

# Evaluating Planning based Approaches for End to End Composition and Execution of Web Services

Vikas Agarwal, Girish Chafle, Koustuv Dasgupta, Sumit Mittal, Biplav Srivastava

IBM India Research Laboratory

New Delhi, India

{avikas,cgirish,kdasgupta,sumittal,sbiplav}@in.ibm.com

## Abstract

An end-to-end view of the web service composition process involves the creation of an executable plan (workflow) that realizes the functionality of a new service and its deployment on an execution engine. A number of planning based techniques have been proposed recently for web service composition. However, in order to choose a suitable technique for an application scenario, one needs a formal classification and evaluation of these approaches. To this end, we first introduce a formalization of the Web service composition and execution process. Next, we classify popular service composition techniques found in the literature into four planning based approaches. We distinguish these approaches based on the type of input specifications they take, and the amount of control they give to a user who is supervising the process. Further, we analyse these approaches on the basis of multiple metrics that are relevant in the context of Web service composition. Finally, we present a case study by picking a service composition problem in the telecom domain, reasoning about our choice of approach for solving the problem, and providing an outline of the problem solution.

## Introduction

The literature on composition of both WSDL-described (S. Staab et al. 2003) and Semantic Web services (McIlraith, Son, & Zeng 2001) is quite comprehensive. Much of it deals with resolving discrepancies in the description of Web services, the syntax and semantics of their composition and how they could be executed. Planning is being explored for automatic Web services composition (McDermott 2002; Blythe & others 2003; Srivastava 2002) and many areas apart from classical planning could be relevant: distributed planning (DesJardins *et al.* 1999), planning as model checking (Giunchiglia & Traverso 1999), planning with extended goals and actions (Dal-Lago, Pistore, & Traverso 2002), and HTN planning (Erol, Hendler, & Nau 1994). It has been noted that web services pose challenges to existing planning methods in representation of complex

actions, handling of richly typed messages, dynamic object creation and specification of multi-partner interactions (Srivastava & Koehler 2003).

The problem of Web service composition and execution (WSCE) cannot be seen as a one-shot plan synthesis problem defined with explicit goals but rather as a continual process of manipulating complex workflows, which requires solving synthesis, execution, optimization, and maintenance problems as goals get incrementally refined (Srivastava & Koehler 2004). This is because the WSCE process involves concepts from the AI domain as well as software engineering/programming domain. When viewed as a *program*, input and output parameters become important whereas when viewed as an *action*, the preconditions and effects become dominant (Sabou, Richards, & van Splunter 2003). Both the views are necessary to meet the expectations of real world applications.

The end-to-end view of WSCE involves development of an executable plan/workflow that realize the composite service functionality, its deployment on an execution environment, and its continuous management (monitoring, feedback, and redeployment, if required). This problem can be looked upon as a process with a user (service developer) supervising it. The user can give input to the process and intervene at various stages. The recently proposed approaches to WSCE can then be classified based upon the amount of freedom given to the user to intervene and the kind of input they take. We focus on the following four approaches to solve the WSCE problem.

- All-in-one Composition
- Staged Composition
- Monolithic Composition
- Replanning-based Composition

Different applications have different characteristics (e.g. input, need for the user to intervene, optimization criteria) and different requirements (e.g. scalability, adaptability, failure resolution). In order to choose a suitable WSCE approach for an application one needs a formalization of the problem that brings out the essential characteristics of the problem and allows the

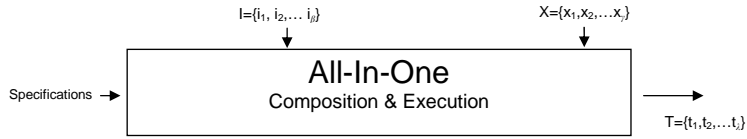


Figure 1: All-In-One Composition and Execution

user to compare decision choices implied in available solution approaches. In this paper, we present a formalization of the end-to-end WSCE problem. The formalization is independent of the specific reasoning that may be done at each stage (e.g., planning, LP, model checking) and makes the underlying assumptions of the corresponding approach explicit. Next, we qualitatively compare the alternative WSCE approaches with respect to multiple metrics relevant to a service composition scenario. We also present a case study for a telecom service provider and demonstrate the use of our framework in reasoning and selection of an appropriate planning-based approach.

## Formalizing End to End Service Composition and Execution

Web service composition and execution is the process of realizing the requirements of a new web service (based on specifications) using the existing component web services. The requirements for web service composition from an end-user can be decomposed into two parts. The first part deals with the desired functionality of the composite service, called the *functional requirements*. The second part involves specification of the *non-functional requirements* that relate to issues like performance and availability. The input to an end-to-end web service composition problem is a composite service specification in a language that is solution dependent, e.g., OWL expression for functional specification, Quality-of-service (QoS) specification. We do not restrict the input to any one representation but assume that it is done in a manner that would make the subsequent composition feasible.

The web services are differentiated into web service *types* which are groupings of similar (in terms of functionality) web services and the actual web service *instances* that can be invoked. The web service types and instances are advertised in a registry. The service instances that can be invoked at a given time are those which are actually deployed (up and running) at that time. In summary:

1.  $C = \{c_1, \dots, c_\alpha\}$ : Set of  $\alpha$  web service types.
2.  $I = \{i_1, \dots, i_\beta\}$ : Set of  $\beta$  service instances advertised in a registry like UDDI. Assuming each service type  $c_i$  has  $M$  instantiations,  $\beta = M \times \alpha$ .
3.  $X = \{x_1, \dots, x_\gamma\}$ : Set of  $\gamma$  services deployed and running at any time. The deployed services are a

subset of the advertised services, i.e.  $X \subseteq I$ , and  $\gamma \leq \beta$ .

The output is a sequential execution trace  $T = \prec t_1, \dots, t_\lambda \succ$  of length  $\lambda$ , where  $t_i$  refers to an invocation of a deployed web service instance.

We are now ready to present four alternative web service composition and execution approaches and characterize them based on the general formalization. The approaches vary depending on how much a user can intervene in the WSCE process and the type of inputs they require.

### The All-in-one Composition and Execution Approach

In the all-in-one approach, the composition and execution processing is internalized as a “blackbox” and the user is not allowed to intervene. The internal process realizes the requirements of the composite service and involves separate or interleaving of the composition and execution stages.

A straightforward solution for web service composition and execution is based on optimizing CSP or mixed integer linear programming, such that the composition semantics are preserved and the composite QoS is met, based on the monitored real-time QoS values. The approach is illustrated in Figure 1. The search space would be at most  $O(\gamma^\lambda)$  since only the deployed web services can be included in any executable composition.

This approach can give an optimal composition based on real-time information and would be ideal if the search space is small. It can capture both composition followed by execution, or interleaving of composition and execution stages. Previous work on metric temporal planning (Do & Kambhampati 2003), planning based on integer optimization (Kautz & Walser 1999) and mixed planning and execution<sup>1</sup> (Johnson *et al.* 2000) are relevant to this approach. The system of (Traverso & Pistore 2004) can be also seen as an implementation of this approach which focuses on generating a sound plan, rather than its optimization.

### The Staged Composition and Execution Approach

In contrast to the all-in-one approach, we can have an approach in which the user can intervene during the composition and the execution stages of the process. Figure 2 gives an illustration of this staged approach.

<sup>1</sup><http://www.cc.gatech.edu/fac/Sven.Koenig/greedyonline/>

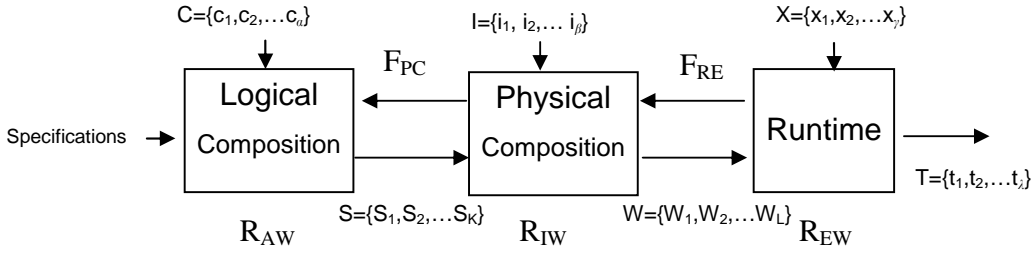


Figure 2: Staged Composition and Execution

This approach distinguishes between service types and instances. The composition first proceeds to generate an abstract plan based on web service types (logical composition), and only the sound abstract plans are concretized into executable plans by selecting the appropriate web service instances (physical composition). Key elements of the approach are:

1.  $S = \{S_1, \dots, S_K\}$ : Set of  $K$  abstract plans selected after Logical composition. An abstract plan  $S_i$  has  $|S_i|$  service types in it and can be found by searching in a  $O(\alpha^{|S_i|})$  space. For  $K$  plans, the worst case complexity is  $O(K \cdot \alpha^\lambda)$ .
2.  $W = \{W_1, \dots, W_L\}$ : Set of  $L$  executable plans selected after Physical composition. The total possible choices for selecting an instantiated plan from  $K$  available abstract plans are  $= \prod_{1..K} M^{|S_i|}$ . The worst case complexity (in terms of search space) for selection of  $L$  instantiated plans is  $O((K \cdot \beta^\lambda)^L)$ .

The output of the logical stage goes to the physical stage, the output of which is passed to the runtime. When multiple plans are passed between stages, one could rank them based on some criteria. Let  $R_{AW}$  denote a ranking function over abstract plans in  $S$  and  $R_{IW}$  denote a ranking function over instantiated plans in  $W$ . At runtime, a similar ranking function  $R_{EW}$  can be defined over the instantiated plans in  $W$ , which takes the deployed service instances into account. The runtime infrastructure can use it to select the composite plan to be executed.

**Function 1**  $R_{AW}(S_i) = r_1(S_i, \Delta_{AW}) \rightarrow \mathfrak{R}$ . *The ranking function is defined over the current plan  $S_i$  and a disruption factor. In semi-automated composition, a user may be inspecting the plan resulting after the logical composition. The selection of given plan would cause a disruption  $\Delta_{AW}$  to the user over previously selected plans. For automated composition,  $\Delta_{AW} = 0$ .*

$R_{AW}$  is used to select among abstract plans. One may extend the definition of  $R_{AW}$  to include a factor about how well an abstract plan meets the given specification, thereby allowing partially sound plans to be passed from one stage to another. We assume that

all plans satisfy the user specifications and hence, the latter does not figure in our definition.

$\Delta_{AW}$  can be decomposed as  $\Delta_c + \Delta_{ui}$ . Here, the first disruption factor accounts for change ( $c$ ) in the comprehension of the plan as perceived by the user. Assume that the user is familiar with a particular plan. If a new plan is now generated, the user has to again try and understand this plan. This effort is inverse to the similarity between the new and the old plan. The second disruption factor accounts for any user intervention ( $ui$ ) on the abstract plan, either by choice or by mandate, before it is given to the physical composition stage.

We can estimate  $\Delta_c$  and  $\Delta_{ui}$  as follows:

- $|S_i|$ : Number of actions in the plan
- $\Delta_c$ :  $|S_{curr} - S_{prev}|$ , which is the cardinality of set difference corresponding to the current and previous plans.
- $\Delta_{ui}$ :  $UIEstimate(S_{curr}) - UIEstimate(S_{prev})$ , where  $UIEstimate(S_i) = \sum_{j \in S_i} (|Action_{j+1}^{IN}| \times |Action_j^{OUT}| - 1)$ .

Note that,  $j$  represents actions at the  $j$ th level of the plan while IN and OUT represent the input and output parameters, respectively.

**Function 2**  $R_{IW}(W_i) = r_2(W_i^{QoS*}) \rightarrow \mathfrak{R}$ . *The ranking function is defined over estimated QoS of the currently instantiated plan.*

$R_{IW}$  is used to select among instantiated plans. Quality of service (QoS) is the most common basis to differentiate among plans and we use it in our definition. Here, QoS\* means that the measures are estimated values rather than the actual runtime values, QoS, that could be monitored at any given time instant in the execution environment. QoS\* is estimated from QoS values aggregated over some time interval and thus, it could lag behind the actual QoS. One may also define the ranking as a function of how well the executable plan meets the given QoS specification, thereby allowing partially compatible plans to be passed from one stage to another. We assume that all plans

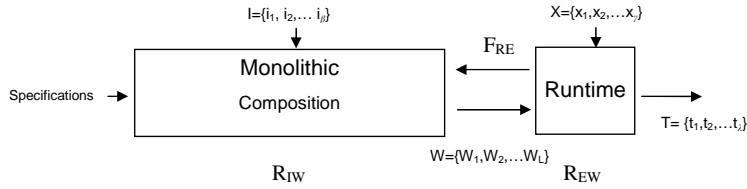


Figure 3: Monolithic Composition and Execution

satisfy the QoS specifications and hence, the latter does not figure as an argument. We have also not introduced a disruption factor in  $R_{IW}$  because executable plans would seldom be modified by users if they can intervene after abstract plans are produced. Using  $R_{IW}$ , we can prefer a plan which is less likely to lead to QoS violations, due to the high margin between its QoS characteristics and the QoS specifications.

**Function 3**  $R_{EW}(W_i) = r_3(W_i, X) \rightarrow \mathfrak{R}$ . *The ranking function is defined over the given instantiated plan and the current set of deployed service instances.*

$R_{EW}$  is a ranking function used to select among instantiated plans. Here, the QoS of  $x_i$  is the monitored runtime value of deployed services rather than the estimated value of all the advertised services. In addition,  $R_{EW}$  can capture other domain-specific preference e.g. execute a plan which leads to fewer deployment of new services.

**Function 4**  $F_{RE}(W) \rightarrow \mathfrak{R}$

The feedback function is given from the runtime to the physical stage and it consists of the monitored QoS of the deployed instances and the performance of the instantiated plans w.r.t. QoS specifications. The feedback will effect the computation of  $R_{IW}$  ranking.

**Function 5**  $F_{PC}(S) \rightarrow \mathfrak{R}$ .

The feedback function is given from the physical to the logical composition and it can contain information that determines the computation of  $R_{AW}$  ranking, e.g. service types with no instance bindings.

A concrete example of this approach, using the ranking and feedback functions, is discussed later in the paper.

### The Monolithic Composition and Execution Approach

In the monolithic approach (Figure 3), the user cannot intervene in the composition phase but can intervene before the workflow is handed off to the execution engine. However, no distinction is made between web service types and instances. The web service instance capabilities can be represented in OWL-S or directly in WSDL with additional annotation about their capabilities, and registries like UDDI store these descriptions. Based on the service specification, all or a subset of services are selected and a planning routine finds the solution to the WSCE problem. The search

space would be at most  $O(\beta^\lambda)$  since potentially all the advertised, and not just the deployed web services, can be used in any generated composition.

The monolithic approach is the most common web service composition approach in literature (Ponnekanti & Fox 2002; McDermott 2002; Srivastava 2002; Sirin, Parsia, & Hendler 2004; Sirin & Parsia 2004). It completely ignores the runtime aspect of web service deployment and assumes that all generated plans can be executed without any problem.

### The Replanning-based Composition and Execution Approach

The replanning-based approach does not plan from scratch. Instead, it takes an initial plan (or plan template) and adapts the plan based on runtime or any other trigger. One way to adapt plans is through a set of *policies* which is a general term used to refer to any specification of behavior. Note that, the use of policies is intrinsic to any decision making problem, and can be used at any stage of the WSCE process. However, we choose to focus on its role in the case of plan adaptation because of the readily available domain-relevant policies that can facilitate such a framework.

Most procedural policy languages (like WS-Policy, REI) support variations of the Event-Condition-Action (ECA) specification. ECA rules specify what actions to take in response to events provided stated conditions hold. An action refers to an activity that can be performed in the domain and a policy may consist of one or more actions. Specifically (Bailey, Poulouvasilis, & Wood 2002):

*On* :  $\prec$  Event  $\succ$   
*If* :  $\prec$  Condition  $\succ$  holds  
*Do* :  $\prec$  Actions  $\succ$

In the replanning-based approach, we have an initial plan which will be executed and a set of policies would guide replanning or plan adaptation if the external events (e.g., runtime failures, changes in advertised instances) occur. The event may take the advertised web service instances into account or be agnostic to it. The plan execution is separate from composition and the approach is illustrated in Figure 4. Systems that want to adapt available plans depending on anticipated contexts (business or operational events) would prefer the replanning-based approach (Chun, Atluri, & Adam 2004).

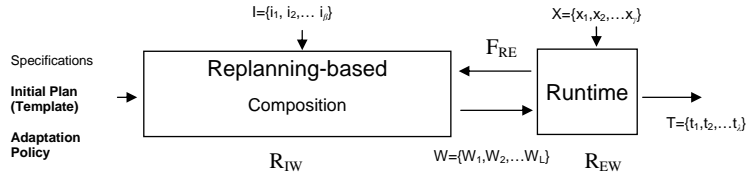


Figure 4: Replanning-based Composition and Execution

The worst case complexity of reasoning with a set of policies is the same as automated reasoning on a knowledge base, and is hence undecidable in nature. (due to the possibility of infinite loops). However, in practice, policies encode domain-relevant information which can be evaluated in a tractable manner.

### Comparing Alternative Approaches for Web Service Composition and Execution

We next evaluate the different approaches for Web service composition and execution described in the previous section. For comparison, we choose a set of metrics that we feel are appropriate in the context of service composition. To the best of our knowledge, this is the first effort to characterize and analyze service composition and execution.

**Completeness:** We define completeness as the ability to compose and execute a (concrete) workflow, if one exists. In other words, given a correct specification the approach is guaranteed to output an execution trace. In this respect, the all-in-one, staged and monolithic approaches are guaranteed to find an executable workflow if one exists. However, in the replanning case, the completeness depends on the set of policies that are defined.

**Composition Effort:** We define composition effort to be a measure of the complexity (time/space) involved in finding an executable workflow using a particular approach. We believe that the search space is a good estimate of the composition effort. Table 1 gives the effort involved with each of the approaches. Note that the figures provided in the table are worst-case estimates. The replanning-based approach has minimum effort (bounded by the number of conditionals) provided the set of policies provided as input can be evaluated in a tractable manner.

**Sensitivity:** The sensitivity of a WSCE approach depends on the number of external inputs that are involved in the process. As shown in Table 1, the sensitivity is highest for the replanning-based approach since a change in one of the policies can arguably radically change the subsequent plans that are produced. Further, we claim that the sensitivities of the monolithic and staged approaches are low and medium, respectively. As is evident, the all-in-one approach is not sensitive to external inputs.

**Composition Quality:** The quality of composition refers to the “goodness” of the plan that is generated by the corresponding approach. We assume that given complete information in terms of the specifications of the service to be composed, and real-time information from the runtime there exists an optimal plan to compose and execute the service. To this end, the all-in-one approach can generate a plan that is optimal. For the staged and monolithic approaches, the quality of the solution depends on the choice of the ranking functions as well as the feedback from the subsequent stages of composition (execution). Similarly, for the replanning-based approach, the quality of the solution is determined by the initial plan and the set of policies defined.

**Control:** This is the amount of control that a user has to intervene and impact the plans that are generated. For example, the user can change one of the ranking functions and effect a change in the plan that is computed by the corresponding approach. In this respect, the “black-box” approach for all-in-one gives no room for the user to control (or intervene with) the composition and execution of a service. The control is low for the monolithic approach, medium in the case of staged, and high for the replanning-based approach. In the case of the latter, the user can impact the subsequent plan by simply changing the initial plan and/or one of the policies.

**Failure Resolution:** The last criterion that we consider is the ability of a particular approach to resolve failures. Note that, such failures might occur during composition (e.g. no abstract plan found, no instance binding available) or during execution (e.g. a deployed service instance fails). In both cases, the approach should be resilient enough to recover from the failure and provide alternate plans to the user. Further, the failure resolution and recovery steps might be possible only by interacting with the user. The “black-box” approach for all-in-one makes it extremely difficult to resolve a failure. The staged approach offers maximum failure resolution because of the inherent nature in which the WSCE process is decomposed into multiple stages. Finally, the replanning-based and monolithic approaches offer low and medium failure resolution, respectively.

### Case Study

Given the intense competition in the telecom sector, service providers need to continually develop compelling

Criteria	All-in-one	Staged	Monolithic	Replanning-based
Completeness	Yes	Yes	Yes	Depends on inputs
Composition Effort	$O(\gamma^\lambda)$	$O(\alpha^\lambda) + O(\beta^\lambda)$	$O(\beta^\lambda)$	$O(1)$
Sensitivity	None	Medium: $\langle R_{AW}, R_{IW}, F_{PC}, F_{RE} \rangle$	Low: $\langle R_{IW}, F_{RE} \rangle$	High: $\langle \text{initial plan, policies}, R_{IW}, F_{RE} \rangle$
Composition Control	None	Medium	Low	High
Composition Quality	Optimal	Depends on $R_{AW}, R_{IW}, F_{PC}, F_{RE}$	Depends on $R_{IW}, F_{RE}$	Depends on policies
Failure Resolution	Minimal	High	Medium	Low

Table 1: Comparison of different Service composition and execution approaches

applications to attract and retain end-users, with quick time-to-market. Often, if a competitor introduces a new service, the service provider must offer a similar or better service within days/weeks, to avoid losing customers. Also, a service provider can attract enterprise customers by offering custom-developed value-added services that leverage its telecom and IT infrastructure. Enterprise customers typically offer significantly higher margins than consumers, and are thus more attractive. Service providers therefore need tools and standards-based runtime platforms to quickly develop and deploy interesting applications for their clients.

### Scenario Description

Suppose a telco wants to enable an enterprise customer to use its telecom and IT infrastructure by creating and deploying services that automate the customer’s business processes. As an example, consider that the telco is attempting to automate a typical Helpline (or call center) for a washing machine manufacturer. A customer calls in to report a problem with her washing machine. This problem needs to be assigned to an agent for resolution. If the problem is such that it could be solved over the phone, a desk-based agent at the call center will be assigned. Otherwise, we need to find an agent in the field who can visit the customer and fix the washing machine. The service provider would like to create a set of web services that automate parts of this process to whatever extent possible, and keep aggregating these components to create higher-level composite services. Once such a software infrastructure is developed, the telco could offer it as a service to various enterprise customers (appliance manufacturers, software vendors, etc), with minor customization. Figure 5 summarizes the workflow in this Helpline scenario.

Here is a sampling of the component services that may be available in the service provider’s infrastructure: Location tracking, SMS, Call Setup, Agent Expertise data, Problem Classification, Agent Selection, Location Based Agent Selector etc. Some of these provide telco-specific functions such as delivering SMS text messages, location tracking of mobile phones, etc. Others are specific to the application domain, e.g. problem classification and are available at enterprise customers’ domain.

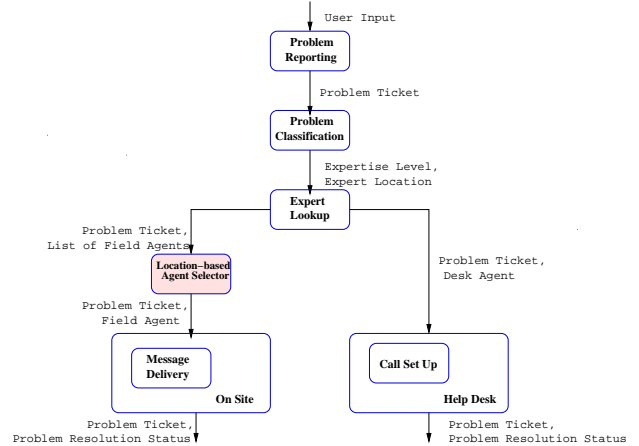


Figure 5: Helpline Service.

### Selecting the Right WSCE Approach

The example scenario that we presented in the preceding subsection is dynamic - new services arrive and depart, and the QoS offerings made by the services change frequently. At any instant, there are hundreds of services providing varying functional capabilities and differing non-functional attributes like response time, cost etc. From the user’s perspective, it might be desirable to verify that the composed service meets its requirements, before it is deployed. We list below several aspects that are desirable of a composition approach for this scenario:

1. Scalability - There are multiple services that provide similar functionality. Moreover, the number of services present in the environment might be quite large. The system should be able to scale with the number of types and the number of service instances.
2. Adaptability - Plans generated should be responsive to the changes in the run-time environment. For example, it should be easy to account for new services as well as to account for departure of existing services. Moreover, any significant changes in the QoS offerings of the component services should impact the creation and execution of the composite service.

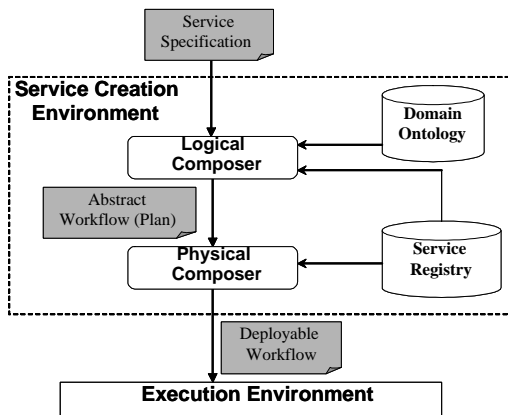


Figure 6: Solution Overview.

3. Failure Resolution - We are dealing with a dynamic environment where faults can occur frequently. Hence, the service creation environment should have a mechanism to detect, resolve and recover from faults at various stages of composition and execution.
4. User Interaction - The user would want the flexibility to supervise the entire composition procedure, starting from generation of abstract plan, to the creation of an executable workflow, and finally its deployment on a run-time infrastructure.

Keeping in mind the above requirements, we choose the staged approach for service composition and execution. The rationale behind our choice is that the decomposition of the WSCE process inherently improves the scalability of the solution as more service types and instances are added to the registry. Further, the ranking functions at different stages along with feedback information, enables the ability to adapt to changes in runtime conditions and better failure resolution. Finally, the staged approach gives more control to the user to intervene and fine-tune both the abstract plan and the deployable workflow.

We now look at our solution based on the staged WSCE approach (Agarwal *et al.* 2005).

## Solution Description

Our solution discovers the relevant services from amongst the available ones, creates the control flow between them, and stitches them together into an executable BPEL workflow. The available services are *semantically* annotated, providing meta-information about their functionality in the context of a domain model. The developer needs to formally specify the functional and non-functional requirements of the service to be created. The tool can then generate a flow, and with some developer inputs, deploy the flow on to a runtime infrastructure.

The basic approach to automating the process of service creation is illustrated in Fig. 6. A **Service Registry** contains information about services

available in-house as well as with participating 3rd-party providers. The *capabilities* of each available service *type* are described formally, using domain-specific terminology that is defined in a **Domain Ontology**. When a new service needs to be created, the developer provides a **Service Specification**. Driven by the specified functional requirements, the **Logical Composer** uses generative planning-based automated reasoning techniques to create a composition of the available service types that meets the specified requirements. The **Physical Composer** next selects the best web service *instances* to produce an executable workflow that meets the non-functional (QoS) requirements. Our **Execution Environment** orchestrates the workflow in a *decentralized* fashion (Chafle *et al.* 2004; Nanda & Karnik 2003), with partitions of the flow executing concurrently in network-proximity with the component services they invoke. This results in better scalability and performance.

The user can interact with the tool at each stage of the composition, providing validation and fine-tuning to the workflow being constructed. In our current implementation, we have one plan going from the logical stage upto the execution stage. The system follows the staged approach (in Figure 2) with  $K = |S| = 1$ ,  $L = |W| = 1$ ,  $R_{AW}(S_i) = r_1(S_i, 0)$ , and  $R_{IW}(W_i) = r_2(W_i^{QoS*})$ .  $R_{EW}$  is not used by the current execution engine. Note that, we propose to enhance the WSCE process by using feedback from each stage and incorporate the ability to pass a ranked set of plans between each stage of composition.

## Conclusion

A number of planning-based approaches have been proposed recently for Web service composition and execution. In this paper, we have classified these approaches based on kind of input they take and the extent of freedom given to the user to intervene with the process. Further, we have presented a comparative analysis of the approaches, and demonstrated its usefulness in selecting an efficient service composition approach for a telecom domain-specific problem. To the best of our knowledge, no such framework exists for evaluating end-to-end service composition techniques and the current work would fill a crucial void.

## References

- Agarwal, V.; Dasgupta, K.; Karnik, N.; Kumar, A.; Kundu, A.; Mittal, S.; and Srivastava, B. 2005. A service creation environment based on end to end composition of web services. In *Proceedings of the 14th International World Wide Web Conference*.
- Bailey, J.; Poulouvasilis, A.; and Wood, P. 2002. An event-condition-action language for xml. In *Proceedings of the 11th International World Wide Web Conference*.
- Blythe, J., et al. 2003. The Role of Planning in Grid

- Computing. Proceedings of International Conference on AI Planning and Scheduling.
- Chaffe, G. B.; Chandra, S.; Mann, V.; and Nanda, M. G. 2004. Decentralized Orchestration of Composite Web Services. In *Proceedings of the 13th International World Wide Web conference*.
- Chun, S. A.; Atluri, V.; and Adam, N. R. 2004. Policy-based web service composition. In *Research Issues on Data Engineering: Web Services for e-Commerce and e-Government Applications (RIDE)*.
- Dal-Lago, U.; Pistore, M.; and Traverso, P. 2002. Planning with a language for extended goals. In *Proceedings of AAAI*, 447–454.
- DesJardins, M.; Durfee, E.; Ortiz, C.; and Wolverton, M. 1999. A survey of research in distributed, continual planning. *AI Magazine* 20(4):13–22.
- Do, M. B., and Kambhampati, S. 2003. Sapa: A Scalable Multi-objective Heuristic Metric Temporal Planner. *Journal of AI Research* 20:155–194.
- Erol, K.; Hendler, J.; and Nau, D. S. 1994. HTN planning: Complexity and expressivity. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*.
- Giunchiglia, F., and Traverso, P. 1999. Planning as model checking. In *Proceedings European Conference on Planning*.
- Johnson, A.; Morris, P.; Muscettola, N.; and Rajan, K. 2000. Planning in interplanetary space: Theory and practice. In *Proceedings AIPS*.
- Kautz, H., and Walser, J. P. 1999. State-space planning by integer optimization. In *Proceedings Fifteenth National Conference on Artificial Intelligence (AAAI-99)*.
- McDermott, D. 2002. Estimated-Regression Planning for Interactions with Web Services. In *Proc. AIPS*.
- McIlraith, S.; Son, T.; and Zeng, H. 2001. Semantic web services. In *IEEE Intelligent Systems (Special Issue on the Semantic Web)*, March/April 2001.
- Nanda, M. G., and Karnik, N. 2003. Synchronization Analysis for Decentralizing Composite Web Services. In *Proceedings of the ACM Symposium on Applied Computing*.
- Ponnekanti, S., and Fox, A. 2002. SWORD: A Developer Toolkit for Web Service Composition. In *Proceedings of the 11th International World Wide Web Conference*.
- S. Staab et al. 2003. Web services: Been there, done that? *IEEE Intelligent Systems* 72–85.
- Sabou, M.; Richards, D.; and van Splunter, S. 2003. An experience report on using DAML-S. In *Proc. of 12th WWW Conf.*
- Sirin, E., and Parsia, B. 2004. Planning for Semantic Web Services. In *Semantic Web Services Workshop at 3rd International Semantic Web Conference*.
- Sirin, E.; Parsia, B.; and Hendler, J. 2004. Composition-driven Filtering and Selection of Semantic Web Services. In *AAAI Spr. Symposium on Semantic Web Services*.
- Srivastava, B., and Koehler, J. 2003. Web Service Composition - Current Solutions and Open Problems. ICAPS 2003 Workshop on Planning for Web Services.
- Srivastava, B., and Koehler, J. 2004. Planning with Workflows - An Emerging Paradigm for Web Service Composition. ICAPS 2004 Workshop on Planning and Scheduling for Web and Grid Services.
- Srivastava, B. 2002. Automatic web services composition using planning. In *Proceedings of KBCS, Mumbai, 2002.*, 467–477.
- Traverso, P., and Pistore, M. 2004. Automated Composition of Semantic Web Services into Executable Processes. In *3rd International Semantic Web Conference*.