure 1. The first action is that of reserving a hotel (state s_1). This action may nondeterministically result in the successful booking of the room (state s_2) or in a failure (return to state s_1). Finally, there are two ways to reach the state s_3 in which a carrier to arrive at the site of the hotel is booked. One may choose to fly or to take a train. This is achieved by choosing one of the two actions `reserveTrain` or `reserveFlight`.

Given the above scenario, a specific user may desire to have a hotel reserved; it can also happen that the user prefers flying to taking a train to her/his final destination, and to spend no more than 100 euros.

3   Web service interactions as planning and execution

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. Planning offers a natural model for this behavior. The complex task of the user is similar to a planning goal, while the business process describing the possible behaviors of the marketplace is similar to a planing domain. The achievement of the user's goal yields a plan, that is sequences of actions, and its execution.

We propose to use constraint programming techniques to implement such a planning model for web service interaction. The service planning domain, i.e., the business process, is encoded as a constraint-based problem together with the encoding of the goal, i.e., the user's requests. Then, putting together the encoding of the domain and the encoding of the goal means representing all possible executions which are plans satisfying the goal.

Next we introduce the formal notions of domain and goal that we use in this paper, and show how we translate them into constraints.

3.1   Service domain and goals

The service planning domain is a state-transition system with one characterizing peculiarity: nondeterministic actions. Formally, we define such a domain as a tuple of states, actions, variables, failure states, and an effect function.

Definition 1 (service planning domain). *A service planning domain $\mathcal{D}$ is a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{V}, \mathcal{T}, f \rangle$, where:*

- *$\mathcal{S}$ is a set of states;*
- *$\mathcal{A}$ is a set of actions;*
- *$\mathcal{V}$ is a set of variables of arbitrary domains;*
- *$\mathcal{T} : \mathcal{S} \times \mathcal{A} \to 2^{\mathcal{S}} \times \mathcal{S}$ is the transition function;*
- *$f : \mathcal{A} \times \overline{\mathcal{V}} \to \overline{\mathcal{V}}$ is the effect function.*

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 reached with

an action one of them is a *normal* outcome state, while all the others represent some kind of *failure* states for that action. Moreover, the effect function expresses every possible action outcome.

Goal	Where satisfied	Type of goal
vital p	In a state where p holds to which there is a path from the initial state modulo failures	reachability
atomic p	In a state where p holds to which there is a path from the initial state despite failures	reachability
vital-maint p	In a state to which there is a path from the initial state modulo failures. p must hold in all states along the path	maintainability
atomic-maint p	In a state to which there is a path from the initial state despite failures. p must hold in all states along the path	maintainability
prefer g_1 **to** g_2	In states where g_1 is satisfied, otherwise the satisfiability of g_2 is checked	preference
optional g	States where g is satisfied are checked first, otherwise the goal is ignored	preference
before g_1 **then** g_2	In states, to which there is a path from the initial state, such that, states along these path where g_1 is satisfied precede those where g_2 is satisfied	sequencing
achieve-all $g_1, \ldots, g_n$	In states, to which there is a path from the initial state, such that, there are states along the path where g_i are satisfied	composition

Table 1. Goal language constructs.

The goal language is derived from our earlier work using a model based planner [9] which in turn is based on temporal logics. Here we give the basic definition of the language and ground the intuitions about the language constructs in Table 1. The goal language is recursively defined as follows.

Definition 2 (goal language). *Basic goals are formed using the following unary operators:* **vital** p, **atomic** p, **vital-maint** p, **atomic-maint** p, *where p is a proposition. A goal g is a basic goal or a combination of goals using the following operators:* **achieve-all** g, **optional** g, **before-then** g, **prefer-to** g.

We are now ready to provide a general definition of what a planning problem is in our setting, that is with respect to our service planning domain and goal.

Definition 3 (service planning problem). *A* service planning problem *is a tuple* $\mathcal{P} = \langle \mathcal{D}, s_0, g \rangle$*, where $\mathcal{D}$ is a domain from Definition 1, s_0 is an initial state, and g is a goal from Definition 2. A set of sequences of actions is a plan π; this is solution to the planning problem $\mathcal{P}$ if all possible executions of the plan, starting in s_0, satisfy g.*

For the precise definition of goal satisfaction we refer the reader to [9]. In Section 4, we explain how we obtain a plan by translating goals and domains into numeric constraints.

3.2 Planning a trip

Let us come back to our travel example of Section 2 and define the service planning domain according to Definition 1. Referring to Figure 1, the set of states $\mathcal{S}$ is $\{s_1, s_2, s_3\}$, the set of actions $\mathcal{A}$ is $\{$bookHotel a_1, reserveTrain a_2, reserveFlight a_3 $\}$, while the set of variables is $\{price, hotelBooked, trainBooked, flightBooked\}$. In this example the first variable is a non-negative integer and all other variables are boolean. As for the transition function, bookHotel a_1 brings the system nondeterministically into two possible states $\langle\{s_1, s_2\}, s_2\rangle$; s_2 is the normal state, while s_1 is the failure state. The other two elements of the transition function are evident from Figure 1. Finally, for the effect functions we have the following situation, which we present informally:

- normal bookHotel action a_1' increases $price$ and sets the boolean variable $hotelBooked$ to true (i.e., the value 1);
- failure bookHotel action a_1'' has no effect on the variables;
- reserveTrain action a_2 increases the $price$ and sets the boolean variable $trainBooked$ to true;
- reserveFlight action a_3 increases the $price$ and sets the boolean variable $flightBooked$ to true;

We assume initial values for the variables to be equal to zero. As for the goal introduced in Section 2 using the goal language of Definition 2, we have:

achieve-all
    vital $hotelBooked$ = 1
    atomic-maint $price$ < 100   (goal 1)
    prefer vital flightBooked = 1
        to vital trainBooked = 1

4 Constraint-based encoding of the service planning domain

Services offer a set of independently invocable operations. The operations act on a number of variables whose value may depend on a single service invocation or, more generally, it may depend on a number of invocations on several independent services. We model the way in which the values of a variable spanning across services may change by means of constraints. Additionally, the user has goals and preferences in achieving complex tasks that guide the service invocations. We model the latter as additional constraints on the service domain.

In particular in the framework we propose, there are two types of Boolean variables: *controlled* variables, denoted by β_i, and *non-controlled* variables, denoted by ξ_i. The underlying idea is that the system is not necessarily free to

choose a specific value for a non-controlled variable, thus a solution to the problem may be such regardless of the values assigned to the non-controlled variables.

Note 1. In this paper, when we talk about nondeterministic actions we refer to their outcomes (that is, states) which can be different; yet, once an action is invoked, we assume that its outcome will be always the same. In other words, any of its future invocations will produce the same outcome.

The rationale for the framework we propose is that of modeling a service domain and user's goal as a set of constraints over controlled and non controlled variables. The constraints have the following form:

$$[\forall \xi_i :] \ \overline{c_v} \ \bowtie \ value \tag{1}$$

where:

- $value$ is a value from the domain of the variable v,
- $\overline{c_v}$ 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 ξ_i are non-controlled variables and the β_i are controlled variables,
- $a_{i,k}$ is the effect function of the action a_i for the outcome k,
- $\bowtie$ is either $<, >, \geq, \leq$ or $=$,
- and $[\cdot]$ denote that the expression is optionally present in the constraint.

Formally, the service constraint problem can be formulated as follows.

Definition 4 (service constraint problem). *A* service constraint problem *is a tuple* $\mathcal{CP} = \langle \beta, \mathcal{N}, \xi, \mathcal{C} \rangle$*, where:*

- β *is a set of* controlled *boolean variables;*
- $\mathcal{N}$ *is a set of* controlled *variables ranging over natural numbers;*
- ξ *is a set of* non-controlled *boolean variables;*
- $\mathcal{C}$ *is a set of constraints, as in Equation 1, in which (i) if a non-controlled variable occurs then it is universally quantified, (ii) otherwise a value is available and substituted for the variable.*

A solution *to a service constraint problem is an assignment to controlled variables such that all constraints are satisfied.*

To arrive at the encoding of the service interaction problem as a set of constraints of the form of Equation 1, we follow a two phase process. In the first phase, one encodes the service planning domain, while in the second phase one encodes the goal.

Phase 1. Domain encoding: the service domain is encoded as a vector of constraints C. Effect functions can be partially specified as numeric values.

Phase 2. Goal encoding: the encoding is completed when the goal is given; then the required symbols and universal quantifiers are added.

4.1 Phase 1: domain encoding

During Phase 1 the service domain is encoded. Starting from a service domain as in Definition 1, we arrive at a set of expressions c_v as in Equation (1) plus a set of linear constraints of the form $\sum \beta_i \leq 1$. In the following, we adopt the notation of Equation (1); in addition, n is a natural number that specifies how many times a cycle is followed, while a_i is overloaded to represent not only the action, but also its effects. From here onwards, we only consider linear effect functions.

The encoding is generated following an algorithm that recursively visits the service domain $\mathcal{D}$, separately keeping track of cycles, and returns a set of constraints. The algorithm (Algorithm 1) begins from an initial state $s \in \mathcal{S}$ from the service planning domain $\mathcal{D}$, and an empty path P (which represents the working path followed on the graph), and returns a vector of constraints; that is, one constraint for every variable in the domain $\mathcal{V}$. It also updates a global *cycle* data structure to record constraints associated with cycles. The algorithm uses a set of variables to mark states as visited in order to detect directed and undirected cycles. Let us consider the various cases of the algorithm, for which we also provide a pictorial explanation in Table 2.

Base case. If the degree of the arcs leaving the state s is 0, then there is no constraint to be returned. Table 2.(A) illustrates this situation. Also the case of the directed cycle, which is presented below, is a base case.

Cycles. With the path sequence P and the visited variables one detects cycles in the state-action graphs. We distinguish two types of cycles: undirected ones and directed ones. In the first case, we simply 'duplicate' the state and proceed in our analysis; in the second case, we add counters to represent the number of times the cycle is followed.

Cycles: state splitting. If one has already visited a state, but the state is not part of the current path P, we have detected an undirected cycle. To proceed we need to duplicate the state s already visited by creating state s' and recursively encode the duplicated state. This situation is illustrated in Table 2.(E). There is no specific encoding for this case.

Cycles: directed cycle. If one has already visited a state and the state is part of the current path P, we have detected a directed cycle. The Algorithm 3 is invoked to encode the cycle constraint which is added to the step where the cycle originated. The constraint resulting from the invocation of the cycle algorithm is stored in the global variable *cycle* together with information on where the cycle began (given by the state s and path P). No constraint is directly added to the expression, so the return value of this case is $\emptyset$.

The Algorithm 3 works by first identifying the cycle. This is done by going backwards in the path P until the same state as the last one is found. Then all the actions involved in the cycle are identified. For each one of the actions which is nondeterministic we add a non-controlled variable ξ_j which multiplies the whole expression, because a cycle is only possible if corresponding nondeterministic actions keep inside the cycle. Then we sum all the actions effects involved in the
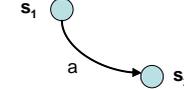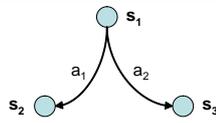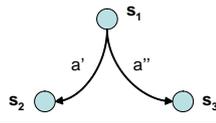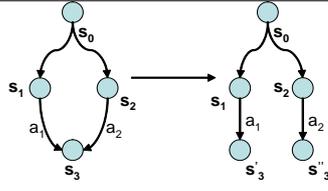
	Domain	Type of action	Encoding
(A)	s ◯	No action	$\emptyset$
(B)	s_1 ◯ $\xrightarrow{a}$ s_2 ◯	Single deterministic action	βa
(C)	s_1 ◯ $\overset{a_1}{\swarrow}$ $\overset{a_2}{\searrow}$ s_2 ◯ s_3 ◯	Deterministic branch point	$\beta_1 a_1 + \beta_2 a_2$ $\beta_1 + \beta_2 \leq 1$
(D)	s_1 ◯ $\overset{a'}{\swarrow}$ $\overset{a''}{\searrow}$ s_2 ◯ s_3 ◯	Nondeterministic branch point	$\beta(\xi_1 a' + \xi_2 a'')$ $\xi_1 + \xi_2 = 1$
(E)	s_0 ... s_1 s_2 a_1 a_2 s_3 → s_0 s_1 s_2 a_1 a_2 s'_3 s''_3	Cycle: state splitting	$\emptyset$
(F)	s_1 a'_4 s_5 a''_4 a_1 a_2 s_3 s_2 a_3 s_4	Cycle: directed cycle	$n\xi(a_1 + a_2 + a'_4)$
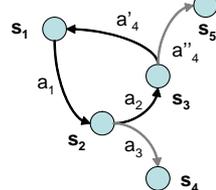
Table 2. Encoding examples of the service domain.

cycle. Finally, we multiply the whole expression for a natural number n which represents the number of times the cycle is followed.

Table 2.(F) exemplifies a cycle situation in which there are simple deterministic actions (a_1), actions taking out of the cycle (a_3), and nondeterministic actions that 'might' lead out of the cycle (a_4''). The expression on the right in Table 2.(F) is what is added to variable *cycle*.

Branch point. Of all the actions leaving from the current state one of these will be performed at the next step. The constraint is a sum of all the possible executed actions which calls the function `add-single-action` (Algorithm 2) containing a mutual recursive call. If there is a cycle associated to the current state in the path, then also the cycle constraint is added by the `get` function. With the β_i one pinpoints which one of the actions is actually executed (at most one β is set to one and all the others to 0).

Algorithm 2 distinguishes between the case the action is deterministic and nondeterministic. In the latter case, non-controlled variables ξ_i are introduced for every possible outcome of the nondeterministic action. Only one of the outcomes will occur. If, on the other hand, the action is deterministic, mutual recursion is called on the next state of the domain to which the action leads. In Table 2.(B & C) examples of one and two deterministic actions are presented, respectively. In Table 2.(D) the case of a nondeterministic action with two possible outcomes is shown.

Algorithm 1 `encode`(state s, domain $\mathcal{D}$, path P): constraint

---

  if `degree`(s) $== 0$ **then**
    // recursion base case
    return $\emptyset$
  end if
  $P = P \cup s$
  if s is visited $\wedge\ s \in P$ **then**
    // visited state: cycle is added, base case
    `put`(s,P,`add-cycle`(s, $\mathcal{D}$, P),*cycle*)
    return $\emptyset$
  end if
  if s is visited $\wedge\ s \notin P$ **then**
    // visited state: splitting point
    $s' = $ `split`(s)
    return `encode`(s', $\mathcal{D}$, P)
  end if
  `set visited` s
  // branch point
  $c = \sum^{\text{degree}(s)} \beta_i$ `add-single-action`(s, $\mathcal{D}$, a_i, P)+`get`(s,P,*cycle*)
  $\sum^{\text{degree}(s)} \beta_i \leq 1$, $\beta_i \in \{0,1\}$
  $P = P \backslash s$
  return c

Algorithm 2 `add-single-action`(state s, domain $\mathcal{D}$, action a, path P): constraint

---
    if a is nondeterministic **then**
        // nondeterministic branch point
        return $\sum^{\text{outcomes}(a)} \xi_i$ `add-single-action`(s, $\mathcal{D}$, a_i', P)$\bigcup$
            $\bigcup \sum^{\text{outcomes}(a)} \xi_i = 1$, $\xi_i \in \{0, 1\}$
    end if
    // single deterministic action outcome
    $s_{next} =$ `get-next-state`(s, $\mathcal{D}$, a)
    return $a +$ `encode`($s_{next}, \mathcal{D}, P$)

Algorithm 3 `add-cycle`(state s, domain $\mathcal{D}$, path P): constraint

---
    // identify last cycle in the path
    $P = \langle \ldots s, t_1, \ldots, t_r, s \rangle$;
    // let a_i be the action going from state t_i to t_{i+1}
    $\forall$ nondeterministic actions ad a ξ_j
    return $n \prod \xi_j \cdot \sum^{r+1} a_i$

4.2 Phase 2: goal encoding

During the second phase of the encoding of the service planning problem, one takes a goal and the encoding of the service domain, and produces a set of constraints which represent how to achieve the user's goal. The goal is expressed in the goal language of Definition 2. By parsing the goal, Algorithm 4 works on the constraints generated in Phase 1 and generates a set of constraints of the form of Equation 1.

We present the encoding of the goal into the domain by following Algorithm 4. The algorithm parses the goal recursively distinguishing the cases of the various operators and updating the set of constraints. Every time a new basic goal constraint is added, a new set of controlled variables is introduced. Relations between controlled variables of different sub-goals are defined by **achieve-all** and **before-then**.

vital $v \bowtie v_0$**.** If the goal is **vital** with respect to the variable v constrained by the $\bowtie$ operator on the v_0 value, then we take the v line in the constraint vector c (denoted by c_v) and we add it to the constraints set $c_v \bowtie v_0$. Since the goal is vital we also set all variables ξ_v associated with c_v to ξ_v^0, by which we mean that the normal execution must be followed, in place of the nondeterministic failure ones.

atomic $v \bowtie v_0$**.** This case is analogous to the vital one, with the difference that all nondeterministic executions must be considered, therefore we come to a universal quantification over the nondeterministic variables ξ.

vital-maint $v \bowtie v_0$**.** For maintainability goals we need to keep track of all the states visited during a plan execution. This information is what is stored in the set P in Algorithm 1. Thus, we quantify over the execution steps and we repeat the constraint as for the vital case above for each step.

atomic-maint $v \bowtie v_0$**.** This case is analogous to the maintainability vital one, with the difference that all nondeterministic executions must be considered, therefore we come to a universal quantification over the nondeterministic variables ξ.

Now we consider the operators which aggregate basic sub-goals.

achieve-all $g_1 \ldots, g_n$**.** First, recursion is called for all sub-goals $g_1, \ldots, g_n$. Second, one considers all pairs of basic goals coming from the recursive call and all execution steps (as done for the maintainability goals). In all these cases, if during the execution some choices have been made for the same branch point among different sub-goals, these choices have to be the same. Therefore, we add expression forcing the choices for the execution of any subgoals to be the same to the set of constraints. The expressions introduce the execution choice variable u. Suppose that $u^j, j \in \{1, 2\}$ denotes the branch that has been chosen by the procedure that tries to satisfy an j-th goal, $u^j = 0$ defines that there were no choice made. Then the following constraints ar added: $u^1 \neq 0 \wedge u^2 \neq 0 \Rightarrow u^1 = u^2$.

before g_1 **then** g_2**.** The principle behind the before-then operator is similar to that of the achieve-all, with the difference that one forces the ordering of the satisfaction of the subgoals. Again, first we recur on the subgoals, then we introduce the execution choice variables u. The second subgoal g_2 should repeat the path of the first subgoal g_1, until the first is satisfied, and only then the second expression is checked. This is ensured by expressions of the form: $u^1 \neq 0 \Rightarrow u^1 = u^2$ which are added to the set of constraints.

prefer g_1 **to** g_2**.** Preferences are handled not as additional constraints, but rather appropriately instantiating the variables. The first step is to recur on the two subgoals g_1 and g_2. Then the goals g_1 and g_2 are placed in a disjunction. When constraints are checked for satisfiability, variables are assigned in preference order. Optional goals are a sub-case of the prefer-to goal, in which g_2 is simply true.

4.3 The example encoded

Let us come back to the travel example of Sections 2 and 3.2 and show how Algorithms 1–4 encode it into a constraint-based problem. Algorithm 1 takes the domain exemplified in Figure 1 and the initial state s_1 and produces the following constraints: $\beta_1(\xi_1 na_1^{fail} + \xi_2(a_1^{ok} + \beta_2 a_2 + \beta_3 a_3))$ which represents the paths from state s_1 to s_3 with n being the number of times the cycle is followed. Additionally, it also produces the constraints on the choice variables $\beta_1, \beta_2, \beta_3 \in \{0, 1\}$, $\beta_2 + \beta_3 \leq 1$, and the constraints on the non-controlled variables $\xi_1, \xi_2 \in \{0, 1\}$, $\xi_1 + \xi_2 = 1$. ξ_1, ξ_2.

Consider again the goal (goal 1) in Section 3.2, Algorithm 4 takes the constraints set above and the goal, and builds a service constraint problem. For each basic goal a new set of free variables is introduced. The first subgoal to be parsed is **vital** $hotelBooked = 1$. Since the $hotelBooked$ variable is influenced

Algorithm 4 `encode-goal`(constraint c, goal g): goalset

// *reachability goals*
if g is 'vital $v \bowtie v_0$' **then**
    $c \bigcup \xi_v = \xi_v^0,\ c_v \bowtie v_0$
    return $\{g\}$
end if
if g is '**atomic** $v \bowtie v_0$' **then**
    $c \bigcup \forall \xi\ c_v \bowtie v_0$
    return $\{g\}$
end if
// *maintainability goals*
if g is '**vital-maint** $v \bowtie v_0$' **then**
    for all steps t_i **do**
        $c \bigcup \xi_v = \xi_v^0,\ c_v(t_i) \bowtie v_0$
    end for
    return $\{g\}$
end if
if g is '**atomic-maint** $v \bowtie v_0$' **then**
    for all steps t_i **do**
        $c \bigcup \forall \xi\ c_v(t_i) \bowtie v_0$
    end for
    return $\{g\}$
end if
if g is '**achieve-all** $g_1, \ldots, g_n$' **then**
    $G_i = $ `encode-goal`(c, g_i)
    for all $g^i \in G_i,\ g^j \in G_j,\ i \neq j$, steps t_k **do**
        $c \bigcup u^k(g_i) \neq 0 \wedge u^k(g_j) \neq 0 \Rightarrow u^k(g_i) = u^k(g_j)$
    end for
    return $\{G_1, \ldots, G_n\}$
end if
if g is '**before** g_1**then** g_2' **then**
    $G_i = $ `encode-goal`$(c, g_i), i \in \{0, 1\}$
    for all $g^1 \in G_1,\ g^2 \in G_2$, steps t_i **do**
        $c \bigcup u_i(g^1) \neq 0 \Rightarrow u_i(g^1) = u_i(g^2)$
    end for
    return $\{G_1, G_2\}$
end if
if g is '**prefer** g_1**to** g_2' **then**
    $G_i = $ `encode-goal`$(c, g_i), i \in \{0, 1\}$
    `raise-priority`(G_1, G_2)
    return $\{G_1, G_2\}$
end if
if g is '**optional** g_1' **then**
    return `encode-goal`$(c,$ **prefer** g_1**to** $\top)$
end if

---

only by a_1^{ok} outcome, then the constraint is $\beta_1' \xi_2 a_1^{ok} = 1$ and the non-controlled variables are assigned to normal execution, i.e., $\xi_2 = 1, \xi_1 = 0$.

The other **vital** goals are treated similarly. For **vital** $flightBooked = 1$ we have $\beta_1''' \xi_2 \beta_3''' = 1$ and $\xi_2 = 1, \xi_1 = 0$, while for **vital** $trainBooked = 1$ we have $\beta_1^{iv} \xi_2 \beta_2^{iv} = 1$ and $\xi_2 = 1, \xi_1 = 0$.

The atomic goal **atomic-maint** $price < 100$ is slightly different as $price < 100$ has to be checked for each state. Since it is an atomic goal, we have to introduce a universal quantification over the non-controlled variables. This goal leads to the following set of constraints: $\forall \xi : s_1' : 0 < 100, s_1'' : \beta_1'' \xi_1 na_1^{fail} < 100, s_2 : \beta_1''(\xi_1 na_1^{fail} + \xi_2(a_1^{ok})) < 100, s_3 : \beta_1''(\xi_1 na_1^{fail} + \xi_2(a_1^{ok} + \beta_2'' a_2 + \beta_3'' a_3)) < 100$.

The goal of preferring flying over taking the train does not add any constraints, but defines ordering of instatiation between the variables β_i''' over β_i^{iv} for all $i \in \{1, .., 3\}$. Finally, **achieve-all** adds the relations between the branch choices, that is, the last 6 constraints in Equation 4.3. Summarizing, the encoding of the travel example of Figure 1 and (goal 1) is the following:

$$
\begin{cases}
\beta_1' \xi_2 = 1, \text{ if } \xi_2 = 1, \xi_1 = 0 \\
\forall \xi : 0 < 100 \\
\forall \xi : \beta_1'' \xi_1 na_1^{fail} < 100 \\
\forall \xi : \beta_1''(\xi_1 na_1^{fail} + \xi_2(a_1^{ok})) < 100 \\
\forall \xi : \beta_1''(\xi_1 na_1^{fail} + \\
\quad + \xi_2(a_1^{ok} + \beta_2'' a_2 + \beta_3'' a_3)) < 100 \\
\beta_1''' \xi_2 \beta_3''' = 1 \text{ if } \xi_2 = 1, \xi_1 = 0 \vee \\
\vee \beta_1^{iv} \xi_2 \beta_2^{iv} = 1 \text{ if } \xi_2 = 1, \xi_1 = 0
\end{cases}
\quad
\begin{cases}
\forall j \in \{1, .., 4\} \, \beta_1^{(j)} \leq 1 \wedge \beta_2^{(j)} + \beta_3^{(j)} \leq 1 \\
\forall i \in \{1, .., 3\} \, \beta_i' \neq 0 \wedge \beta_i'' \neq 0 \Rightarrow \beta_i' = \beta_i'' \\
\forall i \in \{1, .., 3\} \, \beta_i' \neq 0 \wedge \beta_i''' \neq 0 \Rightarrow \beta_i' = \beta_i''' \\
\forall i \in \{1, .., 3\} \, \beta_i' \neq 0 \wedge \beta_i^{iv} \neq 0 \Rightarrow \beta_i' = \beta_i^{iv} \\
\forall i \in \{1, .., 3\} \, \beta_i'' \neq 0 \wedge \beta_i''' \neq 0 \Rightarrow \beta_i'' = \beta_i''' \\
\forall i \in \{1, .., 3\} \, \beta_i'' \neq 0 \wedge \beta_i^{iv} \neq 0 \Rightarrow \beta_i'' = \beta_i^{iv} \\
\forall i \in \{1, .., 3\} \, \beta_i''' \neq 0 \wedge \beta_i^{iv} \neq 0 \Rightarrow \beta_i''' = \beta_i^{iv}
\end{cases}
$$

One of the possible solutions is the following assignment of controlled variables: $\beta_1^{(j)} = 1, \beta_2^{(j)} = 0, \beta_3^{(j)} = 1$, for all $j \in \{1, .., 4\}$. This corresponds to the execution of booking the hotel (`bookHotel`) and reserving a flight (`reserveFlight`) assuming that the total price is less than 100. On the contrary, suppose that the flight price is 200, thus violating the price constraint, then the above assignment is no longer a solution. By the preference ordering, we still have an assignment which is a solution by using the train (`reserveTrain`) instead of the plane assuming that the total cost is less than 100: $\beta_1^{(j)} = 1, \beta_2^{(j)} = 1, \beta_3^{(j)} = 0$, for all $j \in \{1, .., 4\}$.

5 Related work

In the area of constraint satisfaction, the motivations that led to the framework of open constraint satisfaction problems are common to ours. In particular, as the author of [14] pinpoints, for one party in the service "it is often impractical [. . .] to provide the other parties [. . .] with full information on its own constraints" and thus each party "has to solve its constraint satisfaction problem

without complete knowledge"; we refer the reader to [6] for other work related to the framework of open constraint satisfaction problems. Another track of the constraint literature is focused on the resource allocation problem in a distributed scenario, and its optimization (e.g., [7]); however this is not an issue in this paper, as we abstract from this problem.

In fact, in this paper we are concerned with web service oriented business processes and interactions with them, and we proposed a modeling as constraint-based problems. Instead of having "external constraints" as, for instance, in the open constraint setting of [14], we introduced two sorts of variables in our model: variables the constraint system is free to choose values for, and variables that only external parties can choose values for. Accordingly the latter are called "non-controlled variables", and any solution to the problem should be independent of their possible values.

In service-oriented computing several initiatives have been proposed to manage and interact with web service compositions. There are approaches based on formal logics [3, 11, 12]) or other approaches based on logic programming formalisms (e.g., [10]). All these approaches work under the assumption of having available rich semantic service description and run-time information. In contrast to this, in a pure service-oriented environment on the one hand, there is little semantic description and, on the other hand, one deals with incomplete knowledge about service behavior.

Artificial Intelligence techniques can provide a solution to the problem of service composition. In particular, there have been several proposals using AI planning. In [13], a review of web service composition techniques is presented and it is argued that planning techniques can help tackling the problem of automatic web service composition. In particular, Knoblock et al. [8] use a form of template planning based on hierarchical task networks and constraint satisfaction. The authors focus on information gathering and integration rather than on service composition. Encoding planning problems as constraints is not new. For instance, [4] introduce temporal constraint networks, while planning as constraint satisfaction is proposed in [5].

In [1], we proposed a hybrid approach to web service composition and interaction based on both planning and constraint satisfaction. In that work constraint satisfaction was used to select the best plan instance of those generated by the planner and with values coming from interactions with web service implementations. In [9], we concentrated on interleaving planning and execution for satisfying user requests. In the latter work constraint satisfaction was not used, rather a planner was adapted in order to deal with numeric responses from web service instances.

6 Concluding remarks

We proposed an approach to planning interactions with web services based on a constraint programming encoding. The key characteristics of the encoding are its dealing with nondeterminism, its being unbounded, its representing the

possible executions on the domain including traversal properties. We provided algorithms for encoding state representations of the domain together with user requests given in an expressive goal language. The proposed encoding is particularly suited to deal with web service enabled marketplace and give the user the possibility to satisfy his/her desires.

This is a major improvement with respect to the previous frameworks to deal with web service requests based on a model based planner [9]. In particular, with the proposed approach we deal with numeric values in place of considering boolean conditions coming from the satisfaction of an expression; we handle preference goals by ordering variable assignments; while keeping the desired properties of dealing with nondetermisim, having primitives for execution properties (e.g., vital-maint goals) and having a framework to execute the requests.

This is an initial proposal to use constraints to encode the satisfaction of a user's request with respect to a set of autonomous web services. A number of issues are open for further investigation. Most notably, we have not yet considered issues of efficiency of the proposed algorithms with respect to the minimality of the encoding or of the propagation complexity or execution time. We have not considered the framework in the context of interleaving planning and execution, nor with respect to run-time information gathering. The latter is a crucial issue for the success of the approach as in the web service context some information is gathered only at run-time and there is constant need for replanning. We have preliminary results in extending the presented work in this direction.

References

1. M. Aiello, M. Papazoglou, J. Yang, M. Carman, M. Pistore, L. Serafini, and P. Traverso. A request language for web-services based on planning and constraint satisfaction. In F. Casati, L. Fiege, M-C. Hsu, and M-C. Shan, editors, *Technologies for E-Services (TES-02)*, volume 2444 of *Lecture Notes in Computer Science*, pages 76–85. Springer, 2002.
2. BPEL. *Business Process Execution Language for Web Services*, August 2002. `http://www-106.ibm.com/developerworks/library/ws-bpel/`.
3. L. Chen, N.R. Shadbolt, C. Goble, F. Tao, S.J. Cox, C. Puleston, and P. Smart. Towards a knowledge-based approach to semantic service composition. In G. Goos, J. Hartmanis, and J. van Leeuwen, editors, *2nd Int. Semantic Web Conf. (ISWC2003)*, LNCS 2870, pages 319–334. Springer-Verlag, 2003.
4. R. Dechter, I. Meiri, and J Pearl. Temporal constraint networks. *Artificial Intelligence*, 49:61–95, 1991.
5. M. B. Do and S. Kambhampati. Planning as constraint satisfaction: solving the planning graph by compiling it into csp. *Artificial Intelligence*, 132:151–182, 2001.
6. B. Faltings and S. Macho-Gonzalez. Open constraint satisfaction. In Pascal Van Hentenryck, editor, *Principles and Practice of Constraint Programming - CP 2002*, number 2470 in Lecture Notes in Computer Science, pages 356–370. Springer, 2002.
7. Y. Hamadi, A. M. Frisch, and I. Miguel. An overview of the gridline project. In *Planning and Scheduling for Web and Grid Services - ICAPS 2004*, 2004.
8. C. A. Knoblock, S. Minton, J. L. Ambite, M. Muslea, J. Oh, and M. Frank. Mixed-initiative, multi-source information assistants. In *Proceedings of the World Wide Web Conference*, pages 697–707. ACM Press, 2001.

9. A. Lazovik, M. Aiello, and M. Papazoglou. Planning and monitoring the execution of web service requests. *Journal on Digital Libraries*, 2005. To appear.

10. S. McIlraith and T. C. Son. Adapting Golog for composition of semantic web-services. In D. Fensel, F. Giunchiglia, D. McGuinness, and M. Williams, editors, *Conf. on principles of Knowledge Representation (KR)*, 2002.

11. Massimo Paolucci, Takahiro Kawamura, Terry Payne, and Katia Sycara. Semantic matching of web services capabilities. In I. Horrocks and J. Hendler, editors, *Int. Semantic Web Conf. (ISWC2002)*, Lecture Notes in Computer Science 2342, pages 333–347. Springer, 2002.

12. Evren Sirin, James Hendler, and Bijan Parsia. Semi-automatic composition of web services using semantic descriptions. In *Web Services: Modeling, Architecture and Infrastructure workshop in ICEIS 2003*, 2003.

13. B. Srivastava and J. Koehler. Web Service Composition - Current Solutions and Open Problems. In *Workshop on Planning for Web Services – ICAPS'03*, 2003.

14. E.P.K. Tsang. Constraint satisfaction in business processes modelling. *The Journal of Management and Economics*, 7(7), 2003.