Assembling Composite Web Services from Autonomous Components

Jyotishman PATHAK, Samik BASU and Vasant HONAVAR

Department of Computer Science
Iowa State University
Ames, IA 50011-1040, USA
{jpathak, sbasu, honavar}@cs.iastate.edu

Abstract.
   Web services are fast emerging as the technology of choice to build distributed information systems in multiple domains including e-Business and e-Science. An important challenge is to develop methodologies and tools that enable (semi-) automatic composition of services by taking into account the functional, non-functional and behavioral requirements of the service developer. This paper presents the fundamental concepts and issues related to service composition and provides a representative sample of existing work proposed by the AI planning and formal methods communities to address some of the challenges in service composition. It also provides a brief introduction to an iterative and incremental technique for modeling composite Web services proposed by the authors.

Keywords. Web Services, Composition, AI Planning, Formal Methods

1. Introduction

Recent advances in networks, information and computation grids, and the World Wide Web has led to the emergence of a new approach to build highly distributed information systems using Web-based services[1] [1]. They hold the promise to enable development of rich and flexible applications in multiple domains including e-Business and e-Science owing to their loosely coupled and interoperable nature. Consequently, there has been a significant interest in recent years to build Service-Oriented Architectures [2,3] that support the creation and deployment of complex Web services to accomplish different tasks. In particular, the ability to discover and integrate existing services into a composite service (a.k.a. *Web Service Composition*) has received a lot of attention from both academia and industry, and many techniques based on formal methods, AI planning and logic theory have been proposed. The main objective of these approaches is to allow service developers to flexibly locate the component services needed, compose them and orchestrate their execution to achieve the desired requirements, which otherwise cannot be fulfilled by a single (available) service.

---

[1] In this paper, we use the terms "Web services" and "services" interchangeably.

However, developing tools and techniques for service composition is non-trivial due to many inherent challenges. These include: How to search for the most suitable set of services that when composed appropriately will satisfy the desired requirements? How to model expressive description languages for representing services and service compositions? How to validate and verify the behavioral and executional properties of the composite service? How to enable execution monitoring, repair and adaptation of service compositions? How to build user-friendly and intuitive tools for modeling real-world complex services? In other words, automatic composition of Web services require addressing various challenges related to Web service discovery, integration, orchestration, verification and execution monitoring. In addition, to ensure that the proposed techniques can be applied in practical settings, their efficiency, scalability and usability is of great significance.

Against this background, in this paper we discuss few representative approaches to Web service composition that attempt to address some of the challenges outlined above. In particular, we focus on approaches that rely on techniques developed primarily by the AI planning and formal methods communities (Section 2). Even though the specific research topics addressed by these two areas may vary, we have noticed a strong parallelism between the Web service composition techniques based on them. Consequently, we have explored such a synergy to propose a novel framework for iterative and incremental modeling composite Web services which we discuss briefly in this paper (Section 3). Finally, we conclude the paper by outlining some of the open research problems (Section 4) that we believe have to be addressed effectively for development of robust, efficient and practical tools and techniques for Web service composition.

2. State-of-the-art in Web Service Composition: A Short Survey

2.1. What is Web Service Composition?

Web services are software system designed to support interoperable machine-to-machine interaction over a network [4]. Typically, they have an interface described in a machine-processable format which specify their functionalities that can be invoked by other systems through message-based interactions. However, in certain cases a desired functional (and/or non-functional) requirement cannot be met by a single Web service in its entirety, but could be possibly met by appropriately integrating and composing a set of available services. Informally, given a user (goal) requirement G and a set of available Web services $W = \{W_1, W_2, \ldots, W_n\}$, Web service composition amounts to: (i) generating a new composite Web service W_G (Figure 1(a)) by suitably combining a subset of available services, $W_i \ldots \otimes \ldots W_j \ldots \otimes \ldots W_k$, where $\otimes$ is the composition operator, or (ii) establishing a linkage structure $L = \{L_{ij}, L_{ik}, \ldots\}$ (Figure 1(b)) between the participating services, $W_i, \ldots W_j, \ldots, W_k$, that will allow them to communicate directly via message exchanges. The former approach, commonly referred to as the *mediator-based composition*, results in developing a mediator which will enable communication between the client (either a human being or a software agent) and the participating services. Each and every message exchange between the client and the services is channeled through the mediator. Whereas, for the latter approach, commonly referred to as the *choreography-based composition*, the message exchange channels or links are established between the
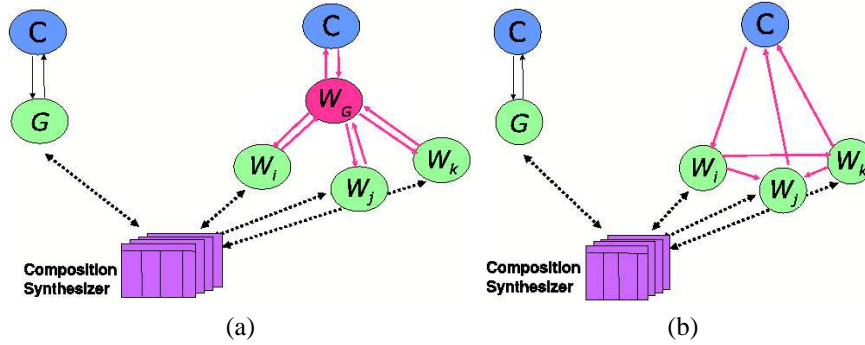
Figure 1. Two different types of composition [8]: (a) Mediator-based (b) Choreography-based

participating services themselves. Thus, the client communicates directly with the respective services.

Many techniques have been proposed in the literature (see [5,6,7] for surveys) for both the approaches in the recent years. In particular, there has been a significant interest from the researchers in the AI planning and formal methods communities to develop techniques for service composition that leverage and build on the existing work. We discuss a representative sample of such approaches in the following sections.

2.2. AI Planning for Web Service Composition

In general, AI planning can be regarded as an area of study that is concerned with automatic generation of plans that will be able to solve a problem within a particular domain. Typically, a plan consists of sequence of actions, such that given an initial state or a condition, a planner will suitably select a set of actions which, when executed according to the generated plan, will satisfy certain goal conditions. In the context of Web services, a planning domain can be represented by a sextuplet $(W, S, A, \longrightarrow, s_0, s_G)$, where W is the set of available Web services, S is the set of all possible states of these services (world), A is the set of actions/functions provided by the services that the planner can perform in attempting to change the state from one to another in the world, $\longrightarrow \subseteq S \times A \times S$ is the set of state transitions which denote the precondition and effects for execution of each action, and finally $s_0 \in S$ and $s_G \in S$ are the initial and goal states, respectively, specified in the requirement of the Web service requesters to indicate that the plan initiates its execution starting from state s_0 and terminates at state s_G. Given this domain, researchers have proposed multiple approaches using a variety of planning techniques that will generate a plan for realizing the goal requirements.

PDDL [9] (Planning Domain Definition Language) is one of the very widely known description languages in the planning domain and has influenced the development of Web service description languages such as OWL-S [10] (Web Ontology Language for Services). McDermott [11] extended PDDL by introducing the notion of 'value of the action', essentially representing certain information that is created or learned as a consequence of executing a particular action. The main intention of introducing this extension was to have the ability to capture the information and the content of messages that are exchanged between the services. The work demonstrates how this extended language can be used with estimated regression planners to create conditional plans that achieve the desired goal. Medjahed et al. [12] applied a rule-based planning technique for finding

feasible compositions and introduced a declarative language for describing the goal requirements. The core of the approach comprised of developing composability rules that consider and analyze syntactic and semantic properties of the Web services to devise a plan. Such rules, for example, might specify that two Web services W_1 and W_2 are composable only if the output messages of W_1 are compatible with the input messages of W_2. Sycara et al. [13] proposed an approach for automatic discovery, interaction and composition of semantic Web services using HTN (Hierarchical Task Network) planning. Their technique represents services using DAML-S [14] (Darpa Agent Markup Language for Services, the predecessor of OWL-S), and provides multiple "degree of match" criteria to determine whether the service provider capabilities conform to the requirements of the requester. Another approach which relied on using the HTN planner for automatic composition of services described in OWL-S was proposed in [15]. The authors provide an algorithm for translating OWL-S service descriptions into SHOP2 (an HTN planner) domain and prove the correctness of their approach by showing the correspondence to the situation calculus semantics of OWL-S. SEMAPLAN [16] attempts to leverage traditional AI planning and information retrieval techniques for building a semi-automated service composition tool. The technique relies on domain-dependent/independent ontologies [17] for calculating semantic similarity scores between the concepts/terms in service descriptions, and applies this score to guide the searching process of the planning algorithm. The experimental results demonstrate that SEMAPLAN performs superior compared to the traditional planning based techniques. A similar approach is also proposed in [18] which attempts at combining traditional AI planning techniques with semantics-based approaches.

2.3. Formal Methods for Web Service Composition

Formal Methods is an area of study that provides a language for describing a software artifact (e.g., specifications, design, source code) such that formal proofs are possible, in principle, about properties of artifact so expressed. In the context of Web service composition, typically the property proved is that an implementation is functionally correct, that is, it fulfills a particular specification. In the recent past, many research efforts for service composition have adopted formal methods techniques to leverage its mathematically-precise foundation for providing theoretically sound and correct formalisms. We discuss a few of those approaches in the following paragraphs.

Pistore et al. [19,20] represent Web services using transition systems [21] that communicate via exchanging messages. Their approach relies on symbolic model checking techniques to determine a parallel composition of all the available services, and then generate a controller to control the composed services such that it satisfies the user-specified requirements. Informally, if $W = \{W_1, W_2, \ldots W_n\}$ is the set of available services, ρ is the user-specified requirement (i.e., ρ describes the goal G), and $\|$ is the composition operator, the aim is to find a "controller" W_c, such that: $W_c \rhd (W_1 \| \ldots \| W_n) \models \rho$. Similarly the Colombo framework [8] models Web services using labeled transition systems. However, this approach relies on specifying linkages to establish communication channels between services that have identical signatures, and exploits them to determine a feasible composition that satisfies the goal requirements by reducing the composition problem to satisfiability of a suitable deterministic propositional dynamic logic formula. On a slightly different note, Gwen Salaün et al. [22] apply Process Algebra [23] (PA)

to model Web services in at least two different ways: (i) at *design time*, PA can be used to describe an abstract specification of the system to be developed, which can be validated and used as a reference for implementation; (ii) by applying *reverse engineering*, existing Web service interface descriptions can be translated to PAs. Specifically, this work adopted CCS [21] as the PA and demonstrated techniques for translating BPEL [24] (Business Process Execution Language) processes into CCS, which can then be verified to reason about properties specified in temporal logic. Hamadi and Benatallah [25] apply a petri net-based algebra to model the control flow and capture semantics of complex Web service compositions. Their framework provides various control flow constructs such as sequence, alternative, iterative and arbitrary, and the authors show these constructs can be used to determine and verify a composition. However, it is unclear whether the composition is done (semi-) automatically or manually. SELF-SERVE [26] extended this work by providing the ability for dynamically composing and executing Web services represented as state charts. One of the key features of SELF-SERVE is to adopt a peer-to-peer (P2P) computing environment for executing the (composite) services, which in practice has multiple advantages (in terms of scalability, fault-tolerance etc.) compared to centralized architectures.

3. The MoSCoE Approach

3.1. Problems with Existing Web Service Composition Techniques

In the previous sections, we have outlined some of the representative approaches that apply AI planning and formal methods techniques for building (semi-) automated solutions to Web service composition. Nevertheless, these approaches have several drawbacks that limit their practical viability and wide-scale adoption. These limitations include:

- *Complexity of Modeling Composite Services:* For specifying functional requirements, the current techniques for service composition require the service developer to provide a specification of the desired behavior of the composite service (goal) in its entirety. Consequently, the developer has to deal with the cognitive burden of handling the entire composition graph (comprising appropriate data and control flows) which becomes hard 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 and updated until a feasible composition is realized.
- *Inability to Analyze 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 component services), 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. This requirement is of particular importance in light of the previous limitation because in many cases the failure to realize a goal service using a set of component services can be attributed to incompleteness of the goal specification.

- *Inconsideration 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 (available) 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.

- *Inability to Handle 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 or ontologies 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.

To overcome some of these limitations, we are working towards developing a novel Web service composition and execution framework called MoSCoE[2] [27,28,29] (Modeling Web Service Composition and Execution) that is based on three basic principles namely *abstraction, composition* and *reformulation*. By abstraction, we refer to the ability of MoSCoE that allows the users (i.e., service developers) to specify an abstract and possibly incomplete specification of the (goal) service. This specification is used to select a set of suitable component services such that their composition realizes the desired goal in terms of both functional and non-functional requirements. 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. We discuss further details about the MoSCoE framework and the system architecture in the following section.

3.2. MoSCoE Framework

Figure 2 shows the architectural diagram of the MoSCoE framework illustrated above. As mentioned, 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 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 Systems[3] (STS), where states are abstraction of the service configuration and transitions represent the way in which such configurations are updated. In addition, the STSs are semantically annotated using appropriate domain ontologies from a repository by importing OWL ontologies into the state machine model [32]. MoSCoE assumes that these ontologies (and mappings between them) are specified by a domain expert using existing tools such as INDUS [33]. The user also provides non-functional requirements (e.g., `cost,reliability`) that need to be satisfied by the goal service.

---

[2] `http://www.moscoe.org`

[3] 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 [30,31].
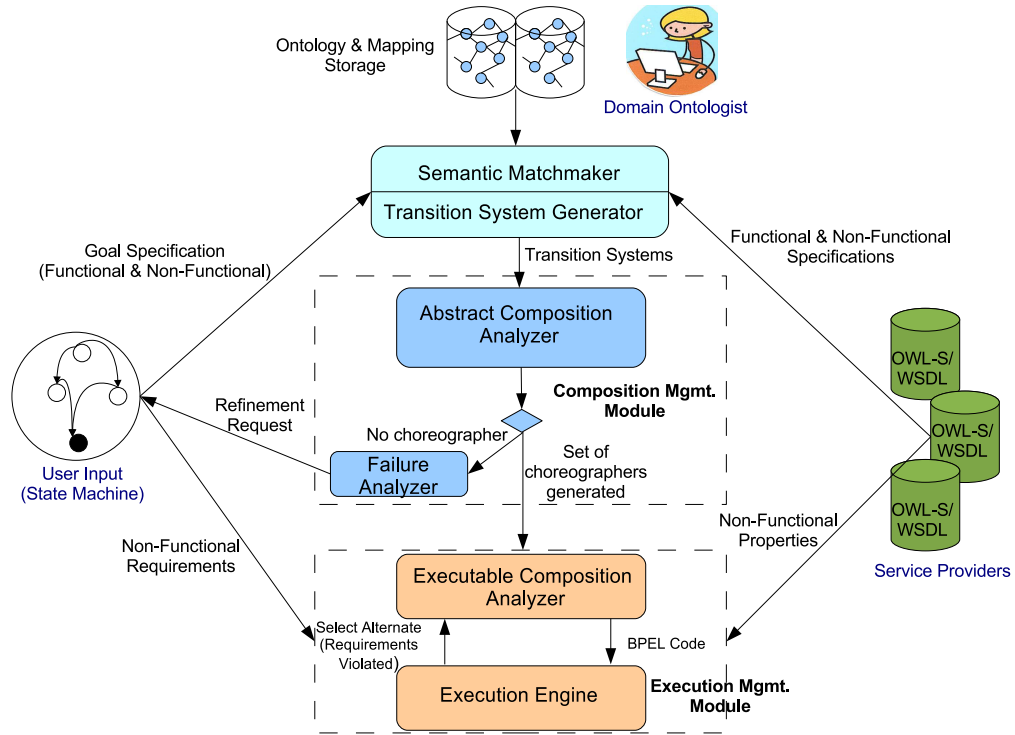
Figure 2. MoSCoE Architectural Diagram

MoSCoE manipulates these input data (user-provided service specification and published component service descriptions) and automatically identifies a composition that realizes the goal service. However, in the event that a 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 system architecture comprises 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 paragraphs.

Composition Management Module. Given the STS representations of a set of N component services $\{W_{STS_1}, W_{STS_2}, \ldots, W_{STS_N}\}$ and a desired goal W_{STS_G}, service composition in MoSCoE amounts to identifying a subset of component services, which when composed with a mediator (to be generated) W_{STS_M}, realize the goal service W_{STS_G}. The role of the mediator 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 component services. The algorithms for generating such a mediator are discussed in [28,27] and they techniques essentially identify whether W_{STS_M} realizes W_{STS_G} using the notion of *simulation* and *bisimulation* equivalence. Informally, simulation equivalence ensures that every behavioral pattern in the goal is present in the composed mediator, whereas bisimulation

equivalence is a symmetric relation which ensures that the composition offers exactly the same behavior as specified in the goal, and nothing more.

However, the algorithms proposed in [28,27] suffer from the state-space explosion problem since the number to ways the component services can be composed is exponential to the number of component service 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., Quality of Service) 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 [29,34]. 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 (both functional and non-functional requirements), our algorithm generates all such compositions and ranks them. It is then 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(s) 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.

Execution Management Module. The result from the composition management module is a set of feasible compositions each defining a mediator 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 [35].

4. Open Research Directions

In the following, we outline several research challenges that we believe have to be addressed by the research community to make existing solutions for automatic Web service composition (including MoSCoE) better and useful in practice. These issues include:

Composition Efficiency: The practical feasibility of approaches to automated service composition is ultimately limited by the computational complexity of the service composition algorithms. However, the existing composition techniques run into exponential complexities and become impractical in real-world situations comprising of hundreds, if not thousands, of services. Hence, intelligent approaches and heuristics for reducing the number of candidate compositions that need to be examined are urgently needed in order to scale up service composition techniques sufficiently to make them useful in practice.

Execution Models: Most of the existing implementations for composite Web service execution adopt a centralized architecture, that is, there exists an orchestrator (representing the composite service) in a centralized location that is responsible for coordinating and forwarding the intermediate results during the execution. Such a design has its limitation in terms of scalability, failure resiliency, and network bottlenecks. Towards this end, we believe that decentralized [36] or Peer-to-Peer (P2P) based architectures such as SELF-SERVE [26] will prove to be more beneficial in practical settings.

Failure Handling and Fault Tolerance: Web services are by nature autonomous and have an unpredictable behavior. For example, it is possible for a particular Web service W_i that is part of a composition to become inaccessible or updated (furnishing additional functions and/or removing existing ones, thereby altering its original behavior). Consequently, an existing integrated system (or a composite service) which comprises of multiple services including W_i, will require appropriate update in the form of replacing W_i. However, very limited research [37,38] has been carried out to address this issue which needs further investigation. The problem becomes even more non-trivial when the replacement of the faulty service has to be carried out, while the composite service is being executed, in such a way that it is transparent to the client.

Security: Addressing security concerns is important for any Web-based system and various researchers have proposed mechanisms for ensuring security in Web services (see [39] for a survey). However, most of techniques build a trust-based framework or assume the existence of an environment, where once a service is identified to be "good" (loosely speaking) based on its security policy etc., it is considered to be trustworthy. However, in certain cases, even though a particular service is trustworthy, it might delegate a part of its functionality to another service which cannot be trusted. For example, an online air ticket reservation service W_x might delegate the process of verifying authenticity of payment methods (e.g., credit card) required to purchase the air tickets to a third-party service provider W_y (in a manner transparent to the client), which may not follow the same security policy as W_x causing a potential security threat. Unfortunately, it is hard to detect such vulnerabilities. Furthermore, even if W_x claims to be "good", it may not strictly adhere to its own security policy, which makes it even harder to detect whether the integrity of client information has been compromised. We believe addressing these two issues is a significant and important research challenge for the Web services community.

Tool Support: An important component of making techniques for automatic Web service composition useful for masses is to develop user-friendly tools and platforms that will allow non-experts to model complex services. Towards this end, model-driven based approaches [40] has shown some promise, although a lot of research has to be carried out, in particular by leveraging techniques from human-computer interaction.

Experimental Benchmark: At present, due to lack of a benchmark (dataset) of Web services, there is no uniform way of comparing, for example, an existing service composition algorithm with another. We believe that developing a comprehensive benchmark and testbed of Web services will act as a quick aid for testing and ease of prototyping to evaluate different techniques. Such a benchmark should comprise of various hardware platforms and a variety of synthetic and real-world Web services. To the best of our knowledge, WSBen [41] is one of the preliminary efforts in this direction.

5. Concluding Remarks

We have briefly surveyed some of the existing techniques for (semi-) automatic composition of Web services that are based on AI planning and formal methods and discussed some of the limitations of these techniques. The paper demonstrates how our framework (called MoSCoE) attempts to address some of these limitations and envisions to provide a user-friendly technique for developing complex Web services. Furthermore, we have outlined potential research issues that need to be addressed by blending techniques from artificial intelligence, software engineering, networks and distributed systems in order develop solutions for Web service composition that are of practical significance and value.

Acknowledgments

References

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

[2] T. Erl. *Service-Oriented Architecture: A Field Guide to Integrating XML and Web Services*. Prentice Hall, New Jersey, 2004.

[3] D. Ferguson and M. Stockton. Service-Oriented Architecture: Programming Model and Product Architecture. *IBM Systems Journal* 44(4):753–780, 2005.

[4] D. Booth, H. Haas, F. McCabe, and et al. Web Services Architecture, W3C Working Group Note 11. *http://www.w3.org/TR/ws-arch/*, 2004.

[5] S. Dustdar and W. Schreiner. A Survey on Web Services Composition. *International Journal on Web and Grid Services* 1(1):1–30, 2005.

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

[7] J. Rao and X. Su. A Survey of Automated Web Service Composition Methods. *1st Intl. Workshop on Semantic Web Services and Web Process Composition*, pp. 43–54, 2004.

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

[9] A. Gerevini and D. Long. Preferences and Soft Constraints in PDDL3. *ICAPS Workshop on Preferences and Soft Constraints in Planning*, 2006.

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

[11] D. V. McDermott. Estimated-Regression Planning for Interactions with Web Services. *6th Intl. Conference on Artificial Intelligence Planning Systems*, pp. 204–211, 2002.

[12] B. Medjahed, A. Bouguettaya, and A. K. Elmagarmid. Composing Web services on the Semantic Web. *The Very Large Databases Journal* 12(4):333–351, 2003.

[13] 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.

[14] A. Ankolekar, M. Burstein, J. R. Hobbs, and et al. DAML-S: Semantic Markup for Web Services. *International Semantic Web Workshop*, 2001.

[15] 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.

[16] R. Akkiraju, B. Srivastava, A.-A. Ivan, R. Goodwin, and T. F. Syeda-Mahmood. SEMAPLAN: Combining Planning with Semantic Matching to Achieve Web Service Composition. *4th IEEE International Conference on Web Services*, pp. 37–44. IEEE CS Press, 2006.

[17] T. Gruber. Ontolingua: A Mechanism to Support Portable Ontologies. *Technical Report, KSL-91-66, Stanford University, Knowledge Systems Laboratory*, 1992.

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

[19] M. Pistore, A. Marconi, P. Bertoli, and P. Traverso. Automated Composition of Web Services by Planning at the Knowledge Level. *19th Intl. Joint Conferences on Artificial Intelligence*, pp. 1252–1259, 2005.

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

[21] R. Milner. *A Calculus of Communicating Systems*. Springer-Verlag New York, Inc., 1982.

[22] G. Salaün, L. Bordeaux, and M. Schaerf. Describing and Reasoning on Web Services using Process Algebra. *2nd IEEE International Conference on Web Services*, pp. 43–50. IEEE Computer Society, 2004.

[23] M. Hennessy. *Algebraic Theory of Processes*. MIT Press, 1988.

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

[25] R. Hamadi and B. Benatallah. A Petri Net-based Model for Web Service Composition. *14th Australasian Database Conference*, pp. 191–200. Australian Computer Society, Inc., 2003.

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

[27] J. Pathak, S. Basu, R. Lutz, and V. Honavar. Selecting and Composing Web Services through Iterative Reformulation of Functional Specifications. *18th IEEE International Conference on Tools with Artificial Intelligence*, pp. 445–454. IEEE CS Press, 2006.

[28] J. Pathak, S. Basu, R. Lutz, and V. Honavar. Parallel Web Service Composition in MoSCoE: A Choreography-based Approach. *4th IEEE European Conference on Web Services*, pp. 3–12. IEEE CS Press, 2006.

[29] J. Pathak, S. Basu, and V. Honavar. Modeling Web Services by Iterative Reformulation of Functional and Non-Functional Requirements. *4th International Conference on Service Oriented Computing*, pp. 314–326. LNCS 4294, Springer-Verlag, 2006.

[30] M. Pistore, P. Traverso, and P. Bertoli. Automated Composition of Web Services by Planning in Asynchronous Domains. *15th Intl. Conference on Automated Planning and Scheduling*, pp. 2–11, 2005.

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

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

[33] D. Caragea, J. Pathak, J. Bao, A. Silvescu, C. Andorf, D. Dobbs, and V. Honavar. Information Inte-

gration and Knowledge Acquisition from Semantically Heterogeneous Biological Data Sources. *2nd International Workshop on Data Integration in Life Sciences*, pp. 175–190. Springer-Verlag, 2005.

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

[35] J. Pathak, D. Caragea, and V. Honavar. Ontology-Extended Component-Based Workflows-A Framework for Constructing Complex Workflows from Semantically Heterogeneous Software Components. *2nd International Workshop on Semantic Web and Databases*, pp. 41–56. LNCS 3372, Springer-Verlag, 2004.

[36] W. Binder, I. Constantinescu, and B. Faltings. Decentralized Orchestration of Composite Web Services. *4th IEEE International Conference on Web Services*, pp. 869–876. IEEE Computer Society, 2006.

[37] B. Benatallah, F. Casati, and F. Toumani. Representing, Analysing and Managing Web Service Protocols. *Data and Knowledge Engineering* 58(3):327–357, 2006.

[38] L. Bordeaux, G. Salaün, D. Berardi, and M. Mecella. When are Two Web Services Compatible? *5th International Workshop on Technologies for E-Services*, pp. 15–28. LNCS 3324, Springer-Verlag, 2004.

[39] C. Gutiérrez, E. Fernández-Medina, and M. Piattini. A Survey of Web Services Security. *International Conference on Computational Science and Its Applications*, pp. 968–977. LNCS 3043, Springer-Verlag, 2004.

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

[41] S.-C. Oh, H. Kil, D. Lee, and S. R. T. Kumara. WSBen: A Web Services Discovery and Composition Benchmark. *4th International Conference on Web Services*, pp. 239–246. IEEE Press, 2006.