Requirements for Service Description and Composition in Agentcities

SDC WG Working Draft, August 19, 2002, sdc-wg@agentcities.org *

Authors:
Stanislaw Ambroszkiewicz (chair) IPI PAN and II AP
Dariusz Mikulowski and Leszek Rozwadowski II AP
Tomasz Nowak IPI PAN
Krzysztof Miodek IM UL
Marek Kisiel-Dorohinicki, Jaroslaw Kozlak, and Krzysztof Cetnarowicz AGH

Abstract. An alternative approach to service description and composition is presented. It is based on agent technology, and on the idea of separating description and composition language from binding, i.e., from specification of data format (exchanged by applications) and transport protocol. Usually, the binding is an integral part of description language, e.g., WSDL, and DAML-S. Starting with this idea, recommendations for construction of a simple service description language as well as for a composition protocol are presented. Agents play crucial role in our approach; they are responsible for service composition.

Status of this document

This is a draft of the document to be submitted to the Agentcities.NET as the output of the working group Service Description and Composition in Agentcities.
Feedback and comments are welcome and may be sent to sdc-wg@agentcities.org

Table of Contents

1. Introduction: What are Web services?
2. Overview of WSDL, DAML-S, and the Work of Web Services Activity of W3C
3. Our Approach to Service Description and Composition
4. General View of an Alternative Proposal of Description Language and Composition Protocol
5. Requirements for Universal Message Format
6. Requirements for the Format of Agent State and Service State
7. Requirements for Description Language
8. Requirements for Composition Protocol
9. Requirements for Abstract Architecture of Agent and Service
10. Requirements for Binding Layer (if any)
11. Usage Scenarios

* The work is done within the framework of KBN project No. 7 T11C 040 20
1 What are Web services?

Perhaps the most popular definition can be found in IBM’s tutorial [9]: 
Web services are self-contained, self-describing, modular applications that can be published, located, and invoked across the Web. Web services perform functions that can be anything from simple requests to complicated business processes ... Once a Web service is deployed, other applications (and other Web services) can discover and invoke the deployed service.

In order to realize this vision simple and ubiquitous protocols are needed. From service providers’ point of view, if they can setup a web site they could join global community. From a client’s point of view, if you can click, you could access services.

What are the solutions proposed by the prominent vendors? Web services are getting to mean just UDDI, WSDL, and SOAP. SOAP (Simple Object Access Protocol) is a standard for applications to exchange XML-formatted messages over HTTP. WSDL (Web Service Description Language) describes what a web service does, where it resides, and how to invoke it, i.e., the interface, protocol bindings and the deployment details of the service. UDDI (Universal Description, Discovery and Integration) is a standard for publishing information about web services in a global registry as well as for web service discovery. Does the stack of standards mentioned above provide sufficient means for automatic service invocation, composition, and integration? The problem is hard. UDDI provides a mechanism for automatic service discovery of potential business partners. At the moment, it is supposed that after discovery, programmers affiliated with the business partners program their own systems to interact with the services discovered. Automatic Web service integration requires more complex functionality than SOAP, WSDL, and UDDI can provide. The functionality includes transactions, workflow, negotiation, management, and security. There are several efforts that aim at providing such functionality, e.g., WSCL, WSFL, XLANG, BTP, and XAML. All these languages are based on SOAP+WSDL+UDDI basic stack, and are complex procedural languages very hard to implement and deploy.

On the other hand there is DAML-S. It is a part of DARPA Agent Markup Language project that aims at realizing the Semantic Web concept. DAML-S is also a complex procedural language for web service composition.

2 Overview of WSDL, DAML-S, and the Work of Web Services Activity of W3C

The basic question is whether the proposed technologies are simple and ubiquitous, and which one is the right one. As we see above, the landscape of solutions for new emerging technology is rich and complex so that it is not easy to find a clear and straightforward path to the one common standard. It seems that the path starts with the basic stack SOAP+WSDL+UDDI, however, it is not clear how to go further. Perhaps the basic stack is not appropriate, i.e., it is too complex so that the next protocols (based on the initial stack) accumulate the initial complexity. There is a consensus that SOAP is the right protocol for the message exchange. However, there is a growing criticism of
WSDL and UDDI, e.g., [5, 6]. As a response to that criticism, a new W3C activity was created: Web Service Activity [18].

According to the Web Service Activity (see [18]) the basic definitions of WSDL are as follows.

A Web Service is a software application identified by a URI [IETF RFC 2396], whose interfaces and binding are capable of being defined, described and discovered by XML artifacts and supports direct interactions with other software applications using XML based messages via internet-based protocols.

A Client is a software that makes use of a Web Service, acting as its 'user' or 'customer'.

A Message is the basic unit of communication between a Web Service and a Client; data to be communicated to or from a Web Service as a single logical transmission.

A set of Messages related to a single Web Service action is called Operation.

An Interface is a logical grouping of operations. An Interface represents an abstract Web Service type, independent of transmission protocol and data format.

An InterfaceBinding specifies the protocol and/or data format to be used in transmitting Messages.

An EndPoint (Port) is an association between a fully-specified InterfaceBinding and a network address, specified by a URI, that may be used to communicate with an instance of a Web Service. An EndPoint indicates a specific location for accessing a Web Service using a specific protocol and data format. A collection of EndPoints is called Service.

As we see above, WSDL is strongly related to the concept of binding. Also DAML-S concept of grounding is consistent with WSDL's concept of binding. The binding is a specification of data format (exchanged by applications) and transport protocol.

3 Our approach to Service Description and Composition

We are going to propose an alternative solution to service description and composition. Our approach is based on agent technology. Perhaps the most related work is LARKS [7] that now has evolved and became a part of DAML-S.

Our idea is to separate binding from service description language as well as from service composition protocol. To realize our idea we must give a different meaning to the notion of message. This meaning is based on the following assumptions:

- Message contents is not a data exchanged by communicating applications.
- Message contents describes how to arrange and synchronize the data exchange between the applications. Transmission of the data is realized in the layer below, where the binding is implemented.

Hence, we have two layers: The first one is for arranging and synchronizing data passing between applications; it is called service description and composition layer. The second one is for message and data transmission; it is called binding layer. We claim that the binding layer can be arbitrary, for example, SOAP is an excellent candidate here, so that service description and composition layer can be specified independently from the binding layer.

One of the important consequences of our assumptions is that we must have one universal message format for communication between services, and the message contents
is a formula of some fixed language rather than data to be processed by an application. The second consequence is that in our approach service is composed of an application and universal interface for communication in the language.

The language of message contents is called service description and composition language. General overview of the environment, the language is supposed to describe, is the following. Client (user or an application) wants to realize a task. Services perform some operations, that is, process data (called e-documents or resources). The processing consists in the following: Given input resources, operation produces an output resource. The task specifies the properties of a final resource to be produced and delivered to a fixed place by a timeout.

Generally, the task should be realized by a composition of a number of operations performed by services. Agent is a process dedicated for a single task realization. The agent is obliged to discover, arrange, and synchronize appropriate services (whose operations are in the composition) into a workflow. Then, the workflow is executed and controlled by the agent.

It is supposed that there are service registries (called infoServices) for service publication on the one hand, and for service discovery by agents on the other hand.

There are four types of actions that can be executed in our environment: send/receive message executed by agent or service; perform operation executed by service; get resource executed by service; change state executed by agent or service.

The tasks are formulated in our description language. Generally, the language describes situations between agents, services, and resources before and after action executions. However, actions and causal relations are not described explicitly in our language. Agents and services have several mental attitudes (e.g., intentions, commitments, goals, knowledge) that describe the workflow formation process in our language.

The description language, the universal format of message and state as well as a proposal of a composition protocol constitute together the service description and composition layer. It is the crucial problem how this layer is implemented in the binding layer, i.e., how the message and resource transport is realized. The solution we have adopted to solve the problem is natural and extremely simple. The sender as well as the recipient name in the message is a URI that contains the absolute address of sender (or recipient) and the name of a transport protocol for communication (e.g., SOAP is the protocol of choice). As to the resource passing between services, we have chosen the "pull" method, that is, the service (say, service1) that wants to pass a resource to the another service (say, service0) sends the message: "The resource you are supposed to receive has the name uri0". Once the service0 received the message and got to know the name of the resource, it can download the resource by HTTP, i.e., the resource names are URLs.

The present work is based on the papers published in [1, 2, 3], where the basic ideas of our approach were introduced. Here, we present a concrete realization of these ideas in the form of formal requirements of description language and composition protocol.
4 General View of an Alternative Proposal of Description Language and Composition Protocol

Two basic components are necessary to realize our vision of service integration: description language, and composition protocol.

Description language should allow to describe not only the type of operation performed by a particular service, but also relations between agents (responsible for task realizations) and services engaged in these task realizations. Agent intentions, and service commitments are examples of these relations. The language must be open. It means that any user / programmer can add to the language new primitive concepts, i.e., new types of resources (e-documents), new relations, as well as new functions. However, this must be done according to the fixed rules (see definitions.xsd), so that the integrity of language syntax is preserved.

The composition protocol is a conversation protocol between agents, services, and service registries. During a conversation (i.e., a protocol session) messages are exchanges that may effect the state of sender / recipient. The language of message contents is supposed to be our description language, i.e., the contents is an evaluated formula of our language describing a situation between an agent, services and e-documents.

5 Requirements for Universal Message Format

The Message format is extremely simple (see message.xsd) and consists of the following items:

- Header;
  - From; sender address.
  - To; recipient address.
  - Protocol; name of protocol.
  - Version; version of protocol.
  - Session; session identifier.
  - Order; type (order) of the message in the protocol.

- Body; contents of the message.

Body contains facts, i.e., evaluated formulas. A fact is called Info (an element defined in message.xsd) and is composed of the following items: formula, time, place, signature. The meaning of Info is that formula was true at time, in place, and this was stated by the one who signed it.

6 Requirements for the Format of Agent State and Service State

We propose one universal state format for agents as well as for services (see state.xsd). The State format consists of the following items:

- Owner; the name of the owner (i.e., an agent or a service) of the state.
- Goal; agent’s task (or the type of operation performed by a service if the State belongs to the service).
– **Intentions:** lists of agent’s intentions.
– **Commitments:** a set of service’s commitments.
– **Knowledge:** the container for agent’s/service’s knowledge; it is a collection of Info elements.

Note that in our framework, agent has no commitments, and service has no intentions. However, it is also reasonable to consider agents that make commitments, as well as services having intentions.

**Goal** consists of two elements: $form_{In}$ and $form_{Out}$. Each of them contains a formula of our description language. If the state belongs to a service, the **Goal** represents the type of operation performed by the service, so that $form_{In}$ describes precondition of service invocation, whereas $form_{Out}$ describes the post condition (effect) of performing the operation by the service. If the state belongs to an agent, then $form_{In}$ is either empty or contains a formula that describes precondition for realizing the agent’s task, whereas $form_{Out}$ formula describes the agent’s task.

**Commitments** is a set of service’s commitments. A **commitment** consists of two elements: $form_{In}$ and $form_{Out}$. Each of them contains a formula of our description language. $form_{In}$ describes precondition of the commitment, whereas $form_{Out}$ describes the post condition, i.e., effect the service has committed to realize. Once a commitment is realized, it is removed from **Commitments**, however the information about the realization is stored in **Knowledge**.

**Intentions** is the element composed of the following three parts:

– **Plan** is a sequence (list) of formulas (called intentions) describing agent’s plan.
– **Workflow** is a set of intentions moved from **Plan** for which agent has already arranged commitments with services. They are supposed to form a workflow for realizing agent’s task.
– **Realized** is a set of intentions moved from **Workflow** that have been already satisfied by realization of the associated commitments.

An intention is moved from **Plan** to **Workflow** if agent has found a service that has committed to realize this intention. An intention is moved from **Workflow** to **Realized** if this intention has been already realized.

An algorithm of agent functioning can be sketched as follows.

1. Agent’s task is set as its first intention and put into **Plan**.
2. The first intention is decomposed into a sequence of intentions on the basis of an **Info** received from an infoService.
3. Agent is looking for a service that can realize the first intention from its **Plan**.
4. Once the agent has found a service that has committed to realize its current intention, the intention is moved from **Plan** to **Workflow** whereas the precondition of the service commitment is set as a new intention and is put as the last element of the agent’s **Plan**.
5. If **Plan** is an empty sequence, a workflow for task realization is completed and may be executed.
6. Once the agent gets confirmation that an intention from **Workflow** is realized, it is moved to **Realized**.
7. If Workflow is empty, then the workflow has been already executed successfully, so that the task is realized, and the agent can send the final confirmation approving the transaction performed by the workflow.

7 Requirements for Description Language

Formula of our description language is the basic component of all the data structure defined in the previous section. In order to specify the formula formally, we must define the syntax of our description language.

The description language is a simple version of the language of first order logic with types, and without negation and quantifiers. The syntax of our language is specified in formula.xsd, where a XML syntax of the language is defined. Specific primitive concepts, i.e., types, relations, and functions are introduced in properEntish.xml that defines the standard part of the language Entish. It is worth to note that these specific concepts are introduced in the same way (using the schema definitions.xsd) as another concepts can be introduced, i.e., properEntish.xml is an instance of the schema definitions.xsd.

Since XML syntax is hard to read, we also introduce more readable syntax for presentation. So that, if \( \psi \) and \( \phi \) are formulas, then (\( \psi \ or \ \phi \)), (\( \psi \ and \ \phi \)), (\( \psi \ implies \ \phi \)) are formulas. Terms and atomic formulas are defined in the usual way however, quantifiers and negation are not used. Predefined types, relations and functions (defined formally in properEntish.xml) are listed below:

Types:

- **Agent** is a primitive type; agent (i.e., element of type Agent) is a process equipped with its own state (i.e., element State defined in state.xsd). It is supposed that all essential data of the agent is stored in its state. Agent is dedicated for a single task realization. It is created when there is a task to be realized, and is terminated after the task realization or if the task can not be realized.

- **Service** is a primitive type; service (i.e., element of type Service) is a process having its own state (i.e., element State defined in state.xsd). The main service’s component is an application that processes data (e-documents). Processing e-documents may result in effecting the real world, e.g., purchasing a commodity or withdraw of some amount of money from a bank account, or just taking some physical actions like switching off/on a washing machine.

- **Time**; element of this type is a time written according to xsd:time format.

- **Token**; token (i.e., element of type Token) is an arbitrary string. It is used as value of function \( \text{token}(\text{resource}) \), (the question mark ? before a string indicates that this string is a variable). Tokens are a general way to identify resources (e-docs) at the language level. Note that our language is independent from data format of resources; the format may be arbitrary, e.g., XML, MS Word, txt, binary, and so on.

Relations:

- **timeout(?)** can be evaluated at any host. It is true if the time ?t is less or equal to the current GMT time at the host.

- \( (?x=?) \) is a polymorphic equality relation. It can be evaluated if ?x and ?y are of the same type.
- `isIn( ?resource, ?service )` states that `?resource` is in `?service`. It can be evaluated locally only by `?service`.
- `intentions(?agent)` is an atomic formula. It is evaluated only locally by `?agent`. During an evaluation it is replaced with the disjunction of all formulas from the element `Plan` of the `State` of the `?agent`.
- `formInOperationType( ?service )` is an atomic formula to be evaluated only by `?service`. During an evaluation it is replaced with the formula from the `formIn` element of `Goal` of the `State` of `?service`. The formula describes the precondition necessary for `?service` invocation.
- `formOutOperationType( ?service )` is an atomic formula to be evaluated only by `?service`. During an evaluation it is replaced with the formula from the `formOut` element of `Goal` of the `State` of `?service`. The formula describes the post condition of `?service` invocation, i.e., the result of performing the operation by `?service`.
- `formInCommitment( ?service )` is an atomic formula evaluated only by `?service`. During an evaluation it is replaced with the disjunction of formulas from `formIn` elements of all `commitment` elements of `Commitments` of the `State` of `?service`. It describes the preconditions of the commitments made by the `?service`.
- `formOutCommitment( ?service )` is an atomic formula evaluated only by `?service`. During an evaluation it is replaced with the conjunction of formulas from `formOut` elements of all `commitment` elements of `Commitments` of the `State` of `?service`. It describes the post conditions of the commitments made by the `?service`.

There is only one predefined function: `token( ?resource )` that returns token (an element of type `Token`) determined for `?resource` by the service that expects it to be delivered as its input resource.

A formula is syntactically valid (well constructed) if it is constructed according to the syntax specified in `formula.xsd`, and `properEntish.xml`, and if the names of types, relations, and functions occurring in the formula have been already defined in XML documents (instances of `definitions.xsd`), and the documents are available by HTTP. And the names are used according to their specifications in the documents.

8 Requirements for Composition Protocol

9 Requirements for Abstract Architecture of Agent and Service

10 Requirements for Biding Layer (if any)

11 Usage Scenarios

References

This article was processed using the \LaTeX\ macro package with LLNCS style