The Landscape of Markup Languages for Web Service Composition

Monika Solanki[1], Charlie Abela[2]

[1] Software Technology Research Laboratory,
De Montfort University,
The Gateway, Leicester LE1 9BH, UK
E-mail:*monika*@dmu.ac.uk
[2] CSAI Dept,
University of Malta,
MSD06. Malta
E-mail:*cabe*002@um.edu.mt

Abstract. Two of the most hyped technologies of recent times, are "Web Services" and "Semantic Web". This is evident in the fact that the Web service technology stack is overloaded with acronyms of these developments. Markup languages for specifications of Web services are set to play an important role, especially, in enabling dynamic service discovery and composition by human users and software agents. There is a plethora of languages, proposed by academic and industrial research groups, for Service description, discovery, composition, execution and interpretation with XML as their backbone. Some of these languages have well defined underlying semantics, others are based on workflow patterns, while a few have emerged as combinations of other independent languages. With so many languages and each one of them proposed to be a standard, it becomes imperative to critically analysis the merits and demerits of these languages. In this paper we present a case study, to emphasis the salient features of some of these languages and compare them with respect to attributes like expressiveness, support for the semantics, supporting tools available, the core composition mechanism, support for fault tolerance and exception handling. Our choice was motivated by the industrial support behind them and/or sound academic research.

keywords: Web Services, composition, Semantic Web

1 Introduction and Motivation

The emphasis in Application Development today has shifted from tightly coupled, monolithic, proprietary software to loosely coupled, dynamically bound service based systems, comprising of distributed components provided by more than one vendor. The current Web service model [13] 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.

However, the dynamics of service composition still remains one of the most challenging aspects for researchers in academia and industry. In the present scenario, configurations for Web services revolve around three main XML based technologies namely, Universal Description Discovery and Integration (UDDI)[15], Web Services Description Language (WSDL)[11] and Simple Object Access Protocol (SOAP)[8]. Several ongoing Industrial initiatives in the development of service markup languages such as BPEL4WS [1], XLANG [12], WSFL[4], WSCI [7] and have exploited these technologies. These 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 important goals of the Semantic Web. Academic research has lead to development of Ontology Markup Languages such as DAML+OIL [2] and OWL [9]. A Web service ontology DAML-S [14], defined through DAML+OIL, 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 these languages. Further, assuming that a majority of these 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 forsee 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. Recent efforts towards some kind of standardization of these technologies has resulted in the creation of the Web Services Choreography Working Group (WSC-WG). Figure 1 depicts a layered view of the currently available technologies for Web services.

In this paper,we aim to review the characteristics of some of the well established markup languages for service composition (also referred to as "Web service execution language" or "Web service Orchestration language") and compare them with respect to those attributes. Our motives behind such a comparison is not to show how certain features in a language, can be achieved through some means in the others. We believe that a comparative analysis would bring forth the strengths and weakness of each language and contribute towards the standardization process. The structure of the paper is as follows. Section 2 presents requirements that are needed to be satisfied by any service composition language. This is followed by a brief summary of various well known service markup technologies in section 3. Section 4 presents a tabular comparison. of various key attributes for these languages, namely,

– Expressivity of the language.
– Supporting tools available.
– Level of abstractness.

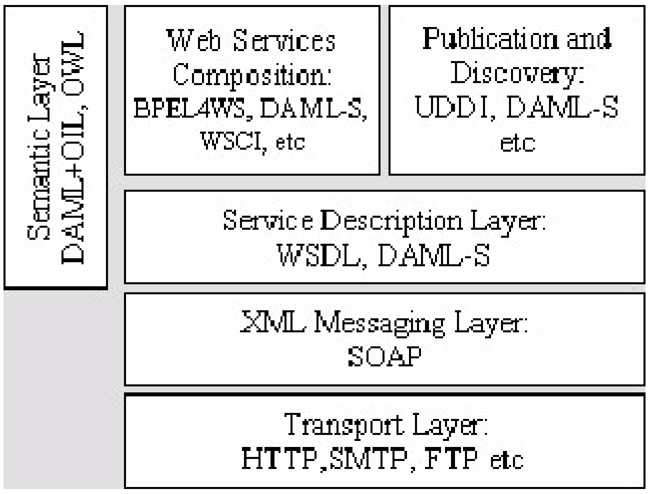
Semantic Layer
DAML+OIL, OWL
Web Services Composition: BPEL4WS, DAML-S, WSCI, etc
Publication and Discovery: UDDI, DAML-S etc
Service Description Layer: WSDL, DAML-S
XML Messaging Layer: SOAP
Transport Layer: HTTP,SMTP, FTP etc

Fig. 1. *Overview of present Web Services Technologies*

– Core composition mechanism: the core technique used for service composition.
– Support for fault tolerance and exception handling.
– Support for the semantic layer of the Web.

An analysis of the comparison is presented in section 5 and we finally conclude with some views on directions of future work in this area.

2 Requirements of a Web Service Composition language

– A composition language should be self sufficient to expose details of a service like its profile, process model and binding mechanism.

– A composition language should be able to specify component interfaces in an implementation independent manner.

– A composition language should support the development of reusable component frameworks using basic programming control constructs.

– A composition language should provide support for state based process modelling control constructs. The model should be able to handle sequential composition, concurrent constructs like split, fork, spilt+join, looping constructs like iterate, repeat and while and nondeterministic constructs like choice and switch. This also implies providing support for modelling subprocesses.

– A composition language should be able to specify process descriptions in an abstract manner without binding to actual execution details to enable reusable descriptions and also be able to model executable processes that encapsulate the internal behaviour of a service.

- A composition language should expose modelling constructs that specify well defined responsibilities for the players involved in the composition process i.e. requestor agents, provider agents and the matchmaker.

- A composition language should be able to cater for fault tolerance,deadlock, livelock and Exception handling mechanism.

- A composition language should be able to model component specifications as abstractions of component behaviour.

- A composition language should expose models and protocols for binding of the actual service.

- The composition language should be adequately expressive, have well defined semantics and a robust underlying formal model, to facilitate automated composition of services through software agents.

- A composition language should be able to explicitly model the order of message passing between the requestor and provider and their implications on the subsequent operations.

- A Composition language should be able to model constraints in the form of rules to inhibit undesirable interactions between the services. This also includes support for a transaction model and quality of service.

- Security and Privacy should form an integral part of any service composition language. The language must be able to model security as a functional requirement of the composed system.

- A composition language should include support for "pay-as-you-go-computing" in the form of well defined constructs for the pricing models.

3 Markup Languages :A brief Summary

In the section, we present an overview of the languages currently proposed for markup.[1]

[1] The descriptions are only an overview and not meant to be complete in any respect. Pls refer individual specifications for complete details.

3.1 Web Service Description Language (WSDL)

WSDL is an XML format for descriptions of abstract Web service functionality and a framework for describing concrete Web service bindings. A WSDL description describes the abstract Web service interface through which a service consumer communicates with a service provider, as well as the specific details of how a given Web service has implemented that interface. Version 1.2 of the WSDL specifications, describe it as a Component Model consisting of WSDL components like messages (typed data elements) and operations (a set of input and output messages), port types (a set of operations), bindings and services, and type system components. Port types are reusable and can be bound to multiple ports. A binding component describes a concrete binding of a port type component and associated operations to a particular concrete message format and transmission protocol. The 1.2 specifications describe bindings for WSDL with SOAP, HTTP and MIME. At a concrete level, a service component is a collection of ports (network endpoints) with each port mapped to a port type. The Types component is an abstract container for datatype schemas included or imported in the service description. WSDL supports any schema provided the semantics for it have been defined. By default W3C XML Schema support is expected to be provided by all tools/editors/parsers for WSDL..

3.2 DAML+OIL and OWL

Ontologies are set to play a key role in the "Semantic Web" extending syntactic interoperability to semantic interoperability by providing a source of shared and precisely defined terms. DAML+OIL is one of the first languages designed for expressing ontologies by the Darpa Agent Markup Language committee. A more recent development is the Ontology Web Language (OWL) by the W3C (World Wide Web Consortium) Web Ontology Working Group (WebOnt). OWL has been inspired heavily by DAML+OIL and builds up on top of it. The DAML service coalition of the DAML (Darpa Agent markup language) program is developing a DAML based Web Service Ontology: DAML-S. DAML-S supplies Web Service providers with a core set of markup language constructs for describing the properties and capabilities of their Web Service in an unambiguous computer interpretable form. DAML-S intends to facilitate the automation of Web Service tasks including automated Web service, discovery, execution, interoperation, composition and execution monitoring. Following the layered approach to markup language development, the current version of DAML-S builds on top of DAML+OIL. As and when OWL stabilises, future versions of DAML-S will be definitely built on top of OWL. We present a brief discussion on DAML+OIL and OWL in this section. DAML+OIL is an Ontology language specifically designed for use on the Web. It exploits existing Web standards (XML, RDF and RDF-S) and extends these languages with richer modelling primitives, adding the familiar ontological primitives of object oriented and frame based systems, and the formal rigor of Description logic. DAML+OIL is designed to describe the structure of a domain. DAML+OIL takes an object-oriented approach with the structure of the domain being described in terms of classes and properties. An Ontology consists of a set of axioms that assert e.g. subsumption/equivalence relationships between classes and properties. DAML+OIL is written in RDF. A DAML+OIL knowledge base is a collection of RDF triples, i.e., DAML+OIL markup is a specific kind of RDF markup. RDF, in turn, is written in XML, using XML Namespaces and URIs Formally DAML+OIL can be seen as a DL with DAML+OIL Ontology corresponding to the Description Logic terminology. DAML+OIL classes can be names or expressions and a variety of constructors are provided for building class expressions. The expressive power of the language is determined by the class and property constructors supported and by the kinds of axioms supported.. A DAML+OIL is made up of several components some of which are optional and some of which may be repeated. A DAML+OIL ontology consists of zero or more headers, followed by zero or more class elements, property elements and instances. Ontology Web language facilitates greater machine readability of Web content than that supported by XML, RDF, and RDF Schema. OWL is a neat revision of DAML+OIL. The OWL language provides three increasingly expressive sublanguages: OWL Lite, OWL DL (description logic), and OWL Full. OWL Full can be viewed as an extension of RDF, while OWL Lite and OWL DL can be viewed as extensions of a restricted view of RDF. Language constructs for OWL Lite include RDFSchema features, property characteristics and property type restrictions, restricted cardinality, header information, datatypes, (In)equality constructs, and class intersection construct. In addition to the constructs from OWL Lite, OWL DL and Full include constructs like Class Axioms, Boolean combination of class expressions, arbitrary cardinality and Filter information construct. OWL makes use of the RDF datatyping scheme, which provides a mechanism for referring to XML Schema datatypes. OWL supports standard notions of ontology referencing, inclusion, and meta-information.

3.3 XLANG

XLANG is Microsoft's idea for the choreography of Web Services. It is a language that makes it possible to formally specify business processes as stateful long-running interactions. The interactions between services occur through message exchanges expressed as WSDL operations. An XLANG service description is defined as a WSDL service description with an extension element that describes the behaviour of the service as part of a business process. XLANG focuses on the publicly visible behaviour in the form of messages exchanged. An interaction in XLANG is an instantiation of a service. Since the exchanged messages not only need to be delivered to the correct destination port, but also to the correct instance of the service that defines the port, each message contains a 'correlation token'. A set of such correlation tokens (a 'correlation set') is defined as a set of properties shared by all messages in the correlation group. XLANG assumes that port types, used in the definition of XLANG service descriptions, are constrained to contain only incoming, or only outgoing operations. WSDL port types however do not have this polarity. This is an issue that should be addressed to achieve the goals of XLANG. The behaviour of an XLANG service is composed of actions. These actions can be WSDL operations, delays (e.g. the thread of behaviour has to pause for some other process to execute), and 'raise actions' (i.e. the notification of exceptional conditions). In an XLANG service, the actions are combined using basic control processes; as there are 'while', 'sequence', 'empty', 'switch', 'all', 'pick', and the advanced forms 'context' and 'compensate'. The 'context' process provides a framework for local declarations, exception handling and transactional behaviour. Contexts may be used to delineate the scope of a part of the behaviour that has transaction-like properties. Within XLANG this primarily means that an implicit or explicit compensation process is associated with the context, and this can be invoked using the 'compensate' process. XLANG not merely talks about compensating actions but about compensating processes.

3.4 Web Service Flow Language

The Web Services Flow Language (WSFL) is a suggested standard of IBM. It is a very complete and neutral language and is integrated with UDDI and WSDL for dynamic selection of Web Services. WSFL enables developers to create, execute, and combine Web Services into complex processes and workflows. WSFL itself does not deal with the modeling of the business process; rather it is a specification for how to implement a business process model using the Web Services architecture. WSFL can be used to create an XML representation of the business model, and then feed that XML-representation into a middleware application designed to invoke and manage the process. A WSFL document uses 'serviceProviderType' elements to identify the roles of the implementers of specific activities within the context of a given business process model. The serviceProviderType is defined by a Web service Interface document using WSDL. Service providers must properly implement the appropriate Web service Interface in order to be classified as the appropriate type of service provider to handle a particular activity in the business process. Each process is represented in an XML 'flowmodel'. This flowmodel is an abstract definition of the workflow process. The flowmodel contains 'Activity' elements, which define activities that are implemented in the form of a Web service defined by WSDL. 'ControlLinks' and 'dataLinks' are also part of the flowmodel. The former describe the sequence of activities, whereas the latter define the flow of data from one activity to another. Next to the flow model there is a 'globalModel' that details how a given process is implemented. As such, the globalModel defines the identity, the location and the implementation of the service provider that implements a specific role. Once the global and the flow model for a given business process are defined, the whole business process can be defined as a single Web service, that may be used by other business processes.

3.5 Business Process Execution Language for Web Services

BPEL4WS is a combined effort from IBM's and Microsoft. Infact it builds on top of WSFL and XLANG by combining accordingly the features of a block structured language inherited from XLANG with those for directed graphs originating from WSFL. The language is intended to support the modelling of both executable and abstract processes. An abstract process is a business protocol that specifies the message exchange behaviour between different parties without revealing their internal behaviour. An executable process specifies the execution order between a number of activities that constitute the process, the partners involved in the process, the messages exchanged between these partners, and the fault and exception handling that specify the behaviour to adopt in cases of errors and exceptions. The BPEL4WS process is a kind of flow-chart, where each element in the process is called an activity. An activity can be either primitive or structured. The set of primitive activities contains: invoke, which is used to invoke an operation on some Web service; receive, that is used to wait for a message from an external source; reply, which is used when replying to an external source; wait, when it is necessary to wait for some time; assign, for copying data from one place to another; throw, to indicate errors in the execution; terminate, when terminating the entire service instance; and empty, when the process is doing nothing. Several structured activities are defined to enable the presentation of complex structures. These are: sequence, which is used to define an execution order; switch, used for conditional routing; while, used for looping; pick, for race conditions based on timing or external triggers; flow, which is used for parallel routing; scope, for grouping activities to be treated by the same fault-handler; and compensate, which is used to undo the effects of already completed activities. Structured activities can be nested and combined in arbitrary ways. Within activities that are executed in parallel the use of links can further control the execution order. These are sometimes also called control links and allow the definition of directed graphs. The graphs can be nested but must be acyclic.

BPEL make also use of two significant and complementary specifications, which are the WS-Coordination and WS-Transaction and are also developed jointly by IBM and Microsoft. They deal with how one coordinates the dependable outcome of both short- and long-running- business activities. The WS-Transaction specifies a framework that allows a composed Web Service to monitor the success or failure of each individual, coordinated activity. It provides the means for the service to monitor the process and reliably cancel the process in case something goes wrong along the way. The WS-Coordination specification on the other hand, defines a framework through which the composed services can work from a shared "coordination context". This context contains the information necessary to link the various activities together.

3.6 Web Service Choreography Interface

WSCI is a combined effort by BEA Systems, Intalio, SAP AG, and Sun Microsystems towards describing an interface definition language for choreographing the flow of messages between Web services. The language takes WSDL as the starting point. WSCI is a construct based language and deals with the external observables rather than the internal definition of service behaviour. The concepts underlying the model are the interface definition, choreography of activities, definition of processes and their properties, message correlation, exception handling, transaction and compensation description and dynamic participation. The details of the behaviour of the Web Service are described in the processes that are contained in the interface. A Web Service may expose multiple interfaces for supporting multiple scenarios. Activities can be atomic activities representing the basic unit of behaviour of a Web service, as well Complex activities which are recursively composed of atomic services. WSCI supports sequential, parallel, looping and conditional execution. WSCI allows the definition of two types of processes, namely, Top level and nested processes which can be referenced using Call or a Spawn statement. Properties are introduced in WSCI as a modeling artifact used to reference a "value" within the interface definition. They are the equivalent of variables in other languages. Context is a WSCI concept that describes the environment in which a set of activities is executed. A context definition may contain two different kinds of declarations: local properties, and local process definitions. The concept of correlation describes how conversations are structured and which properties must be exchanged to retain the semantic consistency of the conversation. A correlation is not limited to a single conversation between two participants; it can span multiple conversations between different participants. WSCI allows declaring exceptional behaviour that is exhibited by a Web Service at a given point in a choreography. The declaration of exceptional behaviour is part of the context definition. The transaction describes, from an interface perspective, the transactional properties of the activities that are executed in this context. A transaction is either atomic or open-nested. WSCI allows also describing a multi-participant view of the overall message exchange by means of the WSCI Global Model. WSCI also supports extensibility of the language constructs.

4 A Tabular Comparison of Language Attributes

We present below a comparison of certain key attributes for various languages[2]. Since WSDL is the underlying infrastructure for all these languages and also a standard, we do not compare attributes with direct dependency on WSDL.

Parameters	WSCI	BPEL4WS	XLANG	WSFL	DAML-S
Support for Automated Composition	-	-	-	-	+
Dependency on other languages (WSDL)	+	+	+	+	+
Expressiveness	+	+	-	-	+
Formal Semantics	-	*	*	-	+
Process modelling constructs supported	+	+	+	+	+
Defined Level of Abstraction	low	low	low	low	high
Transaction management	+	+	-	-	-
Exception Handling mechanism	+	+	-	-	-
Tool support available	low	low	low	low	low
Worflow modelling support	+	+	+	+	+
Semantic Constraints	-	-	-	-	+
Security and Privacy issues	-	-	-	-	-
QoS Requirements	-	-	-	+	+

Table 1. Comparison of Service Composition languages

Parameters	WSCI	BPEL4WS	XLANG	WSFL	DAML-S
Sequence	+	+	+	+	+
Parallel (Split & Split+Join)	+	+	+	+	+
If-then-else	+	+	+	+	+
Looping	+	+	-	+	+
Choice	+	+	+/-	+	+
Termination	+	+	-	+	-
Cancellation	+	+	+	+	-

Table 2. Process Modelling Constructs Supported

[2] **Symbols used in the table:**
: ambiguous, +: direct support for the construct available
-: no direct support for the construct available +/-: limited support for the construct available

4.1 Analysis of the comparison

We analyse the results from the comparison above, taking into account the attributes of three core languages namely, DAML-S, BPEL4WS and WSCI, since BPEL4WS has emerged from a coalition of WSFL and XLANG. This analysis also puts into perspective some of the requirements stated in section 2

To realise 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. BPEL4WS 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, BPEL4WS and WSCI are quite expressive with respect to process modelling 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. BPEL4WS 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 BPEL4WS but not in DAML-S. Fault and compensation handlers in BPEL4WS seems to be more clearly defined then in WSCI. Both WSCI and BPEL4WS 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 BPEL4WS, is important to synchronize the messages which 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 BPEL4WS 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 be 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. Quality of Service requirements are handled to some extent in DAML-S through its profile model, however there are no explicit QoS monitoring mechanisms available for BPEL4WS and WSCI.

5 Conclusions

A number of initiatives have already been taken towards a comparative study of the markup languages. The differences between DAML-S and BPEL4WS have been highlighted in [3]. An extensive pattern based comparison of various service composition languages has been done in [10, 17, 16]. These however offer only a comparative study without stating the requirements that need to be fulfilled by any markup language designed for service composition. [3] offers a comparison only between DAML-S and BPEL4WS.without considering WSCI, which we believe is an important work in this area.

At present there are a number of initiatives that are focused on the composition aspect of Web services. Their promise is to provide a composition language that can be considered as a standard and which is able to support the important components that we have highlighted in this paper. The DAML-S group has a number of pending issues in the pipeline. On the forefront is the advent of OWL, which will bring about a new version of DAML-S, this time based on OWL and with the synonymous name of OWL-S. Some other features that were planned regard some extensions to the WSDL grounding and also a recommended way of specifying conditions. IBM and Microsoft have great plans for BPEL, the first one being the submission of the language to the OASIS standards body. They are promising several improvements to the language, such as those regarding scope and fault handling with some important emphasis on the issue of security. These improvements to the languages will undoubtly bring about more research in the areas of Web Services and composition. We should expect a number of initiatives that will give rise to the creation of a number of semantically enabled technologies related to these areas, from ontology builders and verifiers, to service composition frameworks for dynamic aggregation of Web services and also to Web service registries that are capable of handling the different types of service technologies. Such language improvements will also increase the research in areas such as those involving formal semantics definitions, description logics and rule based semantic Web services. Another ongoing initiative is the European effort WSMF (Web Services Modeling Framework) [5]that will definitely have an important role in this area as the project is targeting the creation of a framework for Web services that is based on the principal of maximal decoupling and scalable mediation of services. It will build over a number of already defined technologies such as UDDI and WSDL but will also consider semantically enabled technologies such as DAML-S or OWL-S.

In this paper we highlighted the various research efforts that are targeting Web service composition languages. We described them by showing the capabilities of each one and compared them on the bases of a number of important features such as expressivity, support for semantics, tool support, core composition mechanism and fault tolerance and exception handing. We then continued by comparing their support for process modelling since this is considered an important issue for composition.

In the absence of a declared standard language, we believe that a possible solution should run parallel to the research being carried on in the Semantic Web and have foundations built over DAML-S. Nonetheless, it is imperative that such a language adapts the concepts of transactions, fault and compensation handling found in other languages such as BPEL4WS and WSCI to provide a complete solution for the automatic composition of Web services.

References

1. Francisco Curbera, Yaron Goland, Johannes Klein, Frank Leymann, Dieter Roller, Satish Thatte,Sanjiva Weerawarana. *Business Process Execution Language for Web Services, Version 1.0*, 2002. *http://www-106.ibm.com/developerworks/library/ws-bpel/.*
2. Joint US/EU ad-hoc Agent Markup Language Committee. Reference description of the daml+oil (march 2001) ontology markup language. 2001. http://www.daml.org/2001/03/reference.
3. DAML-S and Related Technologies. http://www.daml.org/services/daml-s/0.7/survey.pdf.
4. Dr. Frank Leymann, IBM Software Group. *Web Services Flow Language (WSFL) Version 1.0*, 2001.
5. D. Fensel and C. Bussler. The web service modeling framework wsmf. NSF-OntoWeb Invitational Workshop on DB-IS Research for Semantic Web and Enterprises, 2002.
6. IBM. http://www.alphaworks.ibm.com/tech/bpws4j.
7. Intalio, Sun Microsystems, BEA Systems, SAP. *Web Service Choreography Interface (WSCI) 1.0 Specification*, 2002.
8. James Snell, Doug Tidwell and Pavel Kulchenko. Programming Web Services with SOAP. O'Reilly, 2002.
9. M. Dean, D. Connolly, F. van Harmelen, J. Hendler, I. Horrocks, D. Guinness, P.F. Patel-Schneider, L. A. Stein. *Web Ontology Language(OWL) W3C Reference version 1.0*, November 2002. *http://www.w3.org/TR/2002/WD-owl-ref-20021112.*
10. P. Wohed, W.M.P. van der Aalst, M. Dumas, and A.H.M. Pattern-based analysis of bpel4ws. Technical Report FIT-TR-2002-04, Queensland University of Technology, Brisbane, 2002.
11. Roberto Chinnic, Martin Gudgin,Jean-Jacques Moreau, Sanjiva Weerawarana. *Web Services Description Language (WSDL) Version 1.2*, 2003. *http://www.w3.org/TR/2003/WD-wsdl12-20030124/#intro.*

12. Satish Thatte. *XLANG: Web Services for Business Process Design*, 2002.
13. Web Service Architecture Team. Web services architecture overview. *http://www-106.ibm.com/developerworks/library/w-ovr/* .
14. The DAML Service Coalition. *DAML-S Semantic Markup for Web Services.* International Semantic Web Conference (ISWC), 2002.
15. *The UDDI Technical White Paper. http://uddi.org/pubs/Iru_UDDI_Technical_White_Paper.pdf.*
16. W.M.P. van der Aalst. Dont go with the flow: Web services composition standards exposed. web services - been there done that? *IEEE-Trends & Controversies*, 2002.
17. W.M.P. van der Aalst, M. Dumas, A.H.M. ter Hofstede, and P. Wohed. Pattern-based analysis of bpml (and wsci). Technical Report FIT-TR-2002-05, Queensland University of Technology, Brisbane, 2002.