A Comparative Illustration of AI Planning-based Web Services Composition

SEOG-CHAN OH, DONGWON LEE, AND SOUNDAR R. T. KUMARA
Penn State University

As the number of available web services proliferates, finding right web services to fulfill a given goal becomes an important task. In particular, a problem of combining multiple web services to satisfy a single task, known as *web services composition* problem, has received much attention recently, and various solutions have been proposed. Among many proposed solutions, however, it is not clear to use which one in what scenarios. In this paper, to this end, we present: (1) a taxonomy and decision guideline of available solution spaces; (2) an overview of syntactic and semantic matching approaches, and (3) a comparative illustration of three representative solutions from the perspective of e-service workflows.

Categories and Subject Descriptors: I.2.8 [**Artificial Intelligence**]: Problem Solving, Control Methods, and Search – *Plan execution, formation, and generation*; G.1.6 [**Numerical Analysis**]: Optimization - *Integer Programming*
General Terms: Algorithms, Documentation
Additional Key Words and Phrases: Web services composition, Graphplan, SATPlan, Integer Programming

1. INTRODUCTION

Web Services are often considered as one of the most important and vital building blocks for the Semantic Web [Berners-lee et al. 2001]. As such, the industrial support of web services has grown drastically in recent years. For example, it is expected that by 2007, 72% of all application development software will support web services and 45% of all types of software will be web services enabled [Cantera 2004].

Typically, a *client* program first finds a *web services server* that can satisfy certain needs from a yellow page (UDDI), and obtain a detailed specification (WSDL) about the service. Then, using the acquired API, the client sends a *request* to the server via a standard message protocol (SOAP), and in return receives a *response* from the server. Unlike conventional programming interface, web services are self-explanatory. That is, by interpreting XML tags, applications can interpret the operations and data much easier than otherwise.

Author's address: Seog-Chan Oh (sxo160@psu.edu), Industrial and Manufacturing Engineering department, Penn State University, USA; Dongwon Lee(dongwon@psu.edu), School of IST, Penn State University, USA; Soundar R. T. Kumara (skumara@psu.edu), Industrial and Manufacturing Engineering department, Penn State University, USA.
Permission to make digital/hard copy of part of this work for personal or classroom use is granted without fee provided that the copies are not made or distributed for profit or commercial advantage, the copyright notice, the title of the publication, and its date of appear, and notice is given that copying is by permission of the ACM, Inc. To copy otherwise, to republish, to post on servers, or to redistribute to lists, requires prior specific permission and/or a fee.
© 2005 ACM 1073-0516/01/0300-0034 $5.00

The problem that we concern is the first step of this scenario – given a request *r*, finding right web services for *r*. In particular, we are interested in the case where one has to combine multiple web services to satisfy *r* since no single one can. Consider the following motivating example.

Example. Suppose there are two web services available in UDDI as shown in Table 1: (1) `findRestaurant` returns a name, phone number, and address of the closest restaurant provided a zip code and food preference; and (2) `findDirection` returns driving direction and a map image provided a start and destination addresses. "Sylvie" visits "State College, PA" on a business trip and stays in the "Atherton" hotel at "100 Atherton Ave, 16801, PA." Now, she wants to find a Thai restaurant near the hotel along with a driving direction. Let us call this request as r.

Note that neither of two web services can satisfy *r* alone. However, `findRestaurant` can find a Thai restaurant near the hotel, but cannot provide a driving direction. On the other hand, the web service `findDirection` can give a direction from one location to another, but cannot locate a restaurant. Therefore, one has to combine both web services to jointly satisfy *r* as follows: (1) invoke `findRestaurant`("16801", "Thai") to get the address of the closest restaurant, say "410 S. Allen St. 16802, PA"; and (2) invoke the web service `findDirection`("100 Atherton Ave, 16801, PA", "410 S. Allen St. 16802, PA") to get the driving direction. □

Table 1. Example web services

```
<message name='findRestaurant_Request'>
<part name='zip' type='xs:string'>
<part name='foodPref' type='xs:string'>
</message>
<message
name='findRestaurant_Response'>
<part name='name' type='xs:string'>
<part name='phone' type='xs:string'>
<part name='addr' type='xs:string'>
</message>
```

(a) findRestaurant

```
<message name='findDirection_Request'>
<part name='fromAddr' type='xs:string'>
<part name='toAddr' type='xs:string'>
</message>
<message name='findDirection_Response'>
<part name='map' type='xs:string'>
<part                  name='direction'
type='xs:string'>
</message>
```

(b) findDirection

Formally, a web service, w, has typically two sets of parameters: $w_{in} = \{I_1, I_2, ...\}$ for SOAP request (as input) and $w_{out} = \{O_1, O_2, ...\}$ for SOAP response (as output). When w is invoked with all input parameters, w_{in}, it returns the output parameters, w_{out}. We assume that in order to invoke w, all input parameters in w_{in} must be provided (i.e.,

w_{in} are mandatory). In general, given w^1 and w^2, when w^1 can be invoked at the current information state and $w^1_{out} \supseteq w^2_{in}$ then, w^1 can "fully" match w^2.

When one has a request r that has initial input parameters r_{in} and desired output parameters r_{out}, one needs to find a web service w that can fulfill r such that (1) $r_{in} \supseteq w_{in}$, and (2) $r_{out} \subseteq w_{out}$. Finding a web service that can fulfill *r* alone is referred to as *Web Service Discovery (WSD)* problem. When it is impossible for one web service to fully satisfy *r*, one the other hand, one has to compose multiple web services, $\{w^1, w^2, \ldots, w^n\}$ in sequential or parallel way such that (1) for all $w^i \in \{w^1, w^2, \ldots, w^n\}$, w^i_{in} can be grounded when w^i_{out} is required at a particular stage in composition, and (2) $(r_{in} \cup w^1_{out} \cup \ldots \cup w^n_{out}) \supseteq r_{out}$. This problem is often called as *Web Service Composition (WSC)* problem.

Since WSD problem can be trivially solved using a simple Hashtable-like data structure, in this paper, we focus on the WSC problem – *how to efficiently and accurately compose web services to satisfy requests which can not be solved by WSD.*

2. CLASSIFICIATION OF THE WSC PROBLEM

We can classify the WSC problem using the following three facets:

- **Manual vs. Automatic Workflow Composition:** In building workflows by means of web services, one can do either (1) manual composition in cooperation with domain experts; or (2) automatic composition by software programs. In the manual approach, human users who know the domain well (e.g., domain ontology) select proper web services and weave them into a cohesive workflow. Although users may rely on some GUI-based software to facilitate the composition, in essence, it is a labor-intensive and error-prone task, and thus is not appropriate for large-scale WSC problem. On the contrary, in the automatic composition approach, one assume that software programs know if two web services can be connected or not (i.e., via syntactic matching of web services parameters or even via semantic matching).
- **Simple vs. Complex Operator:** The simplest WSC involves only a sequential AND composition – "retrieve data from a web service a_1, AND then from b_5, AND then c_9, AND so on." The more complex WSC, however, can use other operators (e.g., OR,

XOR, NOT) or constraints (e.g., request *r* prefers web services at Asia to ones at Europe) in both sequential and parallel modes.

- **Small vs. Large Scale**: The general WSC problem can be reduced to the satisfiability problem [Vossen et al. 1999] – a well-known *NP-complete* problem. As such, since it is unlikely to find a polynomial algorithm for the WSC problem, exhaustive search algorithms may work for only small-scale WSC problem. For a large-scale problem, therefore, approximate algorithms that find sub-optimal solutions are more desirable.

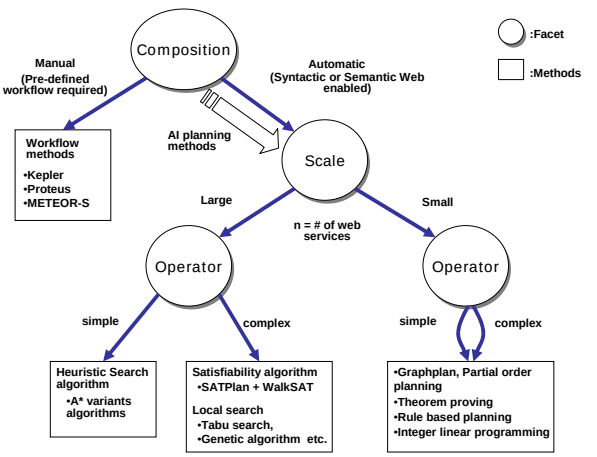


Fig. 1. A decision tree of AI solutions for the WSC problem.

Fig. 1 illustrates a decision tree to help select the right solution using the aforementioned three facets. The manual composition approach can rely on software programs and domain experts to bind manually-generated workflows to the corresponding concrete resources. Therefore, this approach is not appropriate for scenarios where one needs to compose tangible e-services workflow from thousands of available web services. METEOR-S [Sivashanmugam et al. 2003], Proteus [Ghandeharizadeh et al. 2003], and Kepler [Altintas et al. 2004] are examples of this approach.

When the scale or complexity of the WSC problem increases, automatic composition gets more desirable. The automatic composition approach can be complementary to the manual approach such that a few feasible workflows generated from the automatic approach are in turn presented to domain experts who may choose one of them, and refine it further manually. In particular, when complex operators such as negation is not required in the composition, heuristic sub-optimal algorithms such as A* work well [Oh et al. 2005]. On the other hands, when operators are complex and some specific

constraint rules must be checked, rule-based expert systems can work well. However, considering the fast growth of web services, building a full knowledge base by converting all web services into axioms, will be expensive. SWORD [Ponnekanti and Fox 2002] is an example of this approach.

However, for more general WSC problem, often, AI planning based solutions such as STRIPS or Graphplan, or Integer Linear Programming methods work better. For instance, the STRIPS [Fikes and Nilsson 1971] is the first major AI planning system to describe actions in terms of their preconditions and effects. The Graphplan [Blum and Furst 1995] is a general-purpose planner for STRIPS-style domains using graph algorithms. Given a problem statement, Graphplan uses a backward search to extract a plan and allows for partial ordering among actions. As the satisfiability approach for the planning problems, the SATPlan algorithm [Kautz and Selman 1992] is a greedy local search method that translates a planning problem into propositional axioms and finds a model that corresponds to a valid plan. An excellent survey of modern planning algorithms and their application to the WSC problems can be found in [Weld 1999; Rao and Su 2004].

3. OVERVIEW OF MATHCING APPROCHAES

The WSC problem needs to integrate information from heterogeneous sources. Since individual web services are created in isolation, their vocabularies are often with problems having abbreviations, different formats, or typo-graphical errors. Furthermore, two terms with different spellings may mean the same semantic meaning, and thus are inter-changeable (e.g., “price” and “fee”). Conversely, two terms with the same spelling may have different meanings (e.g., “title” may mean either “book title” or “job title”).

In response to these challenges, researchers have developed diverse matching schemes. Consider that x and y are data objects (e.g., web service parameters; individual record field) with a vector of attributes: $x = (x_1, x_2, \ldots, x_k)$, and $y = (y_1, y_2, \ldots, y_k)$, where k is the dimension. We can quantify the “similarity” between x and y by a distance function, $d(x, y)$ with properties: (1) $d(x, y) \geq 0$, where equality holds if and only if $x = y$, (2) $d(x, y) = d(y, x)$: symmetry (3) $d(x, y) \leq d(x, z) + d(z, y)$: triangle inequality.

By using different distance function, $d(x, y)$, one can employ different matching approaches. In general, matching approaches may fall into three categories: (1)

Approach-1: *exact match* using syntactic equivalence; (2) Approach-2: *approximate match* using distance functions; and (3) Approach-3: *semantic match* using ontologies (e.g., RDF and OWL).

In Approach-1, two objects x and y are deemed to be a match if and only if $x = y$. However, with this approach, two objects with slightly different representations (e.g., "William Jefferson Clinton" and "bill Clinton") cannot be matched. For this reason, in Approach-2, if two objects are similar enough according to some distance function (i.e., $d(x, y)$ is above some threshold), then two objects are deemed to be a match. Popular distance functions include TF-IDF, Jaccard, SoftTF-IDF, Jaro, or Levenstein distance [Bilenko et al. 2003]. Although approach-2 is much more flexible than approach-1, it is not still sufficient to identify that "price" and "fee" are inter-changeable. In response, researchers have created the vision of Semantic Web where data has structure and ontologies describe the semantics of the data. Based on Semantic Web foundation, approach-3 can address the ontology-matching problem to find semantic mapping between two ontologies, specified by languages such as OIL, DAML+OIL, OWL, SHOE, and RDF [Doan et al. 2004].

Note that in this paper, the choice of approach to do matching is irrelevant to the WSC problem. We assume that these matching tasks are pre-processed and pre-selected.

4. COMPARATIVE ILLUSTRATION

In this section, we illustrate three selected automatic-composition algorithms for the WSC problem, and discuss the benefits and limitations of them. Among many state-of-the-art proposals, we chose the following three for their impact on other solutions: Glaphplan, SATPlan and Integer Linear Programming (ILP). For instance, most recent STRIPS-style planners (e.g., IPP, STAN, and Blackbox) are originated from both Graphplan and SATPlan. Similarly, ILP with a rich history in operational research community has shown a good performance for AI planning problems [Vossen et al. 1999]. Moreover, ILP naturally allows incorporating various constraints and objectives into planning domain.

Before we proceed to illustrate three approaches, we first cast the WSC problem to the well-studied AI planning problem [Russell and Norvig]. A WSC problem in STRIPS model is represented by $\Pi =< P, W, r_{in}, r_{out} >$ where (1) P is a set of parameters, (2) W is a

set of web services, (3) $r_{in} \subseteq P$ is the initial state, and (4) $r_{out} \subseteq P$ is the set of goal states. In this model, the propositions represent the input and output of the services. The preconditions of a service are to know values of the input parameters, and the effects are to know values of the output parameters.

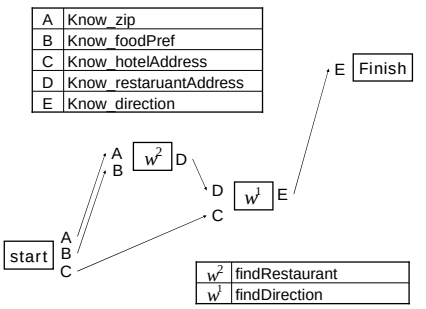


Fig. 2. STRIPS representation.

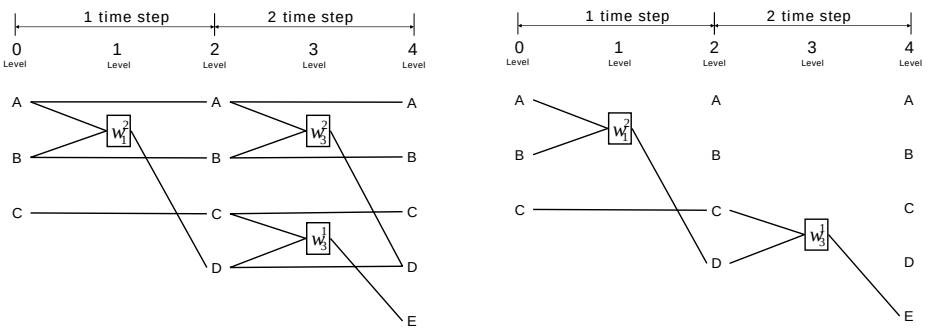
Fig. 2 represents Example 1 in STRIPS-style, where Π is: (1) $P = \{A, B, C, D, E\}$; (2) $A = \{W^1, W^2\}$; (3) $r_{in} = \{A, B, C\}$; (4) $r_{out} = \{E\}$. We can use the STRIPS-style notation for describing the transitions. For instance, `findRestaurant` action has the precondition, "Know_zip AND Know_foodPref" and the effect, "Know_restaurantAddress". "Know_restaurantAddress" is simply a proposition stating that the planner knows a value for "restaurantAddress." In the following, we will illustrate how three methods attempt to solve Example 1 differently.

4.1 Graphplan based planning

Fig. 3 (a) and (b) show a planning graph for Example 1. The graph is expanded to two time steps to find a goal. All axioms in the graph possess situations. For instance, W_1^2 means doing the `findRestaurant` action at the first step. The procedure to expand the graph in Fig. 3(a) to that in Fig. 3(b) is as follows:

- Level 0 starts with the initial state of A, B, and C.
- Level 1 consists of the possible actions that have preconditions satisfied from level 0. Action W_1^2 is possible due to A and B. Note that there are three "maintenance actions" for A, B, and C, respectively, namely "no-op."

- Level 2 consists of the possible effects from the actions in level 1. A, B, and C are possible due to maintenance actions. D becomes possible from action W_1^2.
- Level 3 contains all actions from level 1 and additional actions. Action W_3^1 becomes possible due to the addition of D at level 2.
- Level 4 consists of all possible effects from the actions in level 3. E becomes now possible since action W_3^1 and the goal requirement "Know_direction" are satisfied at this level. Graphplan then proceeds to search backward to find a valid plan as shown in Fig 3(b).



(a) Expanding levels (b) Planning graph solution

Fig. 3. Planning using Graphplan.

4.2 SATPlan based reduction

The Planning Graph in Fig. 3(a) can be converted into a set of logical statements as first proposed by [Kautz and Selman 1992]. First, we can express the initial state at time zero as: $A_0 \wedge B_0 \wedge C_0 \wedge \neg D_0 \wedge \neg E_0$. We also describe the goal states at the highest level as E_4. Then, we can describe the relations between actions and their preconditions as follows:

$W_1^2 \rightarrow A_0 \wedge B_0$, $KeepA_1 \rightarrow A_0$, $KeepB_1 \rightarrow B_0$, $KeepC_1 \rightarrow C_0$, $W_3^2 \rightarrow A_2 \wedge B_2$,

$KeepA_3 \rightarrow A_2$, $KeepB_3 \rightarrow B_2$, $W_3^1 \rightarrow C_2 \wedge D_2$, $KeepC_3 \rightarrow C_2$, $KeepD_3 \rightarrow D_2$

Here, for instance, *KeepAction* corresponds to the maintenance action (i.e., no-op) in Graphplan. In addition, we can express the inference relations between each fact and all actions at the previous level as disjunctions like:

$A_4 \rightarrow KeepA_3$, $B_4 \rightarrow KeepB_3$, $C_4 \rightarrow KeepC_3$, $D_4 \rightarrow W_3^2 \vee KeepD_3$, $E_4 \rightarrow W_3^1$,

$A_2 \rightarrow KeepA_1$, $B_2 \rightarrow KeepB_1$, $C_2 \rightarrow KeepC_1$, $D_2 \rightarrow W_1^2$

Finally, these four logical statements are combined into one conjunction, namely an instance of the satisfiability problem (SAT), and solved by any off-the-shelf tools (e.g., complete methods including Truth Table, Resolution or incomplete methods including WalkSat). The final solution of the problem is:

$$A_0 \wedge B_0 \wedge C_0 \wedge W_1^2 \wedge KeepC_1 \wedge C_2 \wedge D_2 \wedge W_3^1 \wedge E_4$$

4.3 Integer Linear Programming (ILP) formulation

The Planning Graph in Fig. 3(a) can also be formulated as a set of constrains. Suppose that levels 0 and 1 are period 1, levels 2 and 3 are period 2, and level 4 is period 3. Then,

- Variables:

$X_{e,i} = \begin{cases} 1 & \text{if effect } e \text{ is true in period } i \\ 0 & \text{Otherwise} \end{cases}$

$Y_{a,i} = \begin{cases} 1 & \text{if action } a \text{ is carried out in period } i \\ 0 & \text{Otherwise} \end{cases}$

$Y_{e,i}$ = The maintenance action for effect e during period i.

- Initial constraints: $X_{A,1} = X_{B,1} = X_{C,1} = 1$, $X_{D,1} = X_{E,1} = 0$
- Goal constraint: $X_{E,3} = 1$
- Constraints for action preconditions:

$Y_{W^2,1} \le X_{A,1}$, $Y_{W^2,1} \le X_{B,1}$, $Y_{A,1} \le X_{A,1}$, $Y_{B,1} \le X_{B,1}$, $Y_{C,1} \le X_{C,1}$, $Y_{W^2,2} \le X_{A,2}$, $Y_{W^2,2} \le X_{B,2}$,

$Y_{A,2} \le X_{A,2}$, $Y_{B,2} \le X_{B,2}$, $Y_{W^1,2} \le X_{C,2}$, $Y_{W^1,2} \le X_{D,2}$, $Y_{C,2} \le X_{C,2}$, $Y_{D,2} \le X_{D,2}$

- Backward constraints:

$X_{A,3} \le Y_{A,2}$, $X_{B,3} \le Y_{B,2}$, $X_{C,3} \le Y_{C,2}$, $X_{D,3} \le Y_{W^2,2} + Y_{D,2}$, $X_{E,3} \le Y_{W^1,2}$, $X_{A,2} \le Y_{A,1}$, $X_{B,2} \le Y_{B,1}$,

$X_{C,2} \le Y_{C,1}$, $X_{D,2} \le Y_{W^2,1}$

- Objective function:

$$\text{MIN} \sum_{(a,j) \in \text{action set}} Y_{a,j} + \sum_{(e,i) \in \text{maintenance set}} Y_{e,i}$$

The objective function, MIN, is to minimize the number of actions in this programming. In order to get an optimal solution for the ILP model above, one can use any integer linear programming solver (e.g., Excel Solver®, LINDO®, and GAMS® etc.). For instance, an optimal solution is 3 with ($Y_{W^2,1} = 1$, $Y_{W^1,2} = 1, Y_{C,1} = 1$), that is identical to the solutions obtained in Graphplan and SATPlan formulations

4.4 Discussion

Both Graphplan and ILP are approaches suitable for the planning problem with complex operators in a small-scale. On the other hand, SATPlan can be used to find sub-optimal compositions for a large-scale problem with complex operators. Different from Graphplan and SATPlan that address only the shortest time step to reach a goal, in ILP formulation, other QoS (e.g., response time, service cost, or availability of sources) based objective functions can be optimized.

5. CONCLUSIONS

We presented that web services based e-service workflow problem can be formulated as the planning problem of AI and thus can be solved by using the off-the-shelf planning tools. To illustrate their pros and cons, we first introduced three facets that affect the complexity of the problem, and highlighted three representative algorithms within the planning framework.

REFERENCES

ALTINTAS, I. (et al.) 2004. A web service composition and deployment framework for scientific workflows. In *Proceedings of Int'l Conf. On Web Services (ICWS)*.

BERNERS-LEE, T., HENDLER, J., AND LASSILA. O. 2001. *The Semantic Web*. Scientific American.

BILENKO, M., COHEN, W. W., FIENBERG, S., MOONEY R. J., AND RAVIKUMAR P. 2003. Adaptive name matching in information integration. *IEEE Intelligent Systems, 18(5)*.

BLUM A., AND FURST, M.L. 1995. Fast planning through planning graph analysis. In *Proceedings of IJCAI*.

BONET, B., AND GEFFNER, H. 2001. Planning as heuristic search. *Artificial Intelligence*. *129(1-2)*.

CANTERA, M. 2004. *IT professional services forecast and trends for web services*. *ITES-WW-MT-0116*, Gartner Inc.

DOAN, A., MADHAVAN, J., DOMINGOS, P., AND HALEVY, A. 2004. *Handbook on ontologies in information systems*, STAAB, S. AND STUDER, R. (eds.), Springer-Velag.

FIKES, R.E., AND NILSSON, N. 1971. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence, 5(2)*.

GHANDEHARIZADEH, S. (et al.) 2003. Proteus: a system for dynamically composing and intelligently executing web services. In *Proceedings of Int'l Conf. On Web Services (ICWS)*.

KAUTZ, H., AND SELMAN, B. 1992. Planning as satisfiability. In *Proceedings of ECAI*.

OH, S., ON, B., LARSON, E. J. AND LEE, D. 2005. BF*: Web services discovery and composition as graph search problem. In *Proceedings of IEEE EEE*, Hong Kong, China.

PONNEKANTI, S.R., AND FOX, A. 2002. SWORD: A developer toolkit for web service composition. In *Proceedings of WWW*, Honolulu, HI.

RAO, J., AND SU, X. 2004. A survey of automated web service composition methods, In *Proceedings of SWSWPC*.

RUSSELL, S. J., AND NORVIG, P. 2002. *Artificial Intelligence: A modern approach*. Prentice-Hall.

SIVASHANMUGAM, K., VERMA, K., SHETH, A., AND MILLER, J. 2003. Adding semantics to web services standards. In *Proceedings of Int'l Conf. On Web Services (ICWS)*.

VOSSEN, T., BALL, M., LOTEM, A., AND NAU, D. 1999. On the use of integer programming models in AI planning. In *Proceedings of AAAI*.

WELD, D.S. 1999. Recent advances in AI planning. *AI Magazine, 20(2)*.