Semantic Web Services Composition

Charlie Abela

Department of Computer Science and AI,
University of Malta

Abstract. Web services are becoming the most predominant paradigm for distributed computing and electronic business. They are self-contained Internet accessible applications that are capable not only of performing business activities on their own, but they also possess the ability to engage with other Web services in order to build new value-added services. Both academic and industrial bodies have been investigating issues regarding service descriptions, discovery and invocation, but automated service composition was somewhat neglected. The latter involves automated methods for constructing a sequence of Web services to achieve a desired goal. In this work we present initial research that focuses on the issue of automated service composition in conjunction with the Semantic Web. In this report we propose a composition engine that will automatically handle the integration of Web services through the use of a Web service description language such as DAML-S, the planning of workflow definitions, scheduling of tasks, status monitoring of the execution process, handling of faults and communication with other entities such as user agents, service registries and other composition engines.

1 Introduction

Web services are a relatively new technology for building distributed web applications. The three-level architecture for web services defined by Microsoft, IBM and Ariba, includes UDDI (Universal Discovery Description Integration) [1], WSDL (Web Services Description Language) [2] and SOAP (Simple Object Access Protocol) [3].

SOAP and WSDL are designed to provide descriptions of message transport mechanisms and for describing the interface used by each service. However, neither SOAP nor WSDL allow for the automatic location of web services on the basis of their capabilities. UDDI, on the other hand provides a registry of businesses and web services. UDDI describes businesses by their physical attributes such as name, address and services that they provide. In addition, UDDI descriptions are augmented by a set of attributes, called Tmodels, which describe additional features such as classification of services within taxonomies. Since UDDI does not represent service capabilities, it is of no help to a search for services based on what they provide.

A limitation that surrounds XML based standards, such as those mentioned above is their lack of explicit semantics by which two identical XML descriptions could mean totally different things, depending on the context in which they are used. This limits the capability of matching Web services. This is important because a requester for a Web service does not know which services are available at a certain point in time and so semantic knowledge would help in the identification of the most suitable service for a particular task.

The effort to integrate semantics into Web services started with the now standardized RDF and evolved with the creation of DAML+OIL and in particular with DAML-S and OWL-S (where S stands for Services) [4]. Neither of these are W3C standards, but the work done by these working groups is considered as being very important. Infact the WOWG (Web Ontology Working Group)

[5] considered these languages as their initial step in the creation of the new Web Ontology Language called OWL. OWL is heading for standardization and the work involved the creation of two specific subsets of the language that can help implementers and language users.

As defined by the WOWG in [6], OWL Lite was designed for easy implementation and to provide users with a functional subset that will get them started in the use of OWL. OWL DL (where DL stands for "Description Logic") was designed to support the existing Description Logic business segment and to provide a language subset that has desirable computational properties for reasoning systems. The complete OWL language (called OWL Full to distinguish it from the subsets) relaxes some of the constraints on OWL DL so as to make available features which may be of use to many database and knowledge representation systems, but which violate the constraints of Description Logic reasoners.

The rest of the paper is structured as follows. Section 2 will present some background on service composition languages, how they fit in the Web services architecture and a comparison between the major players in this area. In section 3 we define the types of compositions and how these are handled by refering to some related work on composition of Web services. Section 4 will describe our proposal for a service composition engine by considering all the phases in the compostion cycle and give initial ideas how such an engine can be implemented. Finally in section 5 we discuss some open issues and conclude this paper.

2 Service Composition Langauges

In spite of all the interest shown from both industrial bodies and academic institutions, several obstacles are preventing them from harnessing efficiently the Web services technology. The current Web service model, as depicted in Figure 1 below, enables service discovery dynamically, using markup languages for describing service properties, however it does not account for automatic integration of one service with another. Extensive work has been done in the area of service discovery and matchmaking as described in [7], [8] and [9]. However, the dynamics of service composition still remains one of the most challenging aspects for researchers in academia and industry. Several ongoing Industrial initiatives in the development of service markup languages such as BPEL4WS (Business Process Execution Language for Web Services, also called BPEL for short) [10], XLANG [11], WSFL (Web Services Flow Language) [12], WSCI (Web Services Choreography Interface) [13] are aimed at service composition. These, though, have resulted in solution providing frameworks, which are targeted towards proprietary application development environments. The languages may be syntactically sound, however they lack semantics and expressiveness. Nonetheless, automated service recognition, mechanized service composition and service negotiation are still amiss from them. Service Composition also remains one of the most important goals of the Semantic Web.

DAML-S is the result of work done to ensure the integration of Semantic Web technology with Web services. It includes a Web service ontology, defined through DAML+OIL, and provides an ontology for service providers to markup their services for intelligent agents to dynamically discover and compose. However lack of adequate tool support restricts the practical adaptation of such languages. Further, assuming that a majority of these composition languages do get standardized eventually, we are looking at a Web, where independent services would be marked up with different languages. One of the problems we foresee here is interoperability between the software agents trying to compose such disparately marked up services. We would then need ontologies to establish mappings between the different languages. Such a scenario would also require compromise with performances issues in terms of delay. Therefore, it is imperative that a standard language for service composition be used. In [14] we have identified several requirements for a Semantic Web service composition language. One of these requirements regards precisely the fact that a composition

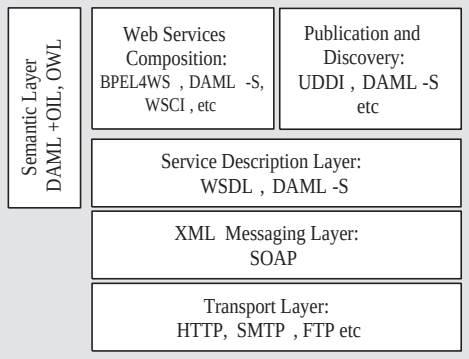

Fig. 1. Overview of present Web services technologies

language has to be adequately expressive and it has to have a well-defined semantics and a robust formal model so to facilitate the automated composition of services through software agents. Recent efforts towards some kind of standardization of these technologies have resulted in the creation of the Web Services Choreography Working Group (WSC-WG) [15].

2.1 Comparison Of Existing Languages

We have analyzed and compared the three most important efforts regarding service composition, namely, DAML-S, BPEL and WSCI. A more detailed discussion can be found in [14].

To realize the automation of service composition on the Web, a language needs to have well defined semantics along with syntactical constructs. Semantics help in defining reasoners for machine interpretation of service description. DAML-S with its base firmly rooted in Description Logics has well-established formal semantics. The process and profile model have been structured to enable intelligent agents to interpret the markup and reason about the composition. BPEL and WSCI do not expose any form of semantics and therefore do not facilitate the process of automated composition.

Expressiveness of a language is a collection of features that makes it easy to use, self-documenting and elegant. DAML-S, BPEL and WSCI are quite expressive with respect to process modeling constructs. DAML-S however offers an advantage over the two in its capability of expressing the pre-conditions and effects of service execution. Since DAML-S is an ontology, apart from XML data types, it also exposes a well-defined type system that enables reasoning about relationships between DAML-S classes. WSDL is restricted in its expressiveness of service behaviour to input/output as XML types. BPEL and WSCI, which derive WSDL port information for service description, therefore have limited expressivity in terms of typing mechanism.

Error handling and transaction management in case of service failure has to be an integral part of any composition model. Exception handling and Transaction constructs are present in both WSCI and BPEL but not in DAML-S. Fault and compensation handlers in BPEL seem to be more clearly defined then in WSCI. Both WSCI and BPEL allow roles to be defined. However no such construct is yet available in DAML-S. This is an important issue since roles help identify the responsibilities of partners in the composition. Correlation mechanism supported by WSCI and BPEL is important to synchronize the messages that are received by a service from different entities. This feature is currently not supported by DAML-S.

The process of marking up services using any of these languages is a cumbersome one if carried out manually. Editors and Engines are needed for creating, parsing and executing processes written

in these languages. The support for tools is very limited as language development is ongoing. An engine for BPEL is available at [6]. A number of efforts are in place for development of tools for DAML-S.A semi-automated service composer has been developed at University of Maryland. The SunONE WSCI Generator supports WSCI, however it does not provide means for testing the generated markup.

Extensibility of language constructs is necessary for enhancing the interface definitions. BPEL and WSCI allow for this with additional constructs from other XML namespaces. DAML-S allows this extensibility through the import construct and also through inheritance. Addition of rules is an important factor to allow for the different types of reasoning domains that are required by Web services. DAML-S is more at an advantage as regards the ease of incorporating rule definitions then the other languages since they are not based on a formal semantics. Security and Privacy issues have not been exclusively handled in any of these languages. They have been mentioned as future work, however currently the specifications of any of these languages do not provide mechanism to enforce security and privacy within the composition. Qualities of Service (QoS) [16] requirements are handled to some extent in DAML-S through its profile model, however there are no explicit QoS monitoring mechanisms available for BPEL and WSCI.

3 Web Service Composition

With the rapid expansion of Web services related applications in fields such as e-business, e-government and e-health, there is an increase in the demand for frameworks and infrastructures that can be used to develop applications that use Web service composition. In this section we first present a taxonomy of service composition types and then we refer to a number of initiatives that are trying to tackle the issue of composition, in particular, research on automated composition of Web services.

3.1 Service Composition Categories

Service composition as defined in [17] as the process through which newly customised services are created from existing ones by a process of dynamic discovery, integration and execution of those services in a deliberate order to satisfy user requirements. This process can be seen from two perspectives as described in [18]: the first is proactive versus reactive composition and the other is mandatory versus optional composition.

Proactive or static composition refers to offline or pre-compiled composition. Services that compose in such a manner are usually stable (i.e., they do not change very frequently) and are highly requested over the Web. Reactive or dynamic composition refers to the creation of services on the fly. Such composition requires some management facility to take responsibility of collaborating with the different sub-services to provide the composite service to the client. This interaction cannot be predefined and varies according to the dynamic situation. Reactive composition is better to exploit the present state of services and to provide certain runtime optimisations based on real-time parameters like bandwidth and cost of execution of the different sub-services.

Mandatory composite services refer to the situation where by all the sub-services must participate for the proper execution. These types of services are dependent on the successful execution of other services to produce the required result. Optional composite services are the opposite of the former and they do not necessarily need the participation of certain sub-services for the successful execution of a user query.

3.2 Related Work

There exists some work done on service composition that mainly focused on the dynamic execution of services. We also refer to ongoing research that is focused on automating this process.

The dynamic service composition called software hot swapping has been developed at the Carleton University, Canada [19]. This work is successor to the research in the field of dynamic software component upgrading at runtime. They have identified two different ways of carrying out heterogeneous service composition. The first method involves the formation of a composite service interface by which the necessary available services are exposed to a client. The advantage of this technique is the speed that a composite service can be created and exported. A second method creates a standalone composite service that is more suitable when the performance of the composite service is more critical. Such a service is created by dynamically assembling the available services by the way of pipes and filters, while all the service components remain independent.

eFlow [20] from HP labs, is an e-commerce services composition system. A composite service is modelled as a graph, which defines the order of execution among the nodes or different processes. The graph modelling a composite service consists of service nodes, event nodes or decision nodes. Service nodes represent simple or composite services. Event or decision nodes specify alternative rules that control the execution flow. Event nodes enable services and the process to receive various types of event notification. The eFlow engine offers the facility of being able to plug in new service offerings and enables adaptivity with several features such as dynamic service discovery, multi service nodes and generic nodes.

The Self-Serv [21] framework can compose Web services and the resulting composite service can be executed in a decentralised dynamic environment. The providers of the services participating in a composition collaborate in a peer-to-peer fashion to ensure that the control-flow dependencies expressed by the schema of the composite service are respected. A subset of statecharts has been adopted to express the control-flow perspective of the composite service. States can be simple or compound. The data-exchange perspective is implicitly handled by variables: which are the inputs and outputs, the parameters of services and events.

Golog [32] has been used for service composition by constructing general templates that are then modified based on user preferences, yielding a composite plan. The templates are not automatically built and constitute part of the plan.

We now turn our attention to some work that we are considering as being the initial idea for the composition engine we propose to implement. In particular we quote the work done in [22] which advocates a phased approach to service composition and which is collectively referred as the service composition life cycle. These phases describe the service composition process from the abstract specifications definition to their execution. Five phases are defined; the planning phase, the definition phase, the scheduling phase, the construction phase and finally the execution phase.

The planning phase assists the user in determining the series of operations that need to be retrieved and aggregated in order to satisfy the users request. Service requests define the desired service attributes and functionality, including temporal and non-temporal constraints between services, and the way the services are scheduled.

Once the services to be composed are chosen, an abstract definition of the composition is handled by the definition phase. This will provide for a customised specification of the new service that represents the definition or process model of the orchestrated service, together with grounding definitions that allow for the service bindings.

The scheduling phase is responsible for determining how and when the composed services will run and prepares them for execution. This phase is the first step towards a concrete definition

of the constructs defined in the process specification created in the previous phase. Scheduling includes the assessment of the service composability and conformance capabilities by correlation of messages and operations, and by synchronizing and prioritising the execution of the constituent services according to the process specification.

The construction phase results in the construction of a concrete and unambiguously defined composition of services that are ready to execute.

Lastly there is the execution phase that implements the service bindings and executes the services.

Though this work does not provide an automated composition process we think it is very important since it presents an interesting approach to the service composition problem. As described in [23], [24], [25] and [26] the semantic-web community is adapting AI planning techniques to give a solution to this problem. In AI planning, researchers investigate the problem of how to synthesise complex behaviours given an initial state. In the next section we propose our initial ideas for service composition based on the findings from the papers above together with some ideas of our own.

4 Composition Engine Some Issues

In this section we propose a composition engine that can be used by a user agent for the automated composition of Web services. In what follows we will give a rationale to the choices we made and discuss how some implementation issues will be handled.

We think that its important that Semantic Web technology is integrated with Web services as discussed in [27], where it is argued that to achieve the long term goal of seamless interoperability, Web services must embrace many of the representations and reasoning ideas proposed by the Semantic Web community and in particular the Semantic Web services community. We have tried to adopt this perspective to our work, and as discussed above we have looked in detail into a number of issues. First comes the composition language of choice. Though BPEL is somewhat more apt to fault and compensation handling, as described in our work [14], we think that it still requires work in the area of reasoning, since it is based on XML. In [27] BPEL is augmented with a semantic component to handle exactly this issue. Nonetheless we feel that DAML-S is somewhat more ideal at this early stage and hence we settled for this language for composition.

We are also opting to abide by the service composition cycle as defined in [22] and plan of adopting the same kind or architecture. This involves the creation of several modules mentioned in the paper, including, planner, definer, scheduler and executor. We will also be adding a reasoning and communications module.

As defined in the paper, the scheduler does not cater for fault and compensation handling nor for service execution monitoring. These are important issues that are missing from DAML-S as well and are discussed in [28]. If we pursue on including such functionality in our engine, then we must enhance the DAML-S language to handle these issues. This is not a trivial task and is still being debated. The other task handled by the scheduler is the decision of which service to execute and when. The scheduled service is then linked with the executor to be executed.

As regards the communication handler, the idea is to include the functionality that the engine can communicate with user agents or other composition engines, as the necessity might arise, hence creating a distributed environment rather then a centralised one. For this aim we plan to use JADE (Java Agent Development Framework)[29]. JADE is a software framework fully implemented in Java. It simplifies the implementation of multi-agent systems through a middle-ware that claims to comply with the FIPA (Foundation for Intelligent Agents) [30] specifications and through a set

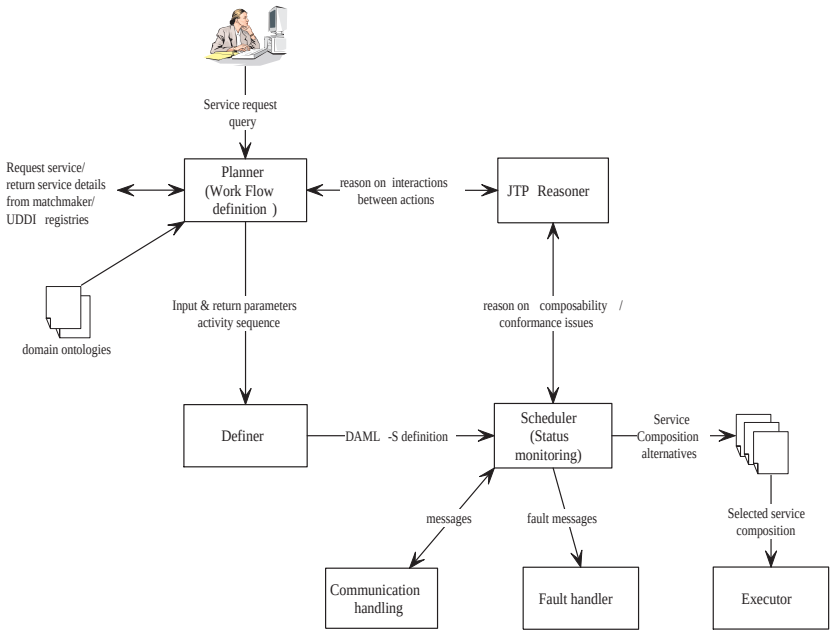

Fig. 2. Composition Engine architecture

of tools that supports the debugging and deployment phase. The agent platform can be distributed across machines (which not even need to share the same OS) and the configuration can be controlled via a remote GUI. The configuration can be even changed at run-time by moving agents from one machine to another one, as and when required. We also plan to use DAML+OIL as the ACL (agent communication language) that will be embedded into SOAP messages.

The reasoning module will involve the use of JTP (Java theorem Prover) [31]. This is an object-oriented modular reasoning system. JTP is based on a very simple and general reasoning architecture. The modular character of the architecture makes it easy to extend the system by adding new reasoning modules (reasoners), or by customizing or rearranging existing ones. The system is implemented in Java and this facilitates both extending JTP's functionality and embedding JTP in other systems.

The definer will be required to generate new DAML-S process specifications from the individually composed service specifications.

The system will also require the facility of a matchmaker or UDDI registries from where the services can be retrieved and which can also be used to store the newly composed service definitions for reusability reasons. But this is an issue that we will be assuming as already in place hence it is out of scope of our work.

5 Discussion

At this stage we are reviewing several AI planning issues that can be used in the engine. We are trying to find a compromise between the work that already exists and the work that we might have to do to extend/adopt such research for our needs. Since this is a relatively new area we are faced with limited reviews and therefore require our own initiative to handle these tasks. For example

we might integrate the planner and scheduler into one module depending on the algorithm that we choose to adopt. Nonetheless we feel confident that since we already have the architecture in place then work can be initiated.

References

1. Universal Discovery Description and Integration Protocol: http://www.uddi.org/
2. Web Services Description Language http://www.w3.org/TR/wsdl
3. Simple Object Access Protocol http://www.w3.org/TR/soap12-part0/
4. DAML-S and OWL-S: http://www.daml.org/services/daml-s/0.9/
5. Web ontology Group http://www.w3.org/2001/sw/WebOnt/
6. OWL http://www.w3.org/TR/owl-ref/
7. M. Paolucci, T. Kawamura, T.R. Payne, and K.Sycara, Semantic Matching of Web Services Capabilities. In The First International Semantic Web Conference (ISWC), 2002
8. M.Montebello, C.Abela, DAML enabled Web services and Agents in the Semantic Web, NetObjects Days, Erfurt Germany, 2002.
9. M. Paolucci, Z. Niu, K. Sycara, C. Domashnev, S. Owens and M. Van Velsen. Matchmaking to Support Intelligent Agents for Portfolio Management. In Proceedings of AAAI2000 (Demo Session)
10. F. Curbera, Y. Goland, J. Klein, F. Leymann, D. Roller, S. Thatte, S. Weerawarana. Business Process Execution Language for Web Services, Version 1.0, 2002. http://www-106.ibm.com/developerworks/library/ws-bpel/.
11. S. Thatte. XLANG: Web Services for Business Process Design, 2002.
12. F. Leymann, IBM. Web Services Flow Language (WSFL) Version 1.0, 2001.
13. Intalio, Sun Microsystems, BEA Systems, SAP. Web Service Choreography Interface (WSCI) 1.0 Specification, 2002.
14. C. Abela, M.Solanki, A Landscape of Markup Languages for Web Services Composition, submitted at NetObject Days 2003, Erfurt Germany, http://cicho0.tripod.com/-WSCompositionLanguages.pdf
15. Web Services Choreography Working Group: http://www.w3.org/2002/ws/chor/
16. Senthilanand Chanrdasekeran, Composition, performance analysis and simulation of Web services, Masters Theses, University of Georgia Athens, 2002.
17. D.Chakraborty, T. Finin, F. Perich, A Reactive Service Composition Architecture for Pervasive Computing Environments, Singapore, PWC 2002,
18. D. Chakraborty, A. Joshi, Dynamic Service Composition: State-of-the-Art and Research Directions, University of Maryland, Baltimore County, Baltimore, USA, 2001.
19. D. Mennie, B. Pagurek, An Architecture to Support Dynamic Composition of Service Components, Systems and Computer Engineering , Carleton University, Canada, 2000.
20. F. Casati, S. Ilnicki, L. Jin., Adaptive an Dynamic service Composition in eFlow, Software technology lab, Palo Alto, 2000.
21. Benatallah, Dumas, Fauvet, Paik, Self-Coordiante, Self-traced Composite Services with Dynamic Provider Selection, Technical report, School of Computer Science & Engineering, University of New South Wales, Australia, 2001.
22. S. McIlraith and T. Son. Adopting Golog for Composition of Semantic Web Services. In proceedings of the Eigth International Conference on knowledge representation and Reasoning (KR 2002), Toulouse, France.
23. J. Yang,. & M.P. Papazoglou,: Service Components for Managing the Life-Cycle of Service Compositions. Will appear in Information Systems, June, 2003.
24. M.P. Papazoglou., M. Aiello, M. Pistore, & J. Yang,: Planning for Requests against Web Services. In Bulletin of the Technical Committee on Data Engineering, December 2002.
25. D. Wu, E. Sirin, J. Hendler, D. Nau & B. Parsia, Automatic Web Services Composition Using SHOP2, International Conference on Automated Planning & Scheduling , ICAPS 2003.
26. M. Carman, L. Serafini & P. Traverso, Web Service Composition as Planning, , International Conference on Automated Planning & Scheduling , ICAPS 2003.
27. M. Sheshagiri, M. desJardins, and T. Finin, A Planner for Composing Services Described in DAML-S, International Conference on Automated Planning & Scheduling , ICAPS 2003.

28. D. Mandell, S. McIlraith, A Bottom-Up Approach to Automating Web Service Discovery, Customization, and Semantic Translation, Twelfth International World Wide Web Conference Workshop on E-Services and the Semantic Web (ESSW '03). Budapest, 2003.
29. DAML-S: Semantic Markup for Web Services, http://www.daml.org/services/daml-s/0.9/-daml-s.html
30. JADE: http://sharon.cselt.it/projects/jade/
31. FIPA: http://www.fipa.org/
32. JTP: http://www.ksl.stanford.edu/software/JTP/