Scalable Orchestration Strategy for Automatic Service Composition

Hiroshi Mizugai*, Incheon Paik*, Wuhui Chen*
*School of Computer Science and Engineering
University of Aizu
E-mail: h.mizugai@gmail.com, paikic@u-aizu.ac.jp, m5132201@u-aizu.ac.jp

Abstract

The goal of Automatic Service Composition (ASC) is to create value-added services automatically from existing services. Most research into service composition is based on four stages (planning, discovery, selection, and execution) and their variations, and do not consider nested dynamic services where ASC calls inner ASCs internally during composition. In practice, there are many situations where dynamic services and existing services are combined in ASC. This research proposes a strategy for orchestrating the existing four-staged composition and nested dynamic service composition. Two approaches are considered: top-down and bottom-up. Top-down composition can create new services with management using a main composer, while bottom-up composition creates a new service under human-oriented direction. For the two proposed solutions, core procedures are described, and prototype systems are implemented in one domain. Finally, the performance of top-down composition is evaluated.

Keywords
Automatic Web Services Composition, *Top-down composition*, *Bottom-up composition*, *Dynamic service*.

1. Introduction

Many Web services are currently available on the Internet. A service can be characterized by its bundle of Inputs, Outputs, Preconditions, and Effects (IOPE). By combining IOPEs of the service, new valued-added services can be created. There has been much research into creating new services automatically, and this is known as Automatic Service Composition (ASC). Previous ASC research [1][2][3][4] assumed that the services involved in the composition are only existing services (known as static services), which are published in service repositories. Therefore, such composition approaches cannot handle nested dynamic services created internally by the inner ASC at composition time. In practice, there are many situations where dynamic services should be combined to create new value-added services using ASC. This research proposes two approaches, top-down composition and bottom-up composition, to enable nested dynamic service composition. Top-down composition creates new services with management by a main composer, while bottom-up composition creates new services with human-oriented direction. For top-down and bottom-up composition, the core procedures that fulfill the nested dynamic service composition are proposed.

The remainder of this paper is organized as follows: Section 2 introduces the background related to ASC. Section 3 describes the proposed approaches and core procedures that fulfill the needs of dynamic service composition. Section 4 shows the results of the implemented prototype systems using two approaches. Section 5 evaluates the performance of top-down composition by measuring the composition time of the nested dynamic structure.

2. Automatic Service Composition

ASC is used to create composite services automatically. To realize this, many methods have been proposed in the literature [1][2][3]. In this paper, several proposed methods utilize ASC and follow the staged approach [4] because of its flexibility and scalability for composing new services compared with the other ASC methods. The ASC method introduces the planning, discovery, selection and execution stages explicitly in the composition to realize a more general ASC, based on the concepts in [5]. The composition is performed using the following steps. The planning stage uses the Artificial Intelligence (AI) planners such as STRIPS [6] and HTN Planner [7] to create workflows, which are sequences of service types. A service type requires information to discover the service instances that have the necessary operators to compose the new services. After the workflows are generated, the available services are discovered for each service type of workflow in the discovery stage through some proposed methods such as keyword-based, table-based, concept-based, ontology-based [8], and statistical approaches [9]. The selection stage selects the optimal services using measures [10][11] related to quality of service (QoS) and user constraints provided by a client. Finally, in the execution stage, the workflows of the service operators are represented in a workflow language (for example Petri-net [12], Business Process Execution Language (WS-BPEL) [13], Web Service Modeling Language

(WSMO) [14], and OWL-S [15]) that can be invoked by an execution engine.

3. Scalable Orchestration Strategy

3.1 Scalable Orchestration

Top-down composition and bottom-up composition are proposed for managing static services, which are existing services and dynamic services that are created by ASC. Top-down composition invokes composing automatically for the provided high-level goal, as in traditional ASC, while bottom-up composition invokes the composition incrementally with several human interventions required for creating the user-oriented services. Using these approaches, the services or nested dynamic services shown in Figure 1 can be composed. The important concept here is that ASC can also internally call ASC. In Figure 1, the child service nested in the dynamic service is also a dynamic service. Figure 2 shows the composition steps of the scalable ASC adopted by the proposed methods. The aqua circles are static services, and the red circles are dynamic services. The orange circles indicate that the child services that are nested in dynamic services are also dynamic services.

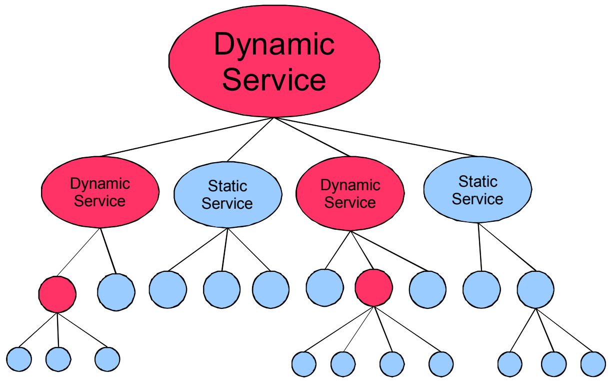

Figure 1: Scalable Orchestration for Automatic Service Composition

3.2 Top-down Composition Approach

The orchestration strategy for services using top-down composition can be considered as an extension of the conventional ASC approaches. By considering the characteristics of dynamic services, the discovery stage and the execution stage are modified in top-down composition. The important characteristic of dynamic services is that one dynamic service potentially conforms to more than one static service. In the ASC described here, several sequences of service types can be created in the planning stage, and the optimal service instances that have the required operators are selected for the service types after the invocation of the discovery stage. Finally, the ASC outputs the composite services, combining the operators of the selected services. Procedure 1 shows the fundamental procedure of top-down composition, where the essential methods and parameters are shown to fulfill the desired composition.

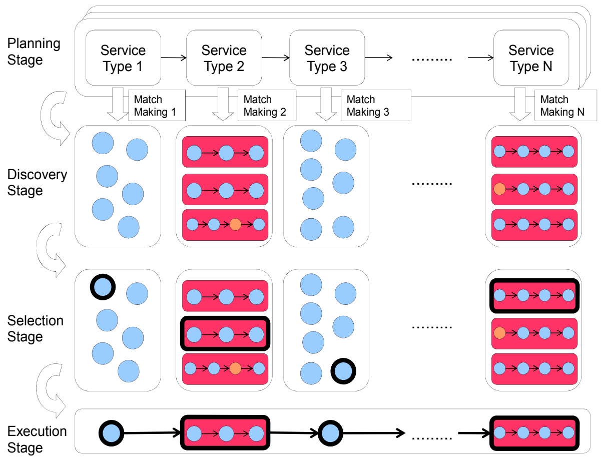

Figure 2: Composition Flow of Scalable Orchestration

The flow of the procedure is as follows: firstly, the ASC function takes the *PlanningDomain* variables and a set of *UserConstraints*. *PlanningDomain* has a problem space and a planning space that are used to invoke the planning for one domain. The *UserConstraint* set contains information about the user constraints. The user constraints and the functional goal can be varied according to the user's requirement. In line 7, the *generateAbstractWorkflow* method receives the *PlanningDomain* variable, and generates an *AbstractWorkflow* set. The size of *AbstractWorkflowset* is determined by the number of generated sequences from *ServiceTypes*. This process corresponds to the planning stage of Figure 2.

The discovery stage of top-down composition is shown between lines 9 and 20. This process discovers the service instances for each service type. If the *ServiceInstance* set cannot be discovered for a *ServiceType* from the service repositories, or the *servicetype* is regarded as a dynamic service, the internal ASC is invoked recursively to generate a *ServiceInstance* set for the *ServiceType*. Following the call of the inner ASC, a variable *PlanningDomain* is generated by the *generatePlanningDomain* method. The method creates the needed information (the required planning and problem spaces) for a planner, which can then derive the desired workflows for the domain. Using the generated *PlanningDomain*, the inner ASC can create services that fulfill the functional goal in the domain. The methods of *setServiceInstances* and *setServiceTypes* described in lines 17 and 19 are used to write the obtained parameters to the corresponding variables. The discovery stages in Procedure 1 correspond to the discovery stage shown in Figure 2.

```
Require: PlanningDomain pd, UserConstraint uc[]
Ensure: ExecutableWorkflow[] ←
        ASC(PlanningDomainpd,UserConstraintuc[])
1: Let ServiceType[] be service types for an AbstractWorkflow
2: Let ServiceInstance[] be service instances for a ServiceType
3: Let AbstractWorkflow[] be workflows, consisting of a set of
        ServiceTypes
4: Let ConcreteWorkflow[] be workflows, consisting of a set of
        ServiceTypes
5: Let ExecutableWorkflow[] be executable workflow languages
6: /** Planning Stage **/
7: AbstractWorkflow[] ← generateAbstractWorkflow(pd)
8: /** Discovery Stage **/
9: for i = 0 to AbstractWorkflow[].length do
10:    ServiceType[] ← AbstractWorkflow[i].getServiceTypes()
11:    for j = 0 to ServiceType[].length do
12:        ServiceInstance[] ← discoverServices(ServiceType[j])
13:        if ServiceInstance[].length is 0 or ServiceType[j]
               calls a dynamic service then
14:            pd2 ← generatePlanningDomain(ServiceType[j])
15:            ServiceInstance[] ← (ServiceInstance)ASC(pd2, uc[])
16:        end if
17:        ServiceType[j].setServiceInstances(ServiceInstance[])
18:    end for
19:    AbstractWorkflow[i].setServiceTypes(ServiceType[])
20: end for
21: /** Selection Stage **/
22: ConcreteWorkflow[] ← doNFPsSelection(pd, uc[],
        AbstractWorkflow[])
23: /** Prepared for Execution Stage **/
24: ExecutableWorkflow[] ← generateExecutableServices(pd,
        ConcreteWorkflow[])
25: for i = 0 to ExecutableWorkflow[].length do
26:     ExecutableWorkflow[i].temporalpublish()
27: end for
28: if Process is in Inner ASC then
29:     return ExecutableWorkflow[]
30: else
31:     return InvocationResultsof ExecutableWorkflow[]
32: end if
```

Procedure 1: Automatic Service Composition by Top-down Composition

In line 22, the *doNFPsSelection* method is called, and the method generates a set of *ConcreteWorkflows*. The *doNFPsSelection* method selects the optimal *ServiceInstance* from the discovered *ServiceInstances* for each *ServiceType* by using the QoS information and the *UCS* measures. So the generated *ConcreteWorkflow* set has optimal service instances for the service types. By the processes of line 22, the results (such as the selection stage in Figure 2) can be obtained.

The execution stage of top-down composition shown in Figure 2 corresponds to line 25 and line 32 in Procedure 1. In line 24, the *generateExecutableServices* method receives the *PlanningDomain* and *ConcreteWorkflow* sets, and generates a set of *ExecutableWorkflows*. After the generation of the *ExecutableWorkflow* set, it is published temporarily as services that can be accessed by a client application. If the process is in the inner ASC, it returns the generated *ExecutableWorkflow* set denoted in line 29. In this process, the parent ASC can obtain the *ServiceInstance* set from the Inner ASC shown in line 15. Also in line 15, the *ExecutableWorkflow* set is converted to a *ServiceInstance* set. This implies that *ExecutableWorkflow* is substantially similar to *ServiceInstance*. In line 31, the invocation results of the executable workflow are returned. This process is invoked when the highest services located on the nested structure are generated. Consequently, by using Procedure 1, the inner ASC can be called recursively, and the nested dynamic composition structure can be generated.

3.3 Bottom-up Composition

The method of bottom-up composition utilizes several human interventions for fulfilling the nested dynamic service composition. In practical applications, manual service selections from the client are very important because there are cases where some undesirable services selected automatically by ASC cannot be satisfied for the clients. To address this problem, bottom-up composition is proposed. Bottom-up composition can be simply realized by changing the selection stage of top-down composition. Instead of a selection based on QoS information and UCS measures, human selections are utilized by interacting with a client. Many dialogue forms can be considered to fulfill the human selections for enquiring about the service selections for the client. For example, the following questions shown in dialogue forms can be considered easily: *What sequence of service types is preferred from the planner? What service instance is preferred for the service type?* In the practical case, the concrete suggestions of the service selection should be varied for the applied domain.

4. Realizing Scenarios and Implementations

4.1 Scenario

A user wants to go to San Francisco from Aizu-Wakamatsu to attend a conference. The user wants to delegate a service agent to prepare a trip plan, and wants to reserve and pay for the services offered by the trip plan. This scenario assumes that the three services, *Trip Scheduling, Trip Reservation, and Trip Payment,* are required to fulfill the desires of the client. The *Trip Scheduling* service produces service IDs of the vehicles (for example, buses, trains, and airplanes) that could be taken to arrive in San Francisco, along with the hotel that could be booked in San Francisco, and then receives a departure time for the client. The *Trip Reservation* service receives service IDs, and reserves only the bookable service IDs. The *Trip Payment* service receives service

IDs, and makes the payment for the targeted service IDs. The *Trip Agent* service combines the three services into one. The main point here is that the *Trip Scheduling* service is not in the service repositories. In such a situation, the roles of the top-down and bottom-up composition methods are to create such a *Trip Agent* service, which calls the inner ASC to create the *Trip Scheduling* service with the nested structure by adopting the procedures described in Section 3.

4.2 Top-down Composition

The ASC first attempts to create a *Trip Agent* service with an initial state, a goal state, and the user constraint provided by the client. The initial state and the goal state are included in the part of the problem space of the *PlanningDomain* variable in the context of Procedure 1. The ASC then plans the sequences of service types according to the *PlanningDomain* variable. As previously described, *PlanningDomain* provides the knowledge of the planner's solution to derive the sequences for one domain. The planner outputs one sequence of service types, which consists of a *trip-schedule*, *trip-reserve*, and *trip-payment*.

The discovery stage is then invoked. The discovery process is invoked for the service types from the order of the derived sequence. Although ASC attempts to find the services that correspond to the *trip-scheduling* service type, the services cannot be discovered because there is no information in the mapping data. At this time, as shown in Procedure 1, a *PlanningDomain* variable is created according to the *trip-scheduling* service type. Then, the inner ASC is called, receiving the *PlanningDomain* generated from the generic *PlanningDomain* method and the user constraints. Similar to the pre-invoked ASC, the inner ASC attempts to do the planning for the *trip-scheduling* domain. Figure 3 shows the results of the sequences of the service types from the planner.

Next, the inner ASC performs the discovery stage on the pre-invoked ASC. Figure 4 shows the results of the discovered services for the workflow (#1) in Figure 3.

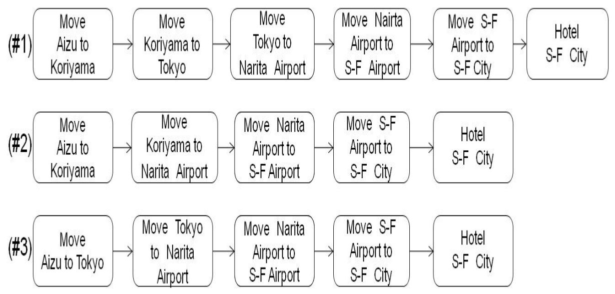
(#1)
Move Aizu to Koriyama
Move Koriyama to Tokyo
Move Tokyo to Narita Airport
Move Nairta Airport to S-F Airport
Move S-F Airport to S-F City
Hotel S-F City
(#2)
Move Aizu to Koriyama
Move Koriyama to Narita Airport
Move Narita Airport to S-F Airport
Move S-F Airport to S-F City
Hotel S-F City
(#3)
Move Aizu to Tokyo
Move Tokyo to Narita Airport
Move Narita Airport to S-F Airport
Move S-F Airport to S-F City
Hotel S-F City

Figure 3: Sequences of Service Types by Inner ASC

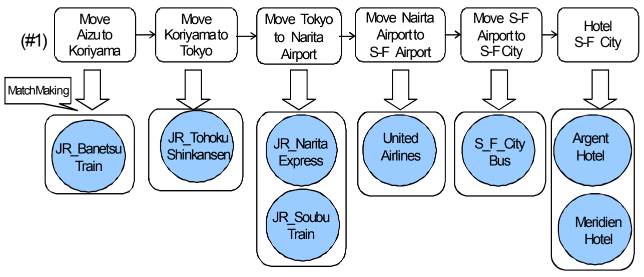

Figure 4: Discovered Services for Workflow #1

After the required service instances are discovered, the selection stage is invoked in the inner ASC. The utilized selection method selects the optimal service instances from the discovered service for each service type. Figure 5 shows the selected service instances for the workflow (#1).

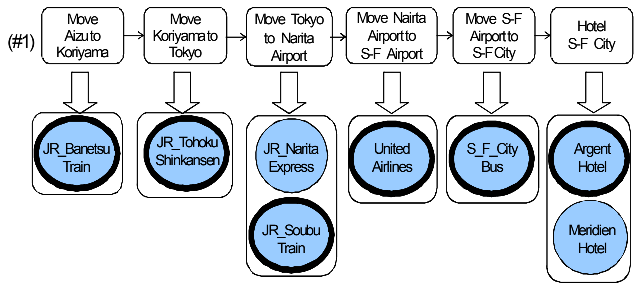

Figure 5: Selected Services for Workflow #1 in the Inner ASC

After the selection stage, the execution stage is invoked in the inner ASC. Three composite services are published as Web services by using Apache Tuscany, combining the inputs and outputs of the selected operators of the services derived from the selection stage. The three Web services are *trip scheduling0*, *trip scheduling1*, *and trip scheduling2.* Figure 6 shows the generated method, which is included in the *trip scheduling2* service. In Figure 6, the outputs of the operators folded by the bold line are returned. The order of the operator's flow was preliminarily defined, and the validities of the data types among the inputs and the outputs of the operators were also known from predefined knowledge.

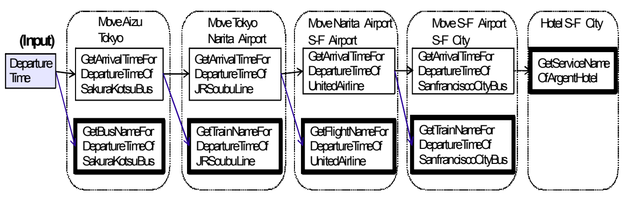
Method: String[] invoke_trip_scheduling2(int hour, int min)
(Input)
Departure Time
Move Aizu Tokyo
GetArrivalTimeFor DepartureTimeOf SakuraKotsuBus
GetBusNameFor DepartureTimeOf SakuraKotsuBus
Move Tokyo Narita Airport
GetArrivalTimeFor DepartureTimeOf JRSoubuLine
GetTrainNameFor DepartureTimeOf JRSoubuLine
Move Narita Airport S-F Airport
GetArrivalTimeFor DepartureTimeOf UnitedAirline
GetFlightNameFor DepartureTimeOf UnitedAirline
Move S-F Airport S-F City
GetArrivalTimeFor DepartureTimeOf SanfranciscoCityBus
GetTrainNameFor DepartureTimeOf SanfranciscoCityBus
Hotel S-F City
GetServiceName OfArgentHotel

Figure 6: The Outputs of the Operators Folded by the Bold Line Are Returned

At this time, the process of the inner ASC (shown in line 13 of Procedure 1) is finished. Recall that the process is in the discovery stage in the parent ASC. So the parent ASC attempts to discover the remaining service types, which are *trip-reserve* and *trip-payment*. Then, like the inner ASC, the selection stage and execution stage are invoked. Finally, the *Trip Agent* service is generated as a Web service. In this way, the process of top-down composition is realized in the scenario.

4.3 Bottom-up Composition

As described above, the characteristic of bottom-up composition is that a client intervenes in the selection stage. In this section, bottom-up composition is implemented for the scenario. To allow suggestions for the service selection, a graphical user interface (GUI) service should be provided for the user that displays the necessary information about the involved services.

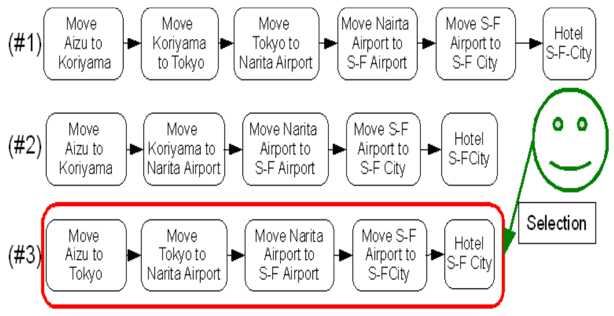
(#1)
Move Aizu to Koriyama
Move Koriyama to Tokyo
Move Tokyo to Narita Airport
Move Nairta Airport to S-F Airport
Move S-F Airport to S-F City
Hotel S-F-City
(#2)
Move Aizu to Koriyama
Move Koriyama to Narita Airport
Move Narita Airport to S-F Airport
Move S-F Airport to S-F City
Hotel S-FCity
(#3)
Move Aizu to Tokyo
Move Tokyo to Narita Airport
Move Narita Airport to S-F Airport
Move S-F Airport to S-FCity
Hotel S-F City
Selection

Figure 7: An Example of User Selection for Workflows

For example, the images and the hyperlinks related to the services can be provided in the GUI. In our prototype systems, the selection of the sequences of the service types and the selection of service instances for the service types are provided for the user. For example, as shown in Figure 7, if the user inputs "3", then "WORKFLOW#3" will be selected. Consequently, the involved services are incrementally decided by the client until the top-most service is composed. As seen above, bottom-up composition could be invoked successfully, enabling user-oriented service composition.

5. Evaluation

In this Section, the performance of top-down composition is evaluated. To evaluate the performance, a composition time, which is the completion time of Procedure 1, was employed. In the evaluation, variation of the composition time is observed by changing the nested dynamic service structures. In the evaluation, simple calculation services were combined to create the new services in each ASC, and a depth of the nested structure (from 1 to 6) was defined to measure the composition time. To evaluate the performance with practical situations, consistent settings are needed.

5.1 Settings for the Environment

To create the nested structure of dynamic services to simulate a practical situation, the following settings were adopted: one ASC generates 3 to 5 workflows randomly (sequences of service types), and 1 or 2 ASCs occur in the generated workflows. The size of the service types for a workflow was set between 3 and 5. All of the sequences (up to 5) of the service types were preliminarily prepared. The next section evaluates the performance of top-down composition to evaluate the computational cost.

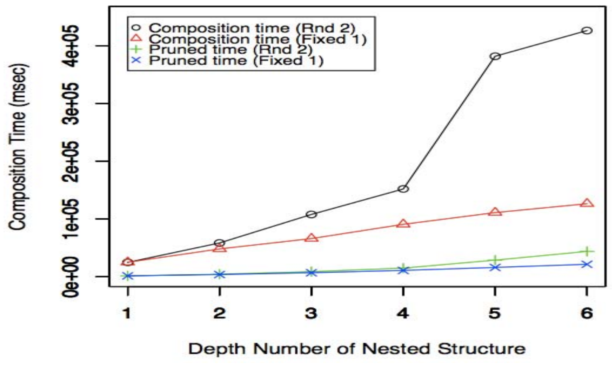

Figure 8: Composition Time of Top-down Composition Compared with the Depth of the Nested Structure

5.2 Performance of Top-down Composition

Figure 8 shows the composition times for top-down composition compared with the depth of the nested structure. Note that the times are shown in milliseconds, and all of the times are the averages of 10 invocations. From depth 1 to depth 4, the composition time grew moderately. However, from depth 4 to depth 6 the composition time grew rapidly. From the settings, it is assumed that one ASC calls the inner ASC once or twice. Therefore, in the worst case, the composition time grows according to $O(2^N)$, where N is the depth of the composition. The composition time depends on the number of services created by the inner ASCs. Figure 9 shows the number of services created by ASCs in the composition compared with the depth of the nested structure. If the average time of one ASC can be estimated, the completion time of the composition can be estimated by multiplying the number of created services by the average time of the creation for one service. In the practical case, the composition time of each ASC and the number of the service creations by the ASCs provide the completion time of the composition, although the concrete selection method was not applied in the evaluation.

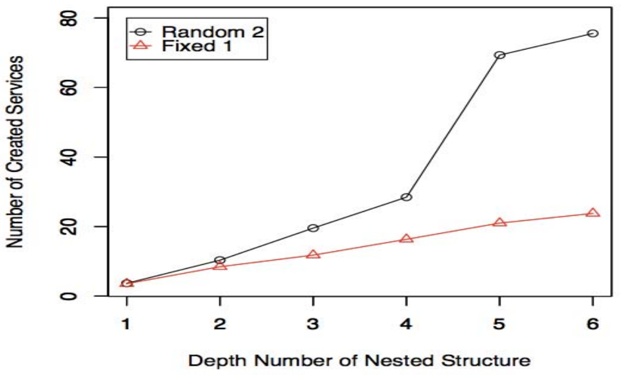

Figure 9: Number of Created Services by Top-down Composition Comparing with the Depth of the Nested Structure

6. Conclusion and Future Work

In this paper, orchestration approaches for nested dynamic service composition (both top-down and bottom-up composition) were proposed to enable scalable ASC, which involves static and dynamic services composed together. Top-down composition has a nested dynamic service composition like in conventional ASC. Bottom-up composition handles the composition with several human interventions required to create the user-oriented services. For each proposed method, the core procedure was proposed. In addition, prototype systems were implemented using the scenario of the trip domain, and these showed the proposed methods were invoked successfully. Moreover, performance of top-down composition was evaluated by measuring composition times, and the increase of the composition time was shown to depend on the depth of the nested structure and rate of sub-node creation. For future work, there are cases that an ASC outputs only data derived from the generated service instances, although the proposed procedures assume that an ASC outputs the service instances. To manage both completion states, more general types and procedures will be proposed. Context adaptable service selection mechanisms and exception handling principles will be incorporated into this architecture.

7. References

[1] S. Narayanan and S. McIlraith, "Simulation, Verification and Automated Composition of Web Services," Proceedings of 11th International WWW Conference, Hawaii, 2002.

[2] B. Srivastava and J. Koehler, "Web Service Composition—Current Solutions and Open Problems", Proceedings of ICAPS Workshop on Planning for Web Services, 2003.

[3] N. Milanovic and M. Malek, "Current Solutions for Web Service Composition," IEEE Internet Computing, pp. 51–59, Nov./Dec. 2004.

[4] V. Agarwal, G. Chafle, S. Mittal, and B. Srivastava, "Understanding Approaches for Web Service Composition and Execution," Proceedings of COMPUTE 2008, Bangalore, India, 2008.

[5] D.B. Claro, P. Albers, and J.K. Hao, "Web Services Composition in Semantic Web Service, Processes and Application", Springer, New York, pp. 195–225, 2006.

[6] R.E. Fikes and N.J. Nilsson. "Strips: A New Approach to the Application of Theorem Proving to Problem Solving". In J. Allen, J. Hendler, and A. Tate, editors, Readings in Planning, pp. 88–97. Kaufmann, San Mateo, CA, 1990.

[7] D. Nau, T. Au, O. Ilghami, U. Kuter, J. Murdock, D. Wu, and F. Yaman. SHOP2: "An HTN planning system". Journal of Artificial Intelligence Research, 20:379–404, 2003.

[8] M.P. Paolucci, T. Kawamura, T.R. Payne, and K. Sycara, "Semantic Matching of Web Services Capabilities," Proceedings of the First International Semantic Web Conference, Sardinia, Italy, 2002.

[9] M. Klusch, B. Fries, and K. Sycara, "Automated Semantic Web Service Discovery with OWLS-MX," Proceedings of AAMAS, Hakodate, Hokkaido, 2006.

[10] Zeng, L., Benatallah, B., Ngu, A.H.H., Dumas, M., Kalagnanam, J., and Chang, H. "QoS aware Middleware for web service composition", IEEE Trans. Software Engineering, 30(5), 2004.

[11] I. Paik, D. Maruyama, and M. Huhns, "A Framework for Intelligent Web Services: Combined HTN and CSP Approach," Proceedings of IEEE International Conference on Web Services (ICWS 2006), Chicago, 2006.

[12] R. Hamadi and B. Benatallah, "A Petri Net-based Model for Web Service Composition", Proceedings of Australasian Database Conference (ADC 2003), Adelaide, Australia, 2003.

[13] T. Andrews et al., "Business Process Execution Language for Web Services version 1.1", BEA Systems, IBM, Microsoft, SAP AG, and Siebel Systems (May 2003), http://www106.ibm.com/developerworks/library/ws-bpel/.

[14] WSMO. "The Web Service Modeling Ontology (WSMO) Primer. Final Draft". Available at: http://www.wsmo.org/TR/d3/d3.1/v0.1/, 2005.

[15] OWL Services Coalition, "OWL-S: Semantic Markup for Web services, OWL-SWhite Paper", http://www.daml.org/services/owl-s/1.0/owls. pdf, 2003.