MoSCoE: A Framework for Modeling Web Service Composition and Execution

Jyotishman Pathak[1,2] Samik Basu[1] Robyn Lutz[1,3] Vasant Honavar[1,2]

[1]Department of Computer Science, Iowa State University, Ames IA 50011

[2]Center for Computational Intelligence, Learning & Discovery, Iowa State University, Ames IA 50011

[3]Jet Propulsion Lab/California Institute of Technology, Pasadena CA 91109

{jpathak, sbasu, rlutz, honavar}@cs.iastate.edu

Abstract

Development of sound approaches and software tools for specification, assembly, and deployment of composite Web services from independently developed components promises to enhance collaborative software design and reuse. In this context, the proposed research introduces a new incremental approach to service composition, MoSCoE (Modeling Web Service Composition and Execution), based on the three steps of abstraction, composition and refinement. Abstraction refers to the high-level description of the service desired (goal) by the user, which drives the identification of an appropriate composition strategy. In the event that such a composition is not realizable, MoSCoE guides the user through successive refinements of the specification towards a realizable goal service that meets the user requirements.

1 Introduction

Recent advances in networks, information and computation grids, and WWW have resulted in the proliferation of a multitude of physically distributed and autonomously developed software components or services [2]. The ability to construct and deploy complex workflows of composite Web services by leveraging such component repositories would lead to substantial gains in productivity in several application domains including e-Enterprise, e-Business, e-Government, and e-Science. In this context, many research and industrial efforts [11] have focused on various aspects of Web service composition ranging from service discovery, to service specification and deployment of composite services. However, despite the recent progress, the current state of the art in developing composite services has several limitations:

- For specifying the functional requirements of a composite service, the user is responsible for understanding and using service specification languages like OWL-S [12, 13, 26] and BPEL4WS [3, 20], or even complicated labeled transition systems [5] which requires more detailed understanding about the behavior of the desired service. This makes the process of service composition tedious and error-prone. As a result, there is a need for developing approaches which allow specifying composition requirements using high-level languages that are intuitive and easy to understand.

- The existing approaches provide "single-step request-response" paradigm for Web service composition. In other words, a user submits a goal specification to a composition analyzer, which tries to find an appropriate strategy for realizing the composite service. In the event, such a composition is unrealizable, the whole process fails. As opposed to this, we claim that, there should be provision for iteratively refining the goal specification in an intuitive, yet efficient way, to build composite services.

- 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 functionality of the services, are inevitable. We believe that frameworks for assembling complex Web services from independently developed component services should provide support for bridging such semantic gaps.

- Available services as well as user requirements change over time. Thus, environments for service composition need to support rapid re-design and re-deployment of services, through appropriate reuse of parts of previously assembled services or incorporation of new components.

Motivated by these concerns, the proposed research develops a new service composition framework, MoSCoE (*Mo*deling Web *S*ervice *Co*mposition and *E*xecution) to overcome some of the above mentioned limitations. Specifically, our work is aimed at:

- Developing a model-driven iterative refinement and incremental approach to Web service composition using high-level description languages such as UML state machines [8].
- Grounding the Web service composition framework within an automata-theoretic [10] setup, and using formal verification methods to guarantee the soundness and completeness (with certain restrictions) of the composition process.
- Building on current approaches [13] for associating semantics to Web services using ontologies and techniques for specifying mappings between ontologies.
- Demonstrating the feasibility of our approach with real-world applications in bioinformatics [27] and electrical power systems [17].

The rest of this paper is organized as follows: Section 2 discusses related work in service composition, Section 3 talks about our contributions and results accomplished so far, Section 4 introduces our current work in developing MoSCoE, and finally Section 5 ends with conclusions and directions for further work.

2 Related Work

Several efforts have led to the development of platforms and languages to support composition and deployment of services [11]. In particular, BPEL4WS [3] has emerged as an industry standard for representing service compositions where flow of a process and bindings between services are known a priori and specified manually. However, manual composition is cumbersome and often error-prone. To this end, many (semi-)automatic composition approaches have been proposed. Berardi et al. [6] discussed the application of Mealy machines for service representation and reduced the problem of composition to satisfiability of deterministic propositional dynamic logic. The technique was later extended in [5] to a framework for representing services as labeled transition systems and defined composition semantics via message passing. Pistore et al. [20, 26] developed a framework for translation of BPEL and OWL-S code into transition systems and applied planning via symbolic model checking technique to do composition. Similarly, KarmaSIM [14] translates OWL-S descriptions into petri nets to automate tasks such as simulation, validation, composition and performance analysis. Sycara et al. [25] proposed an approach for automatic discovery, interaction and composition of semantic Web services using hierarchical task network planning. A similar approach for composing semantic Web services using hierarchical task network planner, SHOP2, was proposed in [22]. More recently, the authors in [1] developed an architecture for integrating semantic Web services by automatically generating a BPEL workflow using planning-based techniques.

A number of model-driven approaches for service composition have also been developed [19] where standard UML tools are used to provide a higher level of abstraction of the desired composite service. SELF-SERV [4] proposed using state-charts for modeling composite services, which are then declaratively composed and executed in a dynamic peer-to-peer environment. Skogan et al. [23] use UML for capturing composite Web service patterns. These patterns, expressed in UML activity diagrams with additional extensions, specify the control flow between the individual services realizing the composite service.

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 extends the traditional notion of finite state automata by defining its language over alphabets that include functions with pre- and post-conditions over infinite-domain variables. This extension allows us to develop a formalism for determining a feasible composition, and at the same item be generic to be reduced to solve a similar problem using transition systems [5, 20]. Secondly, MoSCoE adopts an iterative refinement and incremental paradigm for service composition. The automata-theoretic approach of MoSCoE allows it to identify specific parts of the goal specification which cannot be realized using the pre-existing services. Using this failure-cause information, MoSCoE offers guidance to users for iteratively refining the goal. Thirdly, MoSCoE allows users to specify the non-functional aspects of the composite service which are used to select 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 services.

3 Prior Research

Our proposed approach (Section 4) for developing a framework for Web services composition and execution builds on our work on three related topics: workflow composition, service discovery, and inter-ontology mappings. We describe them briefly as follows:

Workflow Composition: In [16], we introduced the notion of ontology-extended workflow components and proposed semantically consistent methods for manually assembling such components into complex ontology-extended component-based workflows. Our approach relied on making explicit the typically implicit ontologies underlying autonomously developed components and specifying semantic correspondences between them. We defined a workflow schema describing the components of the workflow and the characteristics of the environment in which the workflow will be executed capturing its functional (task to be performed), informational (data flow), and behavioral (control flow) aspects.

Service Discovery: In [18], we developed a framework for semantic Web services discovery to assist composition of complex workflows. The proposed approach relied on user-supplied, context-specific mappings from an user ontology to relevant domain ontologies used to specify Web services (based on OWL-S [12]). We showed how user's query for a Web service that meets certain selection criteria (in terms of functional and non-functional aspects) can be transformed into queries processable by a matchmaking engine that is aware of the relevant domain ontologies and Web services.

Ontology Mappings: Specifying inter-ontology semantic correspondences is an important aspect for both service discovery [18] and composition [16]. To address this need, we have developed INDUS (Intelligent Data Understanding System) [7], which provides a user-friendly editor for editing Web service ontologies (describing properties and capabilities of the services) and for specifying relevant mappings between the ontologies. In addition, INDUS provides basic reasoning capability to verify the semantic consistency of these ontologies and mappings.

4 Current Work

Our current work seeks to develop a new framework for (semi)-automatically realizing new services from pre-existing ones (Figure 1). Composition in MoSCoE is based on the three steps of *abstraction, composition* and *refinement*. In MoSCoE, the user provides a high-level specification of the desired service G using UML state machines [8], a visual paradigm for representing dynamics of software systems. State machines are intuitive, have formal semantics and are relatively easy to use and understand. They provide the sequence of functions and relationships required to attain the goal service. MoSCoE translates this goal specification (by flattening the state machine) into a Finite State Automata [10] (FSA), A_G, which reveals the exact sequence in which G evolves. In addition, state machines in MoSCoE are semantically annotated by the client using appropriate domain ontologies from a repository. This is achieved by importing OWL ontologies into a UML model [9]. MoSCoE assumes that these ontologies (and mappings between them) are specified by a domain expert using existing tools such as INDUS [7]. The user also provides non-functional requirements such as performance and reliability criteria that need to be satisfied by the composite service. The service providers in MoSCoE publish their component services by providing OWL-S and WSDL specifications. In particular, given n component services, $C_1, \cdots, C_n$, their OWL-S process models are translated by MoSCoE into corresponding FSAs, $A_1, \cdots, A_n$.

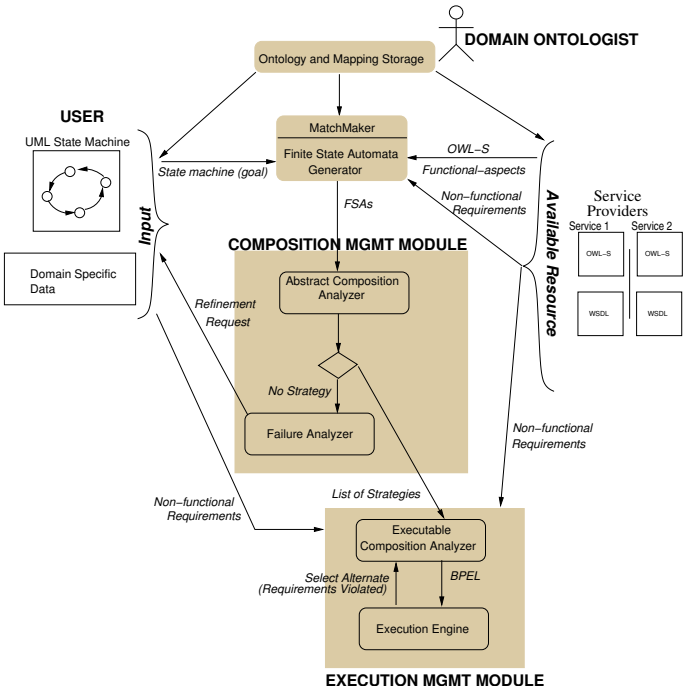

Figure 1. MoSCoE Architectural Diagram

MoSCoE manipulates these input data (user-developed service specification and published component service descriptions) and automatically identifies a composition that realizes the goal service. Because users provide high-level specification of the compos-

ite service which may not be realizable using the published component services, MoSCoE guides the users for iterative refinement of the goal service specification. The framework consists of two main modules: *composition management module* and *execution management module*. The former identifies candidate composition strategies that realize the goal, while the latter deals with actual execution of the composite service.

Composition Management Module. Given a FSA A_G for the goal service and a set of component FSAs $\mathcal{A} = \{A_1, \cdots, A_n\}$ for the existing services, service composition in MoSCoE amounts to realization of a composition strategy $\mathcal{S}$ (or a set thereof). This strategy defines the sequence in which component service FSAs should be composed such that the resultant automata accepts a non-empty sub-language of A_G, in essence, leading it to one of its final state. The details for determining a composition strategy are discussed in [15].

In the event the strategy $\mathcal{S}$ is unrealizable either due to incomplete specification about function names or pre-/post-conditions required by the desired service, but not satisfied by the component services, MoSCoE invokes its failure analyzer sub-module. This module supports iterative refinement of the abstract goal specification and provides feedback to the user regarding the cause of the failure in an intuitive, yet efficient manner [15]. 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 FSA states. We refer to them as *failure transitions* and *failure states*, respectively. The failure states of the goal FSA are mapped back to the corresponding states in the state machine and the user is asked to appropriately refine them to allow matching of failure transitions. This process is iterated until a realizable composition strategy is obtained or the user decides to abort. After each such refinement, the FSA translator is invoked again to generate FSA from the refined state machine description. This translation will be done incrementally and locally from the FSA generated in the previous iteration, and is based on incremental evaluation techniques for declarative languages [21] and automata-theoretic model checking [24].

In order to determine a feasible composition strategy, the composition management module relies on two translators: a state machine to FSA translator and a OWL-S process specification to FSA translator to obtain the automata representation of the corresponding goal and component services, respectively. Translation of state machines to finite state automata is shown in [15], and translation of the OWL-S process specifications to FSA is carried out using techniques proposed in [14, 26].

Execution Management Module. The result from the composition management module is a set of strategies each defining a sequence in which component services can be invoked to realize a specified goal. The execution management module considers non-functional requirements (e.g., performance, cost) of the goal (provided by the user) and analyzes each composition strategy. It selects a strategy that meets all the non-functional requirements of the goal, generates executable BPEL4WS 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 composition strategy. Furthermore, during execution of the strategy, the engine refers to the pre-defined set of inter-ontology mappings to carry out various data and control flow transformations [7, 16].

5 Conclusion and Further Work

In this paper we introduce MoSCoE, a new model-driven paradigm for incremental composition of Web services. MoSCoE extends the current approaches for Web service modeling and introduces a technique for iterative refinement and incremental development of composite services in a dynamic environment. At its core, MoSCoE uses UML state machines for visually representing composite services, and allows service providers to publish their services using standard service description languages such as OWL-S. MoSCoE translates these specifications into finite state automata to determine feasible composition strategies that are sound and provably correct. As more languages for service descriptions are standardized, appropriate translators can be developed to generate FSAs. 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's service creation environment can be used to generate potential workflows for achieving the desired functionality by reusing existing Web services in various domains such as Bioinformatics [27].

In the future, we plan to extend MoSCoE to incorporate additional features such as failure handling, QoS

management, and an interactive visual environment for testing and debugging composite services.

References

[1] V. Agarwal, K. Dasgupta, and et al. A Service Creation Environment Based on End to End Composition of Web Services. In *14th Intl. Conference on World Wide Web*, pages 128–137. ACM Press, 2005.

[2] G. Alonso, F. Casati, H. Kuna, and V. Machiraju. *Web Services: Concepts, Architectures and Applications*. Springer-Verlag, 2004.

[3] T. Andrews, F. Curbera, and et al. Business Process Execution Language for Web Services, Version 1.1. In *http://www.ibm.com/developerworks/library/ws-bpel/*, 2003.

[4] B. Benatallah, Q. Sheng, and M. Dumas. The Self-Serv Environment for Web Services Composition. *IEEE Internet Computing*, 7(1):40–48, 2003.

[5] D. Berardi, D. Calvanese, D. G. Giuseppe, R. Hull, and M. Mecella. Automatic Composition of Transition-based Semantic Web Services with Messaging. In *31st Intl. Conference on Very Large Databases*, pages 613–624, 2005.

[6] D. Berardi, D. Calvanese, D. G. Giuseppe, M. Lenzerini, and M. Mecella. Automatic Composition of e-Services that Export their Behavior. In *1st Intl. Conference on Service Oriented Computing*, pages 43–58, 2003.

[7] D. Caragea, J. Pathak, J. Bao, A. Silvescu, C. Andorf, D. Dobbs, and V. Honavar. Information Integration and Knowledge Acquisition from Semantically Heterogeneous Biological Data Sources. In *2nd Intl. Workshop on Data Integration in Life Sciences*, pages 175–190. Springer-Verlag, 2005.

[8] M. Crane and J. Dingel. On the Semantics of UML State Machines: Categorization and Comparision. In *Technical Report 2005-501, School of Computing, Queen's University, Canada*, 2005.

[9] D. Duric. MDA-based Ontology Infrastructure. *Computer Science and Information Systems*, 1(1):91–116, 2004.

[10] J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages and Computation*. Addison-Welsey, 1979.

[11] R. Hull and J. Su. Tools for Composite Web Services: A Short Overview. *SIGMOD Record*, 34(2):86–95, 2005.

[12] D. Martin, M. Burstein, J. Hobbs, and et al. OWL-S: Semantic Markup for Web Services, Version 1.1. In *http://www.daml.org/services/owl-s*, 2004.

[13] S. McIlraith, T. Son, and H. Zeng. Semantic Web Services. *IEEE Intelligent Systems*, 16(2):46–53, 2001.

[14] S. Narayanan and S. McIlraith. Simulation, Verification and Automated Composition of Web Services. In *11th Intl. World Wide Web Conference*, pages 77–88. ACM Press, 2002.

[15] J. Pathak, S. Basu, R. Lutz, and V. Honavar. An Automata-theoretic Approach to Service Development Using Abstraction, Composition and Refinement. In *Submitted to IEEE 4th International Conference on Web Services*, Under Review.

[16] J. Pathak, D. Caragea, and V. Honavar. Ontology-Extended Component-Based Workflows - A Framework for Constructing Complex Workflows from Semantically Heterogeneous Software Components. In *2nd Intl. Workshop on Semantic Web and Databases*, pages 41–56. Springer-Verlag, 2004.

[17] J. Pathak, Y. Jiang, V. Honavar, and J. McCalley. Condition Data Aggregation with Application to Failure Rate Calculation of Power Transformers. In *39th Annual Hawaii Intl. Conference on System Sciences*. IEEE Press, 2006.

[18] J. Pathak, N. Koul, D. Caragea, and V. Honavar. A Framework for Semantic Web Services Discovery. In *7th ACM Intl. Workshop on Web Information and Data Management*, pages 45–50. ACM press, 2005.

[19] K. Pfadenhauer, S. Dustdar, and B. Kittl. Challenges and Solutions for Model Driven Web Service Composition. In *14th IEEE Intl. Workshop on Enabling Technologies: Infrastructures for Collaborative Enterprises*, pages 126–131. IEEE Press, 2005.

[20] M. Pistore, P. Traverso, P. Bertoli, and A. Marconi. Automated Synthesis of Composite BPEL4WS Web Services. In *3rd Intl. Conference on Web Services*, pages 293–301. IEEE Press, 2005.

[21] D. Saha and C. R. Ramakrishnan. Incremental Evaluation of Tabled Logic Programs. In *International Conference on Logic Programming*, volume 2916, pages 389–406. Springer-Verlag, 2003.

[22] E. Sirin, B. Parsia, D. Wu, J. Hendler, and D. Nau. HTN Planning for Web Service Composition using SHOP. *Journal of Web Semantics*, 1(4):377–396, 2004.

[23] D. Skogan, R. Grønmo, and I. Solheim. Web Service Composition in UML. In *8th IEEE Intl. Enterprise Distributed Object Computing Conference*, pages 47–57. IEEE Press, 2004.

[24] O. Sokolsky and S. A. Smolka. Incremental Model Checking in the Modal Mu-Calculus. In *Proceedings of the 6th International Conference on Computer Aided Verification*, pages 351–363. Springer-Verlag, 1994.

[25] K. Sycara, M. Paolucci, A. Ankolekar, and N. Srinivasan. Automated Discovery, Interaction and Composition of Semantic Web Services. *Journal of Web Semantics*, 1(1):27–46, 2003.

[26] P. Traverso and M. Pistore. Automated Composition of Semantic Web Services into Executable Processes. In *3rd Intl. Semantic Web Conference*, pages 380–394. Springer-Verlag, 2004.

[27] F. Wu, J. Pathak, C. Yan, D. Dobbs, and V. Honavar. PPID: A Database of Protein-Protein Interface. In *To be submitted to BMC Bioinformatics*, 2006.