

MoSCoE: A Specification-Driven Framework for Modeling Web Services using Abstraction, Composition and Reformulation

Jyotishman Pathak

Artificial Intelligence Research Lab
Department of Computer Science
Iowa State University
Ames, IA 50011-1040 USA
jpathak@cs.iastate.edu

Abstract. We propose a new framework for modeling Web services based on the techniques of abstraction, composition and reformulation. The approach allows users to specify an *abstract* and possibly incomplete specification of the composite (goal) service. This specification is used to select a set of suitable component services such that their *composition* realizes the desired goal. In the event that such a composition is unrealizable, the cause for the failure of composition is determined and is communicated to the user thereby enabling further *reformulation* of the goal specification. This process can be iterated until a feasible composition is identified or the user decides to abort.

1 Introduction

With the recent advances in networks, information processing and WWW, automatic composition of Web services has emerged as an area of intense research in both academia and industry. The main objective of these approaches is to construct and deploy complex workflows of composite Web services by leveraging autonomously developed software components or services in several application domains including e-Enterprise, e-Business and e-Science. However, despite the recent progress, the current state-of-art in developing composite services has several limitations:

Complexity in Modeling Composite Services: For specifying the functional requirements, the service developer is required to provide a complete specification of the composite (goal) service. Consequently, the developer has to deal with the cognitive burden of handling the entire composition graph, which becomes complex to manage with the increasing complexity of the goal service. Instead, it will be more practical to allow developers to begin with an abstract, and possibly incomplete specification, that can be incrementally modified until a feasible composition is realized.

Analyzing Failure of Composition: The existing techniques for service composition adopt a ‘single-step request-response’ paradigm for modeling composite services. That is, if the goal specification provided by the service developer cannot be realized by the composition analyzer (using the set of available components), the entire process fails. As opposed to this, there is a requirement for developing approaches that will help identify the cause(s) for failure of composition and guide the developer in applying that information for appropriate reformulation of the goal specification in an iterative manner.

Consideration of Non-Functional Characteristics: Barring a few approaches, most of the techniques for service composition focus only on the functional aspects of the composition. In practice, since there might be multiple component services that can provide

the same functionality, it is of interest to explore the non-functional properties of the components to reduce the search space for determining compositions efficiently.

Handling Differences in Service Semantics: Individual Web services needed for realizing a desired functionality are often developed by autonomous groups or organizations. Consequently, semantic gaps, arising from different choices of vocabulary for specifying the behavior of the services, are inevitable. This requires frameworks for assembling complex Web services from independently developed component services to provide support for bridging the semantic gaps.

Motivated by these concerns, the proposed research develops a new service composition framework, MoSCoE¹ (Modeling Web Service Composition and Execution) to overcome some of the aforementioned limitations. Specifically, our work is aimed at:

- Introducing a new paradigm for modeling Web services based on abstraction, composition, and reformulation. The proposed approach allows users to incrementally develop composite services from their abstract descriptions.
- Developing a sound and complete algorithm for selecting a subset of the available component services that can be assembled into a sequential [1] and parallel [2] composition that realizes the goal service with the user-specified functional and non-functional requirements [3].
- Proposing techniques to identify the cause(s) for failure of composition to assist the user in modifying and reformulating the goal specification in an iterative fashion.
- Building on current approaches for associating semantics to Web services [4, 5] using ontologies and techniques for specifying mappings [6] between ontologies.
- Demonstrating the feasibility of the approach with real-world applications in bioinformatics [6] and electric power systems [7].

The rest of the paper is organized as follows: Section 2 talks about our contributions and results accomplished so far, Section 3 briefly discusses related work in service composition and finally Section 4 ends with conclusions and directions for further research.

2 The MoSCoE Framework

Figure 1 shows the architectural diagram of the MoSCoE framework mentioned above. Developing services in MoSCoE is based on the techniques of *abstraction*, *composition* and *reformulation*. The system accepts from the user, an *abstract* (high-level and possibly incomplete) specification of the goal service. In our current implementation, the goal service specification takes the form of an UML state machine that provides a formal, yet intuitive specification of the desired goal functionality. This goal service and the available component services (published by multiple service providers) are represented using labeled transition systems augmented with state variables, guards and functions on transitions, namely, Symbolic Transition Systems² (STS). In addition, the STSs are semantically annotated using appropriate domain ontologies from a repository by importing OWL ontologies into the UML model [10]. MoSCoE assumes that these ontologies (and mappings between them) are specified by a domain expert using existing tools such as INDUS [6]. The user also provides non-functional requirements (e.g., *cost*, *reliability*) that need to be satisfied by the goal service.

MoSCoE manipulates these input data (user-provided service specification and published component service descriptions) and automatically identifies a composition that

¹ <http://www.moscoe.org>

² The STS specifications for component services can be obtained from service descriptions provided in high-level languages such as BPEL or OWL-S by applying translators similar to those proposed in [8, 9].

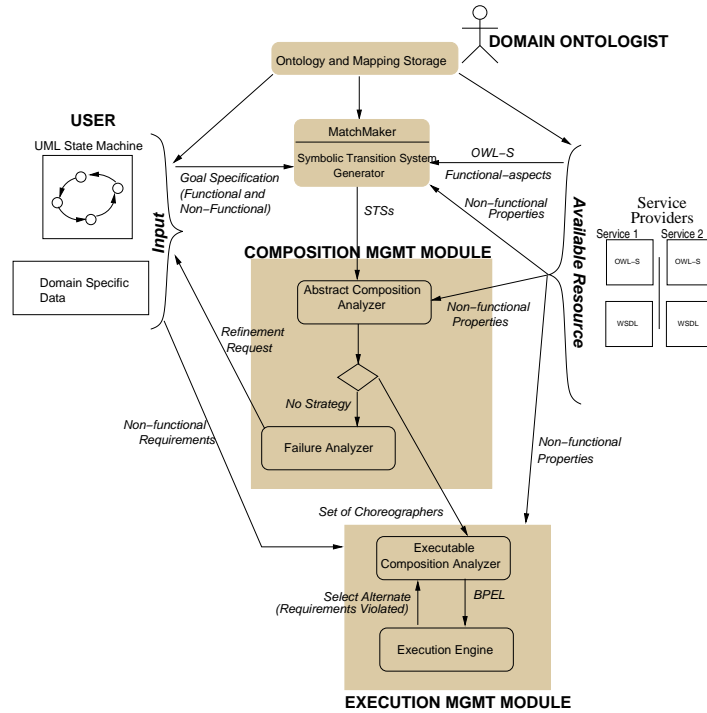


Fig. 1. MoSCoE Architectural Diagram

realizes the goal service. However, in the event that the composition cannot be realized, the system identifies the cause(s) for the failure and provides that information to the developer for appropriate reformulation of the goal specification. The framework consists of two main modules: *composition management module* and *execution management module*. The former identifies feasible compositions (if any) that realize the goal, while the latter deals with the execution of the composite service. We describe these modules in the following.

2.1 Composition Management Module

Given the STS representations of a set of N component services $\{STS_1, STS_2, \dots, STS_N\}$ and a desired goal STS_G , service composition amounts to identifying a subset of component services, which when composed with a choreographer (to be generated) STS_{cr} , realize the goal service STS_G . The role of the choreographer is to replicate input/output actions of the user as specified by the goal and to act as a message-passing interface between the components and between the component(s) and the client. It is not capable of providing any functionality (e.g., credit card processing) on its own; these are provided only by the components. The algorithm for generating such a choreographer is discussed in [1, 2] and essentially identifies whether STS_{cr} indeed realizes STS_G using the notion of *simulation* and *bisimulation* equivalence. Informally, STS_1 is said to be simulation equivalent to STS_2 , denoted by $STS_1 \sim STS_2$, if every transition sequence from the start state in the former is also present in the latter. A symmetric simulation relation, denoted by \approx , leads to a stronger equivalence referred to as bisimulation.

However, the algorithm proposed in [1, 2] suffers from the state-space explosion problem since the number of ways the components can be composed is exponential to the number of component states. This becomes a challenge with the increasing size of the search space of available component services. Hence, to address this limitation, we consider non-functional aspects (e.g., QoS) to winnow components (thereby reducing the search space) and compositions that are functionally equivalent to the goal, but violate the non-functional requirements desired by the user [3]. The non-functional requirements are quantified using *thresholds*, where a composition is said to conform to a non-functional requirement if it is below or above the corresponding threshold, as the case may be. For example, for a non-functional requirement involving the *cost* of a service composition, the threshold may provide an *upper-bound* (maximum allowable *cost*) while for requirements involving *reliability*, the threshold usually describes a *lower-bound* (minimum tolerable *reliability*). If more than one “feasible composition” meets the goal specifications, our algorithm generates all such compositions and ranks them. It is left to the user’s discretion to select the best composition according to the requirements.

In the event that a composition as outlined above cannot be realized using the available component services, the composition management module provides feedback to the user regarding the cause of the failure. The feedback may contain information about the function names and/or pre-/post-conditions required by the desired service that are not supplied by any of the component services. Such information can help to identify specific states in the state machine description of the goal service. In essence, the module identifies all un-matched transitions along with the corresponding goal STS states. Additionally, the failure of composition could be also due to non-compliance of non-functional requirements specified by the user. When such a situation arises, the system identifies those requirements that cannot be satisfied using the available components, and provides this information to the service developer for appropriate reformulation of the goal specification. This process can be iterated until a realizable composition is obtained or the developer decides to abort.

2.2 Execution Management Module

The result from the composition management module is a set of feasible compositions each defining a choreographer that will enable interaction between the client and the component services. The execution management module considers non-functional requirements (e.g., *performance*, *cost*) of the goal (provided by the user) and analyzes each feasible composition. It selects a composition that meets all the non-functional requirements of the goal, generates executable BPEL code, and invokes the MoSCoE execution engine. This engine is also responsible for monitoring the execution, recording violation of any requirement of the goal service at runtime. In the event a violation occurs, the engine tries to select an alternate feasible composition. Furthermore, during execution, the engine refers to the pre-defined set of inter-ontology mappings to carry out various data and control flow transformations [5, 6].

3 Related Work

Several efforts including those based on AI planning techniques, logic programming and automata-theory, have led to the development of platforms to support composition and deployment of services. Berardi et al. [11] proposed the Colombo framework for representing services as labeled transition systems and defined composition semantics via message passing. Pistore et al. [9, 12] developed an approach for translation of BPEL and OWL-S code into transition systems and applied planning via symbolic model checking technique to do composition. Similarly, KarmaSIM [13] translates

OWL-S descriptions into petri nets to automate tasks such as simulation, validation, composition and performance analysis. Sycara et al. [14] proposed an approach for automatic discovery, interaction and composition of semantic Web services using hierarchical task network planning. More recently, the authors in [15] developed an architecture for integrating semantic Web services by automatically generating a BPEL workflow using planning-based techniques.

Several techniques have also been developed which consider non-functional requirements for service composition. Cardoso et al. [16] describe a model that allows prediction of quality of service for workflows based on individual QoS attributes for the component services. Their technique allows compensation of composition deficiency if many services with compatible functions exist. Zheng and Benatallah [17] consider service selection task as a global optimization problem and apply linear programming to find solution that represents service composition optimizing a target function, where the function is defined as a combination of multiple non-functional parameters. Yu and Lin [18] modeled the service selection as a complex multi-choice multi-dimension 0-1 knapsack problem, which takes into consideration difference in QoS parameters offered by multiple services by assigning weights.

MoSCoE builds on the approaches mentioned above and aims to provide a model-driven framework for (semi-)automatically composing Web services. However, there are certain aspects of MoSCoE which are inherently different from others. Firstly, our approach has the ability to work with abstract (and possibly incomplete) goal service specifications for realizing composite services. These specifications can be iteratively reformulated until the required functionality is achieved. Such a technique reduces the cognitive burden on the developer to begin with complete (and perhaps complex) composition model. Secondly, in the event of failure of composition, MoSCoE can identify the cause(s) for the failure and provide such information to the developer, which can be used for appropriate refinement of the goal specification. Thirdly, MoSCoE allows users to specify the non-functional aspects of the composite service which are used to select and compose individual components that realize the goal. Furthermore, these requirements are monitored during execution of the composite service to ensure compliance. Finally, MoSCoE supports specification of semantic correspondences between multiple Web service ontologies—an essential element for integration of autonomous and heterogeneous software entities.

4 Conclusion and Further Work

In this paper we introduce MoSCoE, a novel approach for automatically developing composite services by the applying the techniques of abstraction, composition and reformulation in an incremental fashion. The framework provides a goal-directed approach to service composition and adopts a symbolic transition system-based approach for computing feasible compositions. Our formalism allows us to identify and validate all possible compositions that meet the user-specified functional and non-functional requirements. However, in the event that a composition cannot be realized using the existing set of candidate services, the technique determines the cause(s) for the failure (due to violation of functional and/or non-functional requirements), and assists the user in reformulation of those requirements in the goal specification. Furthermore, by applying ontologies and inter-ontology mappings to ground service descriptions, MoSCoE provides a mechanism for bridging semantic gaps between independently developed components. MoSCoE can be used to generate potential workflows by reusing existing Web services in various domains such as power systems [7].

Our on-going work is aimed at developing heuristics for hierarchically arranging failure-causes to reduce the number of refinement steps typically performed by the user to realize a feasible composition. We also plan to explore approaches to reducing the number of candidate compositions that need to be examined e.g., by exploiting domain specific information to impose a partial order over the available services. Additionally, our framework for modeling service composition and execution has focused on services which demonstrate a deterministic behavior without loops. Handling nondeterministic behavior that often characterizes real-world services is an important area of ongoing research. Furthermore, we plan to do a systematic evaluation of scalability and efficiency of our approach on a broad class of benchmark service composition problems.

Acknowledgment. The author would like to express his sincere gratitude to Vasant Honavar (thesis advisor), Samik Basu and Robyn Lutz for their guidance and collaboration in this research. This work has been supported in part by NSF grant 0219699 and ISU Center for CILD (<http://www.cild.iastate.edu>).

References

1. Pathak, J., Basu, S., Lutz, R., Honavar, V.: Selecting and Composing Web Services through Iterative Reformulation of Functional Specifications. In: ICTAI. (2006)
2. Pathak, J., Basu, S., Lutz, R., Honavar, V.: Parallel Web Service Composition in MoSCoE: A Choreography-based Approach. In: ECOWS. (2006)
3. Pathak, J., Basu, S., Honavar, V.: Modeling Web Services by Iterative Reformulation of Functional and Non-Functional Requirements. In: ICSOC. (2006)
4. Pathak, J., Koul, N., Caragea, D., Honavar, V.: A Framework for Semantic Web Services Discovery. In: WIDM. (2005)
5. Pathak, J., Caragea, D., Honavar, V.: Ontology-Extended Component-Based Workflows - A Framework for Constructing Complex Workflows from Semantically Heterogeneous Software Components. In: SWDB. (2004)
6. Caragea, D., Pathak, J., Bao, J., Silvescu, A., Andorf, C., Dobbs, D., Honavar, V.: Information Integration and Knowledge Acquisition from Semantically Heterogeneous Biological Data Sources. In: DILS. (2005)
7. Pathak, J., Li, Y., Honavar, V., McCalley, J.: A Service-Oriented Architecture for Electric Power System Asset Management. In: WESOA. (2006)
8. Pistore, M., Traverso, P., Bertoli, P.: Automated Composition of Web Services by Planning in Asynchronous Domains. In: ICAPS. (2005)
9. Traverso, P., Pistore, M.: Automated Composition of Semantic Web Services into Executable Processes. In: ISWC. (2004)
10. Duric, D.: MDA-based Ontology Infrastructure. *Computer Science and Information Systems* **1**(1) (2004) 91–116
11. Berardi, D., Calvanese, D., Giuseppe, D.G., Hull, R., Mecella, M.: Automatic Composition of Transition-based Semantic Web Services with Messaging. In: VLDB. (2005)
12. Pistore, M., Traverso, P., Bertoli, P., Marconi, A.: Automated Synthesis of Composite BPEL4WS Web Services. In: ICWS. (2005)
13. Narayanan, S., McIlraith, S.: Simulation, Verification and Automated Composition of Web Services. In: WWW. (2002)
14. Sycara, K., Paolucci, M., Ankolekar, A., Srinivasan, N.: Automated Discovery, Interaction and Composition of Semantic Web Services. *Journal of Web Semantics* **1**(1) (2003) 27–46
15. Agarwal, V., Dasgupta, K., et al.: A Service Creation Environment Based on End to End Composition of Web Services. In: WWW. (2005)
16. Cardoso, J., Sheth, A., Miller, J., et al.: Quality of Service for Workflows and Web Service Processes. *Journal of Web Semantics* **1**(3) (2004) 281–309
17. Zeng, L., Benatallah, B.: QoS-Aware Middleware for Web Services Composition. *IEEE Transactions on Software Engineering* **30**(5) (2004) 311–327
18. Yu, T., Lin, K.: Service Selection Algorithms for Composing Complex Services with Multiple QoS Constraints. In: ICSOC. (2005)