RI07005 13 August 2007 Computer Science

IBM Research Report

Understanding Approaches for Web Service Composition and Execution

Vikas Agarwal, Girish Chafle, Sumit Mittal and Biplav Srivastava
IBM India Research Laboratory
4, Block - C, Institutional Area
Vasant Kunj, New Delhi - 110 070, India.

IBM Research Division
Almaden - Austin - Beijing - Delhi - Haifa - T.J. Watson - Tokyo - Zurich

Understanding Approaches for Web Service Composition and Execution

Vikas Agarwal, Girish Chafle, Sumit Mittal, Biplav Srivastava

IBM India Research Laboratory
New Delhi, India
{avikas,cgirish,sumittal,sbiplav}@in.ibm.com

Abstract. Web services have received much interest due to their potential in facilitating seamless business–to–business or enterprise application integration. Of particular interest is the Web Service Composition and Execution (WSCE) process - the creation of a workflow that realizes the functionality of a new service and its subsequent deployment and execution on a runtime environment. A significant number of solutions have been proposed in the literature for composition and execution of web services. However, in order to choose a suitable technique for an application scenario, one needs to systematically analyze the strengths and weaknesses of each of these solutions. To this end, we present an analysis that includes formalization of the WSCE process, a classification of existing solutions into four distinct categories (approaches), and an in–depth evaluation of these approaches. Our evaluation is based on multiple metrics that we deem critical for a WSCE system, e.g. composition effort, composition control, and ability to handle failures. We also present an application of this analysis to three different scenarios.

1 Introduction

Web services allow interoperability between software components by providing standards–based mechanisms for description, discovery and messaging/integration. This has led to the emergence of web services as a standard mechanism for accessing information and software components programatically. Composition of web services enables businesses to interact with each other and facilitates seamless business–to–business or enterprise application integration [18].

The process of web service composition and execution (WSCE) consists of creation of a workflow that realizes the functionality of a new service followed by its deployment and execution on a runtime infrastructure. A number of approaches have been proposed in literature for WSCE. In AI, the planning discipline looks at how plans can be automatically generated given a set of actions, the initial state and the desired goal state. Web service composition, in this respect, can be seen as a planning problem where each service corresponds to actions, and the initial and goal states are suitably defined to correspond to the requirements of the new service. Unsurprisingly, a large portion of Planning has been explored for automatic web service composition [13, 3, 17] and many

areas apart from classical planning are relevant, like distributed planning [6], planning as model checking [10], planning with extended goals and actions [5], and HTN planning [8], metric temporal planning[7], planning based on integer optimization[12], mixed planning and execution[11], etc.

Two aspect of the WSCE domain set it apart from typical planning domains. One is the open-world setting. Most planning formalisms assume the closed-world assumption wherein the environment is assumed to be unchanged from the time a plan is generated to the time the plan is executed, thereby disallowing any unknown events. When such planners are applied to the open-world WSCE, plans may fail to execute. Therefore, planning techniques which work under uncertainty but can provide some execution guarantee are very useful. Second, messages and operations in the WSCE problem refer to real world entities that are expressed using rich data models (e.g., in ontologies). The models are background theories which should be consistent and integrated into WSCE reasoning. In a typical planning setting like PDDL, the background theory is minimally expressed (e.g., as a type hierarchy).

Apart from planning-based approaches, template–based web service composition has been explored in [16, 21] for customizing an abstract workflow for specific requirements. In general, a large number of WSCE solutions have been reported in literature, for example in [14, 17, 15, 4]. Recently, we have proposed a staged composition approach [1] which combines the strengths of business web services (using WSDL, BPEL, SOAP etc.) and semantic web services(using ontologies, goal directed reasoning etc).

While applying the WSCE process to various application domains, one finds that different applications have different characteristics (e.g. type of inputs, need for the user to intervene, optimization criteria) and different requirements (e.g. scalability, adaptability, failure resilience). Given the plethora of solutions to realize the WSCE process, one needs a systematic way to analyze the strengths and weaknesses of each solution to select a suitable technique for a particular application. Not much work has been done in this direction and we feel that a fresh prespective of the entire WSCE process needs to be taken. To this end, we present an understanding of the different solutions for WSCE. This includes:

- A formalization of the WSCE process
- Classification of existing solutions into four different approaches
- An in–depth evalution of these approaches

We first present a formalization of the WSCE process. The formalization is required to bring out the essential characteristics of the problem so as to allow the user to compare the decision choices implied in each approach. Next, we classify the existing solutions into four categories (approaches) based on the kind of inputs they take and the amount of freedom given to the developer to intervene in the WSCE process. Thereafter, we evaluate the approaches based on multiple metrics which we deem are critical in selecting a suitable WSCE approach, e.g. composition effort, ability to handle failures. Finally, we study the suitability of the different WSCE approaches to three different application scenarios, viz, an application integration case study from telecom domain, a travel reservation

scenario using online services and a data integration application in the case of real estate.

2 Formalizing End to End Service Composition and Execution

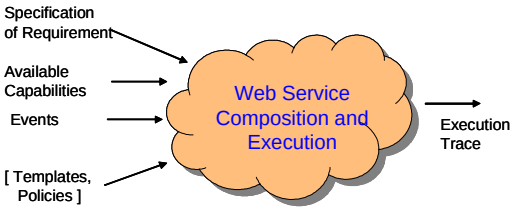

Fig. 1. Web Service Composition and Execution (WSCE)

Web service composition and execution is the process of realizing the requirements of a new web service using the capability specifications of 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 description of the *non-functional requirements* that relate to issues like performance and availability. These inputs can be in different languages, e.g., OWL expression for functional specification and Quality-of-service (QoS) expression for specifying non-functional parameters. Web service capabilities also can be represented in many ways: OWL-S, WSDL descriptions (with possibly additional semantic annotations), etc. These capabilities can be advertised using registries like UDDI. 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.

Web services can be further 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 different web service types and instances can be advertised in a single registry, or in different registries.

Formally, we denote,

1. C = $\{c_1, ..., c_\alpha\}$: Set of α web service types.
2. I = $\{i_1, ..,i_\beta\}$: Set of β service instances advertised in a registry like UDDI. Assuming each service type c_i has M instantiations, $\beta = M \times \alpha$.
3. X = $\{x_1, ..., x_\gamma\}$: Set of γ 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 $T = \prec t_1, ...,t_\lambda \succ$ of length λ, where t_i refers to an invocation of a deployed web service instance.

In the rest of this section, we present our classification of existing solutions into four different approaches. We start with the bases for this classification.

2.1 Bases for Web Services Composition and Execution

Here are some bases to compare WSCE approaches:

1. Are composition and execution phases separable? The choices are **No** or **Yes**.
2. When does composition happen? The choices are **Offline** at design time or **Online** during runtime.
3. How does composition happen? The choices are **Search-based** like what happens in planning or **Template-based** where the search-space is constrained by the templates.
4. What information is used for composition? The choices can be information about a subset from **Service types**, **Service instances published**, **Services deployed**, **Templates/ Policies**.
5. How are external events handled at runtime (adaptation)? The choices are **On-the-fly** where events are handled as they occur or **Gradual** where events are handled after a time lag.

We are now ready to present existing web service composition and execution approaches as four alternatives and characterize them on the above bases. The approaches result in different characteristics of the WSCE process in terms of how much a user can intervene in the WSCE process, the type of inputs required, etc.

2.2 Interleaved Composition and Execution

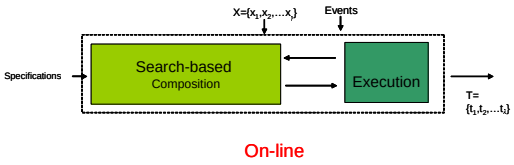

Fig. 2. Interleaved Composition and Execution

Recall that during composition of web services, a complete workflow (plan) is generated describing how to call the primitive web services to get the overall behavior. During execution, these web services are actually invoked to get the desired results. In the interleaved approach, the composition and execution

processes cannot be practically separated as distinct phases. The reason is that the domain and problem has sizable uncertainties which can be easily resolved for a problem by dynamically querying the environment rather that accounting for all possibilities in a single robust plan which can handle any runtime eventuality.

As an example, in order to complete the trip reservation of a frequent traveller from Place X to Place Y, one has to book flight, train or bus (from X to Y and back), book hotel or friend's place (at Y knowing when one arrives) and taxi bookings (at both X and Y depending on the person's location and time in X and Y). One may decide that it is difficult to produce a robust workflow that meets all prospective travel needs (which is what a separable approach would require). An interleaved approach, on the other hand, will dynamically query for the locations(X,Y), time of the year, friends convenience, etc. to arrive at a suitable reservation.

In interleaved approach the composition and execution phases are internalized in a WSCE *blackbox*, and there are no provisions for the service developer to intervene. The internal process realizes the functional as well as non–functional requirements of the composite service. The approach is illustrated in Fig. 2. The search space for interleaved is 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 from the execution environment (e.g. QoS, deployed services) and would be ideal if the search space is small. However, the domain should have the characteristic that there are no pathalogical states which the WSCE agent has landed but can now not recover. If that happens, the interleaved approach in such a domain would become incomplete as a solution (i.e. solution exists, but cannot be reached). For example, after a flight has been booked, one should not find that there are no hotels available for the arrival time of this flight.

In the literature, ConGolog and Heracles+Theseus[9] are based on the interleaved approach.

2.3 Monolithic Composition and Execution

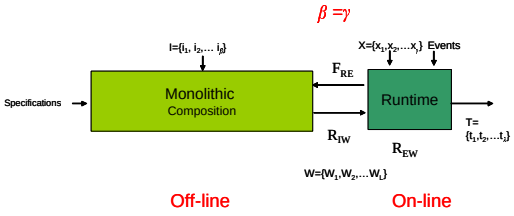

Fig. 3. Monolithic Composition and Execution

In the monolithic approach (Fig. 3), the user cannot intervene in the composition phase but can inspect and (possibly) modify the workflow before it is passed to the execution environment. The composition process ignores the deployment and runtime aspect of web services and only focuses on their advertised descriptions. It assumes that all generated plans can be executed without any problem. If this assumption gets violated, feedback functions would be needed to inform the composer so that it can generate plans with better chances of getting successfully executed.

Based on the service specification, all or a subset of services are selected and a planning routine finds a composite executable workflow. The search space would be at most $O(\beta^{\lambda})$ since potentially all advertised instances can be used in a composition. Further, multiple workflows can be passed to the execution engine. This gives the WSCE system the ability to pick and choose among them (based on some optimization criteria) and also provide better failure resiliency, in case a particular workflow fails. This implies that we need to rank the workflows based on some criteria.

Functions R_{IW} and R_{EW} are used to rank a set of executable workflows with the only difference that R_{IW} is used to decide workflows that are passed from the composition to runtime stage while R_{EW} is used over alternative workflows at the runtime stage. Quality of Service (QoS) is the most common basis to differentiate among workflows. Note that there are different QoS metrics that could be of possible interest for end to end composition (e.g. cost, availability, response time). In this context, the execution environment should have a monitoring infrastructure in place that records the metrics of interest and periodically sends these values to the composition phase in the form of feedback $F_E(W)$.

The monolothic approach is the most common web service composition approach in literature [14, 15, 17]. The system of [20] can be also seen as an implementation of this approach where the input specifications consists of message exchanges between the component services and the output is an executable (BPEL) workflow that implements the specifications.

2.4 Staged Composition and Execution

In the staged approach (Fig. 4), the service descriptions are distinguished between service types and instances. Consequently, the composition first proceeds to generate an abstract worklow based on web service types (which we call *logical composition*). This is subsequently concretized into an executable workflow by selecting the appropriate web service instances (which we call *physical composition*). The two key elements of the approach are:

1. S = $\{S_1, ..., S_K\}$: A set of K abstract workflows selected after the logical composition stage.
2. W = $\{W_1, ..., W_L\}$: A set of L executable workflows selected after the physical composition stage.

Given that each service type has atmost $O(M)$ instantiations, the worst case complexity for the staged approach is $O(\alpha^{\lambda}) + O(M^{\lambda})$. Once again, multiple

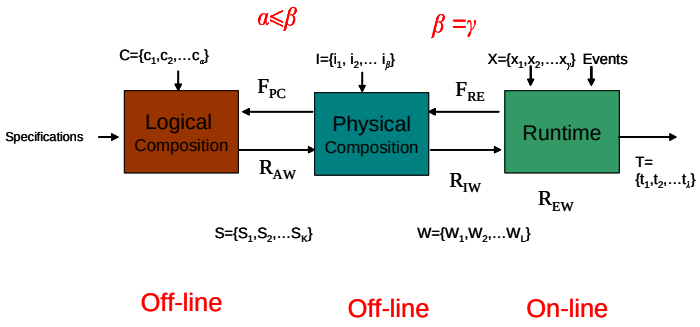

Fig. 4. Staged Composition and Execution

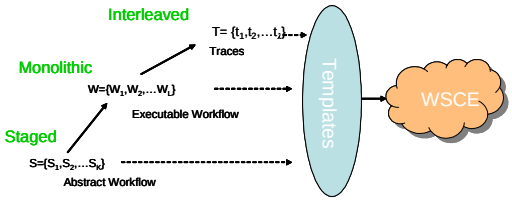

Fig. 5. Template–based Composition and Execution

workflows can be passed between the stages, thereby implying the need for ability to rank the workflows at each stage. With this in mind, we introduce a third ranking function R_{AW} (in addition to R_{EW} and R_{IW} defined earlier) to define a ranking over the set of abstract workflows.

R_{AW} is used to rank a set of abstract plans. The ranking function incorporates metrics that are characteristics of the plan that is generated. For example, $\mid S_i \mid$ - the number of steps in the workflow, could serve as a ranking function. One could also extend the ranking function R_{AW} to account for the developer's effort required to match the input-output parameters between each step of the workflow, before it is handed off to the physical composition stage.

2.5 Template–based Composition and Execution

The template–based approach does not plan from scratch like other approaches. Instead, it takes outlines of permissible workflows (or templates) and instantiates one of them to obtain an executable workflow. The selection and instantiation of the template can be based on a set of *policies* covering goals, events and runtime situations. Policies are specifications of behavior; the most popular method of describing these policies is through Event-Condition-Action (ECA) formulations.

Consider Figure 5. The initial template can either be hand-coded or can be extracted by generalizing the workflows/ traces of any of the previous three approaches. By generalization, we mean removal of any commitment from a previously successful workflow/ trace so that it can be re-applied to a new situation.

Criteria	**Interleaved**	**Monolithic**	**Staged**	**Template–based**
Separable?	No	Yes	Yes	Yes
When	Online	Offline	Offline	Online/ Offline
How	Search	Search	Search	Template
What Information	Deployed Instances	Advertised Instances	Advertised Type & Instances	Templates/ policies Advertised Types & Instances
Change Rate	On-the-fly	Gradual	Gradual	On-the-fly/ Gradual

Table 1. Summarizing WSCE approaches along different bases

For example, in the case of interleaved approach, we can take one successful trace of execution and substitute the actual service instances that were invoked with their service types. Similiarly, for monolithic or staged approaches, we can take successfully executed workflows and substitute actual service instances by their types in the workflow.

In a new composition problem, based on the available templates and the prevalent situations, a set of policies would guide which template gets selected and instantiated to arrive at the executable workflow. The policies can also guide plan adaptation if the external events (e.g., runtime failures, changes in advertised instances) occur while a composition is being executed. The adaptation policy may take the advertised web service instances into account or only use the monitored data. The template-based approach works best for applications where compositions follow regular patterns and its adaptation is required to be along anticipated contexts (business or operational events)[4].

The complexity of template-based approach depends on its generalization. Given a template of length λ where each service type has atmost $O(M)$ service instances, the worst case complexity for a template-based approach which selects instances to produce executable workflows is O(M^{λ}).

2.6 Summarizing Web Service Composition Approaches

We now summarize the alternative approaches along the different bases for comparison on Table 1. The Interleaved approach does not separate the composition and execution phases, while the others do; hence its compose phase is online, while the other approaches can compose offline. The Interleaved approach works with information of deployed instances while the monolithic approach works with information of advertised instances. Staged needs advertised type and instance information while template approach additionally needs templates and the policies to use these templates. The rate at which a change can be incorporated in each approach reflects the inter-relationship between the composition and execution phases.

Criteria	Interleaved	Monolithic	Staged	Template
Composition Effort	O(γ^{λ})	O(β^{λ}) Min:O(γ^{λ})	O(α^{λ}) + O(M^{λ})	O(M^{λ})
Composition Control	None	Low: $< R_{IW}, F_E >$	High: $< R_{AW}, R_{IW}, F_C, F_E >$	High: <template, Underlying $CompositionMethod >$
Handling Composition Failure	None	Medium	High	Low
Adaptation during Execution	High	Medium	Medium	Low to Medium
Information Modeling	Simple (Instances)	Simple (Instances)	Elaborate (Types and Instances)	Elaborate (Templates and Instances)

Table 2. Comparison of different Service Composition and Execution approaches

3 Comparing Alternative Approaches for Web Service Composition and Execution

In this section, we analyze and compare the different approaches for Web service composition and execution. For this purpose, we choose a set of metrics that we feel are appropriate in this context. Table 1 summarizes our analysis for each approach.

Composition Effort: We believe that search space is a good estimate of the composition effort involved in finding an executable workflow. Table 1 gives the effort involved with each of the approaches. Note that the figures provided in this table are worst–case estimates. The interleaved approach considers only the deployed instances during the process of composition and execution. While a monolithic architecture has to work with all web services advertised, a template–based solution only needs to find service instances matching the tasks in the template. A staged approach, on the other hand, proceeds in two stages. It first looks at the available service types for realizing the functionality of the required composite service before selecting instances for each of the chosen types. A clean separation of services into types and instances reduces the search space, providing scalability to the composition process.

Composition Control: This is the amount of flexibility that a developer has to intervene and fine-tune the workflows that are being generated. For example, the developer can affect a workflow by changing one of the ranking functions. In this respect, the "black-box" approach for interleaved gives no room for the user to control (or intervene in) the composition and execution of a service. The control is low for monolithic and high for template-based approaches. In each of these two solutions, the user has the flexibility to intervene before the workflow is deployed on an execution infrastructure. However, for the latter, the developer has the additional flexibility of choosing a template. Finally, for a staged architecture, the control given is the highest. The user in this case can inspect and modify the output of both the logical and physical stages.

Handling Composition Failure: Another important criterion that we consider is the ability of a particular approach to resolve failures. Note that 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 fail-

ure and provide alternate workflows to the user. Further, the failure resolution and recovery steps might be possible only by interacting with the developer. In the interleaved approach, composition and execution go hand-in-hand, therefore resolving a failture requires a roll-back of the services executed. The staged approach offers maximum failure resolution because of the decomposition of WSCE process into multiple stages, where the developer can provider her inputs for verification and refinement. A monolithic approach offers a medium level of failure resolution. In a template–based solution, the ability to recover from failtures is restricted to the choice of services for tasks listed out in the template.

Adaptation during Execution: Often, incorporation of environmental changes and domain knowledge of a developer is crucial for improving the quality and efficiency of the composition process. When a service does not meet its advertised QoS values or in scenarios where new services become available or existing ones unavailable, a WSCE system can benefit from incorporating feedback. In template–based and monolithic systems, feedback can come from the execution infrastructure to the composition phase. Whereas in the former, adaptation is restricted to picking new instances for the template (in extreme cases, even the template itself), the latter can involve recomposing the workflow from scratch. In a staged architecture, feedback can be provided from one stage to another. Each stage benefits from the information obtained from the subsequent stage(s). In an interleaved approach, at each step of composition and execution, the choice of service to be executed for the step can incorporate not only the changes in the environment, but also the results of service executions for previous steps. This provides a high level of adaptavity to this approach.

Information Modeling: During development of a WSCE framework, one needs to capture the information that would be used during composition and execution. In all the approaches, the developer has to determine how to model the functional and non-functional capabilities of available services so that it becomes easy to discover, invoke and compose them. For a staged architecture, service types need to be differentiated from service instances. They can either be stored in a single registry or in separate ones. Modeling of templates becomes important in a template–based system, precise capture of different scenarios through templates leads to efficient domain handling. For the monolithic and interleaved approaches, we just need to model the instances, therefore, information modeling is simpler.

4 Case Studies

We now present three different scenarios that employ composition and execution of web services. For each of these scenarios, we outline the desired properties of a WSCE scheme followed by choice of an approach for the particular scenario.

4.1 Application Integration in Telecom

Mobile telephony service providers have to continually develop compelling 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.

The user applications in this domain often rely on several, relatively simple building blocks – user profile look-ups, address books, location-tracking services, accounting and billing services, etc. An example of an application is a helpline service for a consumer electronics company which would accept service requests (e.g., complaints) from users and directs them to be satisfied by remote help-desk or a field agent based on cost versus severity trade-off. Many of the building blocks are already in place, the cost of building new applications can be alleviated by using web services as the underlying abstraction for these building blocks.

We now list 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 a large range of functionalities and the number of service instances.
2. Adaptability - Plans generated should be responsive to changes in the run–time environment. For example, it should be easy to incorporate new services as well as to account for departure of existing ones. Moreover, any significant changes in the QoS offerings of the component services should impact the creation and execution of the composite service.
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 an abstract plan, to the creation of an executable workflow, and finally its deployment on a run-time infrastructure.

In this scenario, we argue that a staged approach for service composition and execution is appropriate. 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 provides 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. A solution based on the staged WSCE approach for this scenario is discussed in [1].

4.2 Travel Reservation Using an Online Service

In [2], a scenario for online travel reservation is described which requires interactive building of travel itineraries and then their monitoring for successful

execution. In this scenario, the number of service providers is known and limited to those with which the travel site has partnership. There can be unexpected disruptions to plans (e.g., flight delays, price drops, weather) but the response allowed is low level - propogate constraints on the itinerary but not take new decisions. Further the user is not closely involved after itinerary creation unless new decisions are needed. The plans can be broadly categorized across different travel requests (for example, one category includes flights, hotel and car reservations while another category includes only flight and hotel reservations).

We argue that here, a template-based approach is most appropriate since itineraries follow a well-defined set of patterns and this fact could be further leveraged for simpler user-interfaces and output verification. Interestingly, an interleaved approach is also acceptable in this scenario if the application developer wants to internalize the failure handling during monitoring or execution of the plan. The monolithic and staged approaches would not leverage the domain properties effectively.

The solution proposed by the authors in [2] consists of the Heracles constraint planner which helps the user navigate the space of itinerary choices, and the Thesus information agent platform which executes information gathering tasks to monitor the environment for possible disruptions and adapt the itinerary to the changes. Their solution can be considered as a variation of the template approach where templates are implicit in the constraint rules used to build the composition.

4.3 Integration of Data

Consider the data integration scenario in real estate presented in [19]. Here, the user wants to find out the value of properties that a given company owns in a given city. The available services are: a yellow pages service to find the address of a business, two services to find property values in specific geographic counties, and a service to find county of a city. Composition of services is performed by a mediator that accepts user query and reformulates it as a combination of source queries on a subset of available services.

In this scenario, a staged approach is not appropriate because the user has little interest in how the plan is composed or executed. Moreover, failure of execution is less of a concern here because the web services are data sources and thus do not interfere with the applicability of one another. Therefore, the monolithic and template-based approaches are also of less relevance. We argue that an interleaved approach is most appropriate in this scenario since it can effectively address the functional and non-functional aspects of realizing the user request in an integrated setting. For example, depending on the county value, choice of the service to find property values can be made.

5 Conclusion

In this paper, we have attempted to classify and qualitatively evaluate various approaches for web service composition and execution. We came up with

a formalization of the WSCE process, a classification of existing solutions into different approaches based on the kind of inputs they take and the amount of flexibility given to the user to intervene in the process, and an in-depth evaluation of these approaches. We also demonstrated the usefulness of this understanding in selecting appropriate service composition approaches for three different scenarios. Our work offers a unifying perspective on WSCE systems and fills a crucial void.

References

1. Vikas Agarwal, Koustuv Dasgupta, Neeran Karnik, Arun Kumar, Ashish Kundu, Sumit Mittal, and Biplav Srivastava. A Service Creation Environment based on End to End Composition of Web Services. In *Proceedings of the 14th International World Wide Web Conference*, 2005.
2. J. Ambite, G. Barish, C. Knoblock, M. Muslea, J. Oh, and S. Minton. Getting from here to there: Interactive planning and agent execution for optimizing travel. In *Proc. IAAI*, 2002.
3. J. Blythe et al. The Role of Planning in Grid Computing. Proceedings of International Conference on AI Planning and Scheduling, 2003.
4. Soon Ae Chun, V. Atluri, and N. R. Adam. Policy-based Web Service Composition. In *Research Issues on Data Engineering: Web Services for e-Commerce and e-Government Applications (RIDE)*, 2004.
5. U. Dal-Lago, M. Pistore, and P. Traverso. Planning with a Language for Extended Goals. In *Proceedings of AAAI*, pages 447–454, 2002.
6. M. DesJardins, E. Durfee, C. Ortiz, and M. Wolverton. A Survey of Research In Distributed, Continual Planning. *AI Magazine*, 20(4):13–22, 1999.
7. M. B. Do and S. Kambhampati. Sapa: A Scalable Multi-objective Heuristic Metric Temporal Planner. *Journal of AI Research*, 20:155–194, 2003.
8. Kutluhan Erol, James Hendler, and Dana S. Nau. HTN Planning: Complexity and Expressivity. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*, Seattle, Washington, USA, 1994.
9. Jos Luis Ambite et al. Getting from here to there:. interactive planning and agent. execution for optimizing travel. In *IAAI*, 2002.
10. F. Giunchiglia and P. Traverso. Planning as Model Checking. In *Proceedings European Conference on Planning*, 1999.
11. A. Johnson, P. Morris, N. Muscettola, and K. Rajan. Planning in Interplanetary Space: Theory and Practice. In *Proceedings AIPS*, 2000.
12. Henry Kautz and Joachim Paul Walser. State-space Planning by Integer Optimization. In *Proceedings Fifteenth National Conference on Artificial Intelligence (AAAI-99)*, 1999.
13. D. McDermott. Estimated-Regression Planning for Interactions with Web Services. In *Proc. AIPS*, 2002.
14. S. Ponnekanti and A. Fox. SWORD: A Developer Toolkit for Web Service Composition. In *Proceedings of the 11th International World Wide Web Conference*, 2002.
15. E. Sirin, B. Parsia, and J. Hendler. Composition-driven Filtering and Selection of Semantic Web Services. In *AAAI Spr. Symposium on Semantic Web Services*, March 2004.

16. Evren Sirin, Bijan Parsia, and James Hendler. Template-based Composition of Semantic Web Services. In *AAAI Fall Symposium on Agents and the Semantic Web*, Virginia, USA, November 2005. To Appear.
17. B. Srivastava. Automatic Web Services Composition Using Planning. In *Proceedings of KBCS, Mumbai, 2002.*, pages 467–477, 2002.
18. B. Srivastava and J. Koehler. Web Service Composition - Current Solutions and Open Problems. ICAPS 2003 Workshop on Planning for Web Services, 2003.
19. S. Thakkar, C. Knoblock, and J. Ambite. A view integration approach to dynamic composition of web services. In *Proc. ICAPS 2003 Workshop on Planning for Web Services*, 2003.
20. P. Traverso and M. Pistore. Automated Composition of Semantic Web Services into Executable Processes. In *3rd International Semantic Web Conference*, 2004.
21. L. Zeng, B. Benatallah, A Ngu, M. Dumas, J. Kalagnanam, and H. Chang. QoS-Aware Middleware for Web Services Composition. *IEEE Transactions on Software Engineering*, 30:311–327, 2004.