Optimizing Web Service Composition While Enforcing Regulations

Shirin Sohrabi and Sheila A. McIlraith

Department of Computer Science, University of Toronto, Toronto, Canada
{shirin,sheila}@cs.toronto.edu

Abstract. To direct automated Web service composition, it is compelling to provide a template, workflow or scaffolding that dictates the ways in which services can be composed. In this paper we present an approach to Web service composition that builds on work using AI planning, and more specifically Hierarchical Task Networks (HTNs), for Web service composition. A significant advantage of our approach is that it provides much of the how-to knowledge of a choreography while enabling customization and optimization of integrated Web service selection and composition based upon the needs of the specific problem, the preferences of the customer, and the available services. Many customers must also be concerned with enforcement of regulations, perhaps in the form of corporate policies and/or government regulations. Regulations are traditionally enforced at design time by verifying that a workflow or composition adheres to regulations. Our approach supports customization, optimization and regulation enforcement all at composition construction time. To maximize efficiency, we have developed novel search heuristics together with a branch and bound search algorithm that enable the generation of high quality compositions with the performance of state-of-the-art planning systems.

1 Introduction

Increasingly, corporations are providing services within and between organizations by publishing programs on corporate intranets or on the World Wide Web. Many of these programs represent component software that can be composed together either manually or automatically to provide value-added service. To direct automated Web Service Composition (WSC), it is compelling to provide some sort of template, workflow or scaffolding that dictates the ways in which services can be composed while leaving enough flexibility for different possible realizations of the programs within the template. A template-based composition is compelling for many applications in domains including e-science (e.g., [1]), e-government (e.g., [2]), and Grid computing (e.g., [3]).

A WSC template is designed with respect to a particular task to be performed. It provides high-level guidance on how to perform a task, but leaves many of the details to run-time synthesis. For many WSC problems, the task can be realized by a diversity of different services, offering comparable, but not identical services. Also unknown at the outset is the data that serves as choice points

A. Bernstein et al. (Eds.): ISWC 2009, LNCS 5823, pp. 601–617, 2009.

in a WSC – the availability of goods, their properties and pricing, etc. A composition template streamlines the generation of a problem, and customer-specific WSC, while enabling the individual customer to customize the composition with respect to their preferences and constraints and/or those of the corporation they work for, the laws of the countries in which they are doing business, etc.

A composition template can be represented in a variety of different ways. One way to represent a template is to use a workflow or a flowchart. This can be expressed pictorially as a schematic or alternatively in a form akin to a procedural programming language. The Algol-inspired Golog agent programming language provides one such procedural language (e.g., [4]). Indeed, the first template-based approach to WSC exploited Golog to provide a so-called generic procedure that provided a template specification of the composition [5,6]. The Golog procedures were combined with individual user constraints (e.g., *"I want to fly with a star alliance carrier"*) at run time, resulting in *dynamic binding of Web services*. However, the user constraints considered were hard constraints, i.e., realizations that did not satisfy those constraints were eliminated. In [7], we extended this framework to be able to deal with *soft* user constraints (i.e., preferences). The proposed preference language handled a wide variety of user constraints. It enabled the synthesis of a composition of services, where the selection of services and service groundings (e.g., in the case of travel, the selection of the specific flight) was customized to individual users at run time. Unfortunately, the implementation of the system, GologPref was not optimized.

Another type of composition template that can be used is based on Hierarchical Task Networks (HTNs) [8]. Like Golog, HTNs provide useful control knowledge — advice on how to perform a composition. However, this how-to knowledge is specified as a *task network*. The task network provides a way of hierarchically abstracting the composition into a set of tasks that need to be performed and that decompose in various ways into leaf nodes realized by programs. Sirin et al. [9] used SHOP2, a highly-optimized HTN planner for the task of WSC. The HTN induces a family of compositions and the if-then-else ordering of SHOP2 provided a means of reflecting a preference for achieving a task one way over another. However this limited form of preference was hard-coded into the SHOP2 domain description (i.e., the method description) and could not be customized by an individual user without recoding the HTN. In [10], an HTN-DL formalization was proposed in which they combined reasoning about Web service ontologies using a DL reasoner with HTN planning. Like their predecessor, they exploited SHOP2 domain ordering to reflect preferences, but these were again not easily customizable to an individual user. They further provided a means of preferring services according to their class descriptions, but did not optimize the selection of service groundings.

Most recently, Lin et al. [11] proposed an algorithm for HTN planning with preferences described in the Planning Domain Definition Language PDDL3 [12] that did allow for preferences over service groundings. They implemented a prototype of the algorithm in a planner, **scup**, tailored to the task of WSC. A merit of this work over previous HTN work is that it is not restricted to SHOP2 syntax

and as such provides the nondeterminism (flexibility) necessary for preference-based planning. Unfortunately, the ability of the planner to deal with preferences was somewhat limited. In particular, it appears to be unable to handle conflicting user preferences. The authors indicate that conflicting preferences are removed (rather than resolved) during a pre-processing step prior to run time.

In this paper, we build on our previous work on GologPref, our previous work on HTN planning with rich user preferences, and on the previous work of others on HTN WSC to propose another template-based WSC system, based on HTN planning, **HTNWSC-P**. Our work advances the state of the art by providing an HTN-based WSC system that:

1. synthesizes compositions that adhere to policies and regulations expressed as a subset of linear temporal logic (LTL);
2. exploits a preference language that is truly tailored to WSC with HTNs and that can express preferences over how a task is to be decomposed, as well as preferences over service and data selection;
3. imports and exploits OWL-S profiles for Web service selection;
4. synthesizes a composition that simultaneously optimizes, at run time, the selection of services based on functional and non-functional properties and their groundings, while enforcing stated regulations; and that
5. provides an implementation that combines HTN templates, the optimization of rich user preferences, and adherence to LTL regulations within one system, that reflects and exploits state-of-the-art techniques for planning with preferences. In particular, we exploit our own recent work on HTN planning with preferences [13] as the computational foundation for **HTNWSC-P**.

Elaborating on the first point, many customers must be concerned with enforcement of regulations, perhaps in the form of corporate policies and/or government regulations. Software that is developed for use by a particular corporation or jurisdiction will have the enforcement of such regulations built in. For Web services that are published for use by the masses this is not the case, and the onus is often on the customer to ensure that regulations are enforced when a workflow is constructed from multiple service providers. For inter-jurisdictional or international business, different regulations may apply to different aspects of the composition. In this paper we provide a mechanism for generating compositions from templates that adhere to such regulations.

Figure 1 provides a high-level depiction of our WSC framework. We assume that Web services are described in OWL-S, a leading ontology for describing Web services [14]. We also assume that the composition template (e.g., for trip planning, commodity purchasing, etc.) is described in OWL-S, though it need not be. The user's task (e.g., the specifics of the trip) are specializations of the composition template. We provide a translation from OWL-S to HTN that not only translates OWL-S process models, but also translates service profiles (see Section 2). A user's task, is translated to an initial task network in the HTN framework. User preferences and regulations are also important elements of the structure and could be described within an OWL ontology, though likely not in a

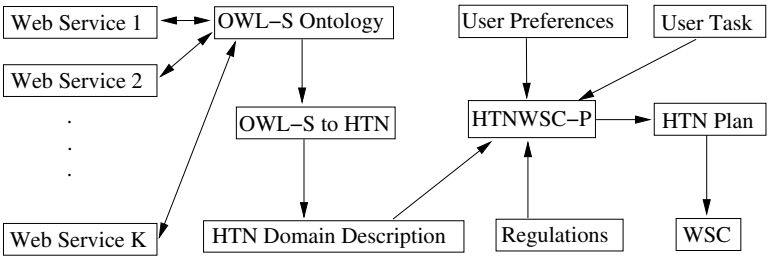

Fig. 1. The overall structure of our composition framework

way that preserves their semantics given their description in LTL. We do not address this issue in the paper. Our HTN WSC planner, **HTNWSC-P**, then takes user preferences specified in PDDL3 syntax (see Section 3) along with a user's task specified as an initial task network and computes a preferred plan, pruning plans that do not meet the imposed regulations specified by LTL formulae. The final HTN plan is then converted to a composition of Web services. In the sections that follow, we elaborate on individual components of our framework.

2 Preliminaries

In this section, we first overview OWL-S [14], a Web ontology for Web services. Next, we describe HTN planning [8] and show how OWL-S can be translated into HTN. The translation is similar to [9], but has some key differences that make service selection based on the non-functional properties possible.

2.1 OWL-S

OWL-S [14] is a Web ontology [15] for Web services with a view to supporting automated discovery, enactment and composition of Web services. The OWL-S ontology has three major components: service profile, process model, service grounding. The service profile is used to advertise the service by describing its functional properties (i.e., input, output, precondition, and effects) and non-functional properties (e.g., service trust, reliability, subject, cost, etc.). The process model describes how the service works, similar to workflow. Finally, service grounding explains how to interact with the service.

OWL-S defines three classes of processes: atomic, composite and simple processes. Each process has input, output, precondition and effects. Atomic processes have no subprocesses and can be executed in a single step. Simple processes provide an abstract view for an existing process. However, unlike atomic processes a simple process is not associated with a grounding. A composite process is composed of other processes via control constructs such as Sequence, Split, Any-Order, Choice, If-Then-Else, Repeat-While, Repeat-Until, and Iterate.

Web service composition systems generally translate OWL-S process models into internal representations such as HTN, PDDL, or Golog that are more amenable to AI planning [7,9,10,11]. Our approach to WSC also translates OWL-S description into an HTN planning problem. In the next section, we briefly describe HTN planning and then describe this translation.

2.2 HTN Planning

Hierarchical Task Network (HTN) planning [8] is a popular and widely used planning paradigm that has been shown to be promising for the task of Web service composition (e.g., [9,11]). In HTN planning, the planner is provided with a set of tasks to be performed, together with a set of so-called *methods* that tell how to decompose tasks into subtasks. Given an initial task network, a plan is formulated by repeatedly decomposing tasks into smaller and smaller subtasks until a primitive decomposition of the initial task network is found. Most of the basic definitions that follow originate in [8].

Definition 1 (HTN Planning Problem). *An HTN planning problem is a 3-tuple* $\mathcal{P} = (s_0, w_0, D)$ *where* s_0 *is the initial state,* w_0 *is a task network called the initial task network, and* D *is the HTN planning domain which consists of a set of operators and methods.*

A domain is a pair $D = (O, M)$ where O is a set of operators and M is a set of methods. An operator is a primitive action, described by a triple o =*(name(o), pre(o), eff(o))*, corresponding to the operator's name, preconditions and effects.

A *task* consists of a task symbol and a list of arguments. A task is primitive if its task symbol is an operator name and its parameters match, otherwise it is *nonprimitive*. A method, m, is a 4-tuple *(name(m), task(m),subtasks(m), constr(m))* corresponding to the method's name, a nonprimitive task and the method's task network, comprising subtasks and constraints. Method m is relevant for a task t if there is a substitution σ such that $\sigma(t)$ =$task(m)$. Several methods can be relevant to a particular nonprimitive task t, leading to different decompositions of t. An operator o may also accomplish a ground primitive task t if their names match.

Definition 2 (Task Network). *A task network is a pair w=(U, C) where U is a set of task nodes and C is a set of constraints. The constraints normally considered are of type precedence constraint, before-constraint, after-constraint or between-constraint.*

Definition 3 (Plan). $\pi = o_1 o_2 \ldots o_k$ *is a plan for HTN planning program* $\mathcal{P}$ *if there is a primitive decomposition,* w*, of* w_0 *of which* π *is an instance.*

2.3 From OWL-S to HTN

In this section, we describe how to translate an OWL-S description into an HTN planning domain and problem. We first describe how to encode an OWL-S

process model as elements of HTN planning (i.e., operators and methods). Then we describe how to encode the service profile. Encoding the service profile as a component of HTN planning will enable users to specify preferences over how to select services based on their non-functional properties (i.e., those specified in the service profile).

Our translation is similar to that in [9]. In particular, we encode each atomic process as an HTN operator just as in [9]. We also encode each composite and simple process as an HTN method. Where our translation differs is that we associate each method with a unique name. Having a name for a method allows preferences to refer to methods by their name. This is particularly important in preferences that describe how to decompose a particular task. Since a task can be realized by more than one method, being able to distinguish each method by its name allows the user to express preferences over which methods they prefer, or in other words, how they prefer the task to be realized. In the next section, we will give examples of such preferences. Below we show how to translate the Sequence construct. The translations for the rest of the constructs is similar.

Translate-Sequence(Q)
Input: a OWL-S definition of a composite process Q in the form Q_1;Q_2;...;Q_k with Sequence control construct.
Output: an HTN method M.
Procedure:
(1) let $\boldsymbol{v}$ = the list of input parameters defined for Q
(2) let Pre = conjunct of all preconditions of Q
(3) **for all** $i : 1 \leq i \leq k$: let n_i be a task node for Q_i
(4) let C= $\{before(n_1, Pre), (n_i, n_{i+1}) | 1 \leq i < k\}$
(5) **Return** $M = (N_m, Q(\boldsymbol{v}), \{n_1, n_2, ..., n_k\}, C)$, where N_m is a unique method name.

In addition, for every process and subprocesses in the process model that is associated with a service (i.e., is executable on the Web), we compile its service profile as extra properties of their corresponding HTN element. Hence, if an atomic/composite process is associated with a service, its corresponding HTN operator/method will be associated with that service profile. We capture this extra property using a predicate *isAssociatedWith*.

For example, let us assume that the Air Canada service can be described by an atomic process AP and service profile SP. In addition, assume that the service profile SP *hasName* AirCanada, *has-url* www.aircanada.com, *has-Language* English, *has-trust* high, *has-reliability* high. Then we will encode the atomic process AP into an HTN operator with the same name as described above. Next, we would capture the service profile of the service Air Canada associated with the atomic process AP by the binary predicate *isAssociatedWith(AP, SP)*. Note AP is the name of the encoded HTN operator. In the case of composite process we would have the name of the corresponding HTN method. The profile information of the service profile SP would now be described by predicates *has-language(SP, English)*, *has-trust(SP, High)*, and *has-reliability(SP, High)*.

3 WSC with Preferences

In this section, we describe the syntax of the preference language we use for specifying user preferences. The preference language is an extension of the Planning Domain Definition Language, PDDL3 [12] that we proposed in [13]. The preference language supports specification of preferences over how tasks are decomposed analogous to how the process model is realized. It also allows users to specify preferences over the non-functional properties of services as well as their preferred parameterizations of tasks analogous to processes. The semantics of the preference language is defined using the situation calculus [4]. We will not discuss the semantics here and direct readers to [13].

Illustrative Example (Travel Example). To help illustrate our preference language, consider the problem of arranging travel for a conference. The problem can be viewed as having a top-level composite process *bookTravelforConference* that is composed of several other composite processes via the Choice construct. One of the composite processes among them can be viewed as a composite process that is constructed via an Any-Order construct into registering for a conference, arranging transportation, accommodations, local transportation, and getting insurance for the trip. Each of these processes can be constructed via the Choice construct to consider several different Web services that offer flights, hotels, cars, trains, buses, insurance, etc.

3.1 Specifying Preferences in Our PDDL3 Extension

The Planning Domain Definition Language (PDDL) is a standard input language for many planning systems. PDDL3 extends PDDL2.2 to support the specification of preferences and hard constraints over *state* properties of a trajectory. In [13], we extended PDDL3 to support preferences that are over how to decompose tasks as well as expressing preferred parameterization of a task (i.e., constraints over *action* properties). In the context of WSC and OWL-S, following the translation from OWL-S to HTN, how to decompose a tasks is analogous on how to realize a service using its process model. This is particularly important when the process model is constructed using the Choice construct and users may prefer one choice over another. Each preference formula is given a name and a metric value (i.e., penalty if the preference formula is not satisfied). The quality of a plan is defined using a *metric function*. A lower metric value indicates higher satisfaction of the preferences, and vice versa.

PDDL3 supports specification of preferences that are temporally extended in a subset of Linear Temporal Logic (LTL). *always*, *sometime*, *at-most-once*, *sometime-after*, *sometime-before* are among the constructs allowed in PDDL3.

We extended PDDL3 to give users the ability to express preferences over how to decompose tasks as well as expressing preferences over the preferred parameterization of a task. We added three new constructs to PDDL3: **occ**(a), **initiate**(x) and **terminate**(x), where a is a primitive task (i.e., an operator or an atomic process), and x is either a task (i.e., a composite process' name and its

input parameters) or a method name (i.e., the unique method name assigned for each method during the translation). **occ**(a) states that the primitive task a occurs in the present state. On the other hand, **initiate**(t) and **terminate**(t) state, respectively, that the task t is initiated or terminated in the current state. Similarly, **initiate**(n) (resp. **terminate**(n)) states that the application of method named n is initiated (resp. terminated) in the current state. Below are some examples from our travel domain given a particular origin Origin and destination Dest[1] that use the above extension.

```
(preference p1 (sometime-after (terminate arrange-trans)(initiate arrange-acc)))
(preference p2 (sometime-after (terminate arrange-acc)(initiate get-insurance)))
(preference p3 (always (not (occ (pay MasterCard)))))
(preference p4 (sometime (initiate (book-flight SA Eco Direct WindowSeat))))
(preference p5
        (imply (different Origin Dest) (sometime (initiate by-flight-trans))))
(preference p6 (imply (and (hasBookedFlight ?Y)(hasAirline ?Y ?X)(member ?X SA))
       (sometime (occ (pay ?Y CIBC)))))
(preference p7 (imply (hasBookedCar ?Z) (sometime (occ (pay ?Z AE)))))
```

p1 states that the task associated with the *arrange-trans* process is terminated before the task associated with the *arrange-acc* process begins (for example: finish arranging your transportation before booking a hotel). Similarly, **p2** states that the task associated with the *arrange-acc* process is terminated before the task associated with the *get-insurance* process begins. The **p3** preference states that the user never pays by Mastercard. Note here that payment with MasterCard is thought of as an atomic process. The **p4** preference states that at some point the user books a direct economy window-seated flight with a Star Alliance (SA) carrier. Here, booking a flight is believed to be a composite process. The **p5** preference states that if Origin and Dest are different, the user prefers that at some point a method named *by-flight-trans* is chosen for decomposition of a task (i.e., the arrange transportation process). The **p6** preference states that if a flight is booked with a Star Alliance (SA) carrier, pay using the user's CIBC credit card. Finally **p7** preference states that if a car is booked, the user prefers to pay with their American Express (AE) credit card.

The **metric function** defines the quality of a plan, generally depending on the preferences that have been achieved by the plan. PDDL3 defines an `is-violated` function, that takes as input a preference name and returns the *number of times* the corresponding preference is violated. It is also possible to define whether we want to maximize or minimize the metric, and how we want to weight its different components. For example, the PDDL3 metric function:

```
(:metric minimize (+ (* 40 (is-violated p1)) (* 20 (is-violated p2))))
```

specifies that it is twice as important to satisfy preference `p1` as to satisfy preference `p2`. Note that it is always possible to transform a metric that requires maximization into one that requires minimization, henceforth, we will assume that the metric is always being *minimized*.

[1] For simplicity, many parameters have been suppressed. Variables start with ?

Further note that inconsistent preferences are handled automatically using the PDDL metric function as discussed above. The metric function is a weighted sum of individual preference formulae. This function is then minimized by our planning approach. In doing so, it makes an appropriate trade off between inconsistent preferences so that it can optimize the metric function.

3.2 Service Selection Preferences

Service selection or discovery is a key component of WSC. However, the only other approaches, to our knowledge, that treat this as a preference optimization task *integrated with* actual composition are [10] and our previous Golog work [7]. In [10], they rely on extending an OWL-S ontology to include abstract processes that refer to service profiles. These descriptions also need to be represented as assertions in an OWL ontology, and an OWL-DL reasoner needs to undertake the task of matching and ranking services based on their service selection preferences. Unfortunately, combining OWL-DL reasoning with planning can create significant performance challenges since one needs to call the OWL-DL reasoner many times during the planning phase, leading to very expensive computations.

Our approach is different. Following discussion in Section 2, during the translation phase we compile each service profile as an extra property of its corresponding HTN element. Note that not all processes will be associated to a service since a process can correspond to an internal subprocess of the service. We only associate profiles with Web-accessible processes. We capture the profile property using a binary predicate *isAssociatedWith(process, service-profile*). The service-profile serves as an index for the profile information and is encoded as additional predicates (e.g., *has-trust(service-profile, trust)*, *has-reliability(service-profile, reliability)*, etc). Below are some service selection preferences for our travel domain.

```
(preference  p8 (always
   (imply (and (initiate ?X)(isAssociatedWith ?X ?Y))(has-trust ?Y High)))
(preference  p9 (sometime
   (and (initiate ?Z)(isAssociatedWith ?Z ?Y)(has-name ?Y AirCanada))))
(preference  p10 (never
   (and (initiate ?Z)(isAssociatedWith ?Z ?Y)(has-reliability ?Y Low))))
```

p8 states that the user prefers selecting services that have high trust values. **p9** states that a user prefers to invoke the AirCanada service. Lastly, **p10** states that the user prefers to never select low reliability services.

4 Regulation-Based Composition

Policies and regulations are an important aspect of semantic Web services. A number of researchers have proposed approaches to both regulation representation and regulation enforcement as part of semantic Web service tasks (e.g., [16]). Kolovski et al. [17] proposed a formal semantics for the WS-policy [18] language by providing a mapping to a Web ontology language OWL [15] and describing how an OWL-DL reasoner could be used to enforce policies. They

provided two translations of WS-Policy to OWL-DL by treating policies as instances and classes in the DL framework. Chun et al. [19] considered policies imposed on both service selection and on the entire composition, expressed using RuleML [20]. In their work, policies take the form of condition-action pairs providing an action-centric approach to policy enforcement.

Regulations are traditionally enforced at design time by verifying that a workflow or composition adheres to the regulations. In our approach, we enforce regulations during composition construction. In particular, during the planning phase we consider only those partial plans that adhere to the regulations while pruning those that do not. In the next section, we provide an algorithm that specifies exactly how this pruning occurs within the HTN algorithm.

In this paper, we focus on regulations that are more geared towards the verification community, particularly those that can be specified as safety constraints. During our regulation enforcement phase, we ensure that the computed composition preserves certain properties of the world. These types of regulations can be specified potentially by state conditions that must hold during the composition. Hence, rather than having action-centric rules in the form of RuleML or rule-based languages, we are interested in assertions that must be enforced during the composition. Classically this form of verification has been represented in Linear Temporal Logic (LTL) [21] or some combination of first-order logic with temporal logic (e.g., [22]). Here, we are not concerned with the representation of regulations within an ontology but rather with how we enforce them within our framework. Hence, for the purpose of this paper we represent regulations in a subset of LTL considering for the most part the *never* and *always* constructs. Below are some example regulations that corporations might impose on their employees when traveling: (1) Always book flights with US-carriers. (2) Never book business or first-class flights. (3) Get pre-approval for travel outside the US. (4) Always pay for flights and hotels with your corporate credit card. As an example, the first regulation above can be written in LTL as follows[2]:

$\Box$ `[((hasBookedFlight ?Y)` $\wedge$ `(hasAirline ?Y ?X))` $\Rightarrow$ `(USCarrier ?X)]`

5 Computing Preferred WSC Adhering to Regulations

In this section we address the problem of how to compute a preferred Web service composition while enforcing regulations. Having the HTN encoding of the problem in hand, we turn to planning techniques to help guide construction of the composition. In particular, we exploit our developed heuristics for HTN planning and augment our algorithm [13] to enforce regulations.

Our algorithm is outlined in Figure 2. Our HTNWSC planner performs best-first, incremental search (i.e., always improves on the quality of the plans returned). It takes as input a planning problem (s_0, w_0, D), a metric function MetricFn, a heuristic function HeuristicFn, and regulations Regulations.

[2] $\Box$ is a symbol for *always*.

```
1: function HTNWSC(s0, w0,D, MetricFn,HeuristicFn, Regulations)
2:   frontier ← ⟨s0, w0, ∅⟩                                  ▷ initialize frontier
3:   bestMetric ← worst case upper bound
4:   while frontier is not empty do
5:     current ← Extract best element from frontier
6:     ⟨s, w, partialP⟩ ← current
7:     if SatisfiesRegulations(s) then                  ▷ pruning to enforce regulations
8:       lbound ← MetricBoundFn(s)
9:       if lbound < bestMetric then                  ▷ pruning suboptimal partial plans
10:        if w = ∅ and current's metric < bestMetric then
11:          Output plan partialP
12:          bestMetric ← MetricFn(s)
13:      succ ← successors of current
14:      frontier ← merge succ into frontier
```

Fig. 2. A sketch of our HTNWSC algorithm

frontier contains the nodes in the search frontier. Each of these nodes is of the form $\langle s, w, partialP \rangle$, where s is a plan state, w is a task network, and $partialP$ is a partial plan. *frontier* is initialized with a single node $\langle s_0, w_0, \emptyset \rangle$, where $\emptyset$ represents the empty plan. Its elements are always sorted according to the function HeuristicFn. *bestMetric* is a variable that stores the metric value of the best plan found so far initialized to a high value representing a worst case upper bound. In each iteration of the while loop, the algorithm extracts the best element from the *frontier* and places it in *current*. If the state violates the regulations (i.e., SatisfiesRegulations(s) returns false), this node will be pruned from the search space. LTL regulations are enforced by progression of the formula as the plan is constructed (e.g., [23]). The LTL formulation is more expressive than HTN-type constraints and thus the enforcement is different that e.g., Redux [24]. Using the function MetricBoundFn a lowerbound estimation of the metric value is computed. If *lbound* is greater than or equal to *bestMetric* this node would again be pruned. If *current* corresponds to a plan, *bestMetric* is updated, and the plan is returned. All successors to *current* are computed using the Partial-order Forward Decomposition procedure (PFD) [8], and merged into the *frontier*. The algorithm terminates when *frontier* is empty.

Although templates specified in HTN greatly reduce the search space, a task can be decomposed by a fairly large number of methods corresponding to a large number of services that can carry out the same task. Hence, we use the heuristics proposed in [13] to guide the search towards finding a high-quality composition quickly. We will use four heuristic functions as follows: Optimistic Metric Function (OM), Pessimistic Metric Function (PM), Lookahead Metric Function (LA), and Depth (D). The OM function estimates optimistically the metric value resulting from the current task network w. Recall that in PDDL3 the metric function defines the quality of a plan. The PM function is the dual of OM. LA function estimate the metric of the *best successor* to the current node. It first solves the current node up to a certain depth, then it computes

a single primitive decomposition for each of the resulting nodes. In the end, it returns the best metric value among all the fully decomposed nodes. D is another heuristic to guide the search. This heuristic encourages the planner to find a decomposition soon. The HeuristicFn function we use in our algorithm is a *prioritized sequence* of the above heuristics. However, as shown in [13] the best combination is to use D, LA, OM, and PM, in that order, when comparing two nodes. Hence, if the depths are equal, we use the other heuristics in sequence to break ties. We will use this prioritized sequence in our evaluations.

The search space for a WSC is reduced by imposing the template, imposing the regulations, and by further sound pruning that results from the incremental search. In particular, the OM function provides sound pruning if the metric function is non-decreasing in the number of satisfied preferences, non-decreasing in plan length, and independent of other state properties. A metric is non-decreasing in plan length if one cannot make a plan better by increasing its length only (without satisfying additional preferences).

Using inadmissible heuristics does not guarantee generation of an optimal plan. However, we have shown in [13] that in the case the search is exhausted, the last plan returned is guaranteed to be optimal. In our algorithm we are pruning those states that violate the regulations, so optimality is with respect to the subset of plans that adhere to the regulations.

Proposition 1. *If the algorithm performs sound pruning, then the last plan returned, if any, is optimal.*

6 Implementation and Evaluation

We implemented our Web service composition engine using templates specified in HTN, user preferences specified in PDDL3 syntax, regulations specified as LTL formulae, and the user's initial task specified as HTN's initial task network. Our implementation, **HTNWSC-P**, builds on our earlier work **HTNPlan-P** [13] that itself is a modification of the LISP version of **SHOP2** [25]. It implements the algorithm and heuristic described above. We used a 15 minute time out and a limit of 1 GB per process in all our experiments.

HTNWSC-P builds on the effective search techniques for **HTNPlan-P**, which was shown to generate better quality plans faster that the leading planners from the IPC-5 planning competition. We do not repeat these experimental results here. However, as such, we had three main objectives in performing our experimental evaluation. We wanted to evaluate the performance of our implementation as we increased the number of preferences and the number of services. We also wanted to compare our work with other WSC preference-based planners that use HTNs. Unfortunately, we were unable to achieve our third objective, since we could not obtain a copy of **scup**[11], the only other HTN preference-based planner (for WSC) we know of (See Section 7 for a qualitative comparison).

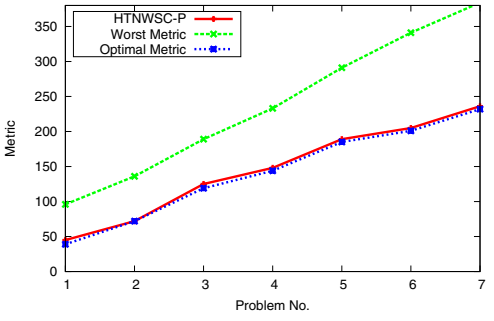

Fig. 3. Evaluating the quality of the last plan as the number of preferences increases. A low metric value means higher quality plan. Worst Metric is a metric value if none of the preferences are satisfied.

Prb #	Ser #	FirstPlan Time(s)	LastPlan Time(s)
1	10	0.22	580.00
2	30	0.23	610.00
3	50	0.21	636.00
4	70	0.22	640.00
5	110	0.23	643.00
6	130	0.24	656.00
7	150	0.24	660.00
8	170	0.26	668.00
9	190	0.24	671.00
10	210	0.25	673.00

Fig. 4. Time comparison between the first and last plan returned as we increase the number of services in the problem

We used the Travel domain described in this paper as our benchmark. (Note that **HTNPlan-P** was additionally evaluated with IPC-5 planning domains.) The problem sets we used were designed to help us achieve our first and second objectives. We achieved this by adding more preferences some of which could potentially be conflicting with each other, and by increasing the number of services, achieved by increasing the branching factor and grounding options of the domain. To this end, we automatically generated 7 problems where the number of services were kept constant and the number of preferences were increased. We similarly generated 10 problems with increasing number of services, keeping the number of preferences constant. The preferences were rich, temporally extended preferences over task groundings and task decompositions. Note that we used a constant number of policies in each problem.

Figure 3 shows the last metric value returned by **HTNWSC-P** for the 7 problems with increasing number of preferences and constant number of services. It also shows the Worst and Optimal Metric value for these problems. Worst Metric is the metric value of the problem if none of the preferences are satisfied while Optimal Metric is the best possible metric value achievable. The result shows that **HTNWSC-P** finds a very close to optimal solution within the time limit. Furthermore, similar to our work in [13], we observe a rapid improvement during the first seconds of search, followed by a marginal one after that.

Next, we evaluated the performance of **HTNWSC-P** by increasing the number of available services. This results in having more methods and operators in the HTN description, hence, the number of possible ways to decompose a single task increases. This causes the number of nodes in the *frontier* to blow up according to the algorithm described in Section 5, and the planner to run out of stack. There are two common ways HTN planners solve this problem. Combining the advantages of both, we propose a middle-ground solution to the problem.

One way to avoid the problem is to have a limit on the size of the *frontier* as in [11]. However, this approach only works if the size is relatively small. Moreover,

many possible decompositions and high-quality solutions could potentially be removed from the search space. Another approach is to use the if-then-else non-determinism semantics taken for example by **SHOP2**. In this semantics, if there are several methods m_1 to m_k that can be used to decompose a task t, method $m1$ should be used if it is applicable, else method m_2, else method m_3, and so forth. Hence, the order in which the methods are written in the domain description can influence the quality of the results. This simple ordering is considered a form of user preferences in [25]. Hence, users must write different versions of a domain description to specify their preferences. However, this form of preferences is very limited and is analogous to writing different templates for different users as opposed to customizing one fixed template to meet users' differing needs.

In this experiment, we employed a combination of the above two approaches, modifying our algorithm to place a limit on the number of applicable methods for a task. Our search considered *all* tasks by considering all of their corresponding nodes in the *frontier* but we limited the number of applicable methods for each task. Note that with this approach we might also potentially prune good-quality plans but the likelihood of this is small compared to limiting the size of the frontier. Nevertheless, our optimality result does not hold for this experiment. Our results are summarized in Figure 4.

Figure 4 shows the time to find the first and the last plan within the time-out. The experiments are run on the 10 problem sets with constant preferences and increasing service numbers. Note that the metric value of all the first and last plans is equal since all 10 problems use the same sets of preferences. The result shows that as the number of services increases, the time to solve the problem increases only slightly.

Finally, recall that our implementation is incremental, performing search in a series, each one returning a better-quality plan. To see how effective this approach is, we calculated the *percent metric improvement* (PMI), i.e., the percent difference between the metric of the first and the last plan returned by our planner (relative to the first plan). The average PMI for the problems used in our experiments is 23%.

7 Summary and Related Work

A number of researchers have advocated using AI planning techniques to address the task of Web service composition including planners that are based on model checking (e.g., [26]) and planners that use a regression-based approach [27]. Previous work has also considered using a template or workflow to ease the task of composition including the work using Golog [5,6,13] and HTNs [9,10,11]. Work by Calvanese, de Giacomo and colleagues on the so-called Roman model is another example of a template-like approach in that they provide a desired behaviour to be synthesized (e.g., [28]) by a set of services. This desired behaviour plays a similar role to that of a template however the synthesis itself is performed using techniques from finite state controller synthesis. Following in this tradition, we also take a template-based approach to WSC. Our templates are specified using HTN domain descriptions and can be customized by

the specification of rich user preferences and by the specification of hard regulations. Users specify their preferences in our PDDL3 extension that supports conditional, temporally extended, service selection preferences as well as preferences over how to parameterize and how to decompose a task. Regulations are specified at LTL formulae. We provide translation from OWL-S to HTN that not only translates OWL-S process models, but also translates service profiles into our HTN framework. Our composition engine, **HTNWSC-P**, then takes user preferences and computes a preferred composition while pruning those that do not meet the imposed regulations. Our algorithm is based on our previous work on HTN preference-based planning that has been demonstrated to outperform leading planners. Experimental evaluation shows that our approach can be scaled as we increase the number of preferences and the number of services.

Most of the related work with respect to specifying and imposing regulations has already been discussed in Section 5. There has also been work on compliance checking using a constraint-based approach that is similar in spirit to regulation enforcement (e.g., [29]). Also, recent work [30] has considered integrity constraints, and proposed various ways to solve the ramification problem. Solving the ramification problem is not a focus of this paper.

The most notable and closest work to ours that uses both HTNs and preferences developed for IPC-5 is [11]. Unfortunately, the **scup** prototype planner is not available for experimental comparison. There are several differences among our works. In particular, they translate user preferences into HTN constraints and preprocess the preferences to check if additional tasks need to be added. They also have an interesting approach to the problem by combining HTN planning with DL, and by using a DL reasoner. However, their preferences are specified in PDDL3, while our preferences can be expressed in the PDDL3 extension that uses HTN-specific preference constructs. Moreover, they do not translate service profiles; hence, they are unable to specify preferences over service selections. Additionally, they do not consider handling regulations, a hallmark of our work. Further, their algorithm cannot handle conflicting user preferences at run-time, and so conflicts need to be detected as a pre-processing step.

Acknowledgements. We thank our colleague Jorge Baier for helpful discussion. We gratefully acknowledge funding from the Natural Sciences and Engineering Research Council of Canada (NSERC) and the Ontario Ministry of Innovations Early Researcher Award (ERA).

References

1. Cheung, W.K.W., Gil, Y.: Privacy enforcement through workflow systems in e-science and beyond. In: Proceedings of the ISWC 2007 Workshop on Privacy Enforcement and Accountability with Semantics (PEAS) (2007)
2. Chun, S.A., Atluri, V., Adam, N.R.: Policy-based Web service composition. In: Proceedings of the 14th International Workshop on Research Issues on Data Engineering: Web Services for E-Commerce and E-Government ApplicationsRIDE, pp. 85–92. IEEE Computer Society, Los Alamitos (2004)

3. Gil, Y., Deelman, E., Blythe, J., Kesselman, C., Tangmunarunkit, H.: Artificial intelligence and grids: Workflow planning and beyond. IEEE Intelligent Systems 19(1), 26–33 (2004)
4. Reiter, R.: Knowledge in Action: Logical Foundations for Specifying and Implementing Dynamical Systems. MIT Press, Cambridge (2001)
5. McIlraith, S., Son, T., Zeng, H.: Semantic Web services. IEEE Intelligent Systems. Special Issue on the Semantic Web 16(2), 46–53 (2001)
6. McIlraith, S., Son, T.: Adapting Golog for composition of semantic Web services. In: Proceedings of the 8th International Conference on Knowledge Representation and Reasoning (KR), pp. 482–493 (2002)
7. Sohrabi, S., Prokoshyna, N., McIlraith, S.A.: Web service composition via generic procedures and customizing user preferences. In: Cruz, I., Decker, S., Allemang, D., Preist, C., Schwabe, D., Mika, P., Uschold, M., Aroyo, L.M. (eds.) ISWC 2006. LNCS, vol. 4273, pp. 597–611. Springer, Heidelberg (2006)
8. Ghallab, M., Nau, D., Traverso, P.: Hierarchical Task Network Planning. In: Automated Planning: Theory and Practice. Morgan Kaufmann, San Francisco (2004)
9. Sirin, E., Parsia, B., Wu, D., Hendler, J., Nau, D.: HTN planning for Web service composition using SHOP2. Journal of Web Semantics 1(4), 377–396 (2005)
10. Sirin, E., Parsia, B., Hendler, J.: Template-based composition of semantic Web services. In: AAAI 2005 Fall Symposium on Agents and the Semantic Web (2005)
11. Lin, N., Kuter, U., Sirin, E.: Web service composition with user preferences. In: Bechhofer, S., Hauswirth, M., Hoffmann, J., Koubarakis, M. (eds.) ESWC 2008. LNCS, vol. 5021, pp. 629–643. Springer, Heidelberg (2008)
12. Gerevini, A., Long, D.: Plan constraints and preferences for PDDL3. Technical Report 2005-08-07, Department of Electronics for Automation, University of Brescia, Brescia, Italy (2005)
13. Sohrabi, S., Baier, J.A., McIlraith, S.A.: HTN planning with preferences. In: Proceedings of the 21st International Joint Conference on Artificial Intelligence(IJCAI), pp. 1790–1797 (2009)
14. Martin, D., Burstein, M., McDermott, D., McIlraith, S., Paolucci, M., Sycara, K., McGuinness, D., Sirin, E., Srinivasan, N.: Bringing semantics to Web services with OWL-S. World Wide Web Journal 10(3), 243–277 (2007)
15. Horrocks, I., Patel-Schneider, P., van Harmelen, F.: From $\mathcal{SHIQ}$ and RDF to OWL: The making of a Web ontology language. Journal of Web Semantics 1(1), 7–26 (2003)
16. Tonti, G., Bradshaw, J.M., Jeffers, R., Montanari, R., Suri, N., Uszok, A.: Semantic Web languages for policy representation and reasoning: A comparison of KAoS, Rei, and Ponder. In: Fensel, D., Sycara, K., Mylopoulos, J. (eds.) ISWC 2003. LNCS, vol. 2870, pp. 419–437. Springer, Heidelberg (2003)
17. Kolovski, V., Parsia, B., Katz, Y., Hendler, J.A.: Representing Web service policies in OWL-DL. In: Gil, Y., Motta, E., Benjamins, V.R., Musen, M.A. (eds.) ISWC 2005. LNCS, vol. 3729, pp. 461–475. Springer, Heidelberg (2005)
18. WS-Policy: Web service policy framework (WS-policy), `http://www.w3.org/Submission/WS-Policy/`
19. Chun, S.A., Atluri, V., Adam, N.R.: Using semantics for policy-based Web service composition. Distrib. Parallel Databases 18(1), 37–64 (2005)
20. RuleML: Rule markup language (RuleML), `http://ruleml.org/`
21. Emerson, E.A.: Temporal and modal logic. In: Handbook of theoretical computer science: formal models and semantics **B**, pp. 995–1072 (1990)

22. Gerth, R., Peled, D., Vardi, M.Y., Wolper, P.: Simple on-the-fly automatic verification of linear temporal logic. In: Proceedings of the 15th International Symposium on Protocol Specification, Testing and Verification (PSTV), pp. 3–18 (1995)
23. Bacchus, F., Kabanza, F.: Using temporal logics to express search control knowledge for planning. AI Magazine 16, 123–191 (2000)
24. Petrie, C.J.: The Redux Server. In: Proc. Intl. Conf. on Intelligent and Cooperative Information Systems (ICICIS), pp. 134–143 (1993)
25. Nau, D.S., Au, T.C., Ilghami, O., Kuter, U., Murdock, J.W., Wu, D., Yaman, F.: SHOP2: An HTN planning system. Journal of Artificial Intelligence Research (JAIR) 20, 379–404 (2003)
26. Traverso, P., Pistore, M.: Automatic composition of semantic Web services into executable processes. In: McIlraith, S.A., Plexousakis, D., van Harmelen, F. (eds.) ISWC 2004. LNCS, vol. 3298, pp. 380–394. Springer, Heidelberg (2004)
27. McDermott, D.V.: Estimated-regression planning for interactions with Web services. In: Proceedings of the 6th International Conference on Artificial Intelligence Planning and Scheduling (AIPS), pp. 204–211 (2002)
28. Calvanese, D., Giacomo, G.D., Lenzerini, M., Mecella, M., Patrizi, F.: Automatic service composition and synthesis: the Roman Model. IEEE Data Eng. Bull. 31(3), 18–22 (2008)
29. Hoffmann, J., Weber, I., Governatori, G.: On compliance checking for clausal constraints in annotated process models. In: Journal Information Systems Frontiers (2009)
30. Hoffmann, J., Bertoli, P., Helmert, M., Pistore, M.: Message-based Web service composition, integrity constraints, and planning under uncertainty: A new connection. Journal of Artificial Intelligence Research (JAIR) 35, 49–117 (2009)