Available online at www.sciencedirect.com
SCIENCE DIRECT
Information and Software Technology 48 (2006) 441–455
INFORMATION AND SOFTWARE TECHNOLOGY
www.elsevier.com/locate/infsof

Towards an ontology-based approach for specifying and securing Web services

Z. Maamar[a], N.C. Narendra[b,*], S. Sattanathan[c]

[a]Zayed University, Dubai, U.A.E.
[b]IBM Software Labs India, Bangalore, India
[c]National Institute of Technology Karnataka, Surathkal, India

Received 7 December 2004; revised 13 April 2005; accepted 11 May 2005
Available online 23 June 2005

Abstract

With the increasing popularity of Web services and increasing complexity of satisfying needs of users, there has been a renewed interest in Web services composition. Composition addresses the case of a user request that cannot be satisfied by any available Web service, whereas a composite service obtained by integrating Web services might be used. Because Web services originate from different providers, their composition faces the obstacle of the context heterogeneity of Web services. An unawareness or poor consideration of this heterogeneity during Web services composition and execution result in a lack of the quality and relevancy of information that permits tracking the composition, monitoring the execution, and handling exceptions. This paper presents an ontology-based approach for context reconciliation. The approach also focuses on the security breaches that threaten the integrity of the context of Web services, and proposes appropriate means to achieve this integrity.

Keywords: Web services; Composition; Context; Ontology; Security

1. Introduction

A Web service is an accessible application that other applications and humans can discover and trigger to satisfy multiple needs (e.g. weather forecasts) [2]. One of the strengths of Web services (also called services in this paper) is their capacity to be composed into high-level business processes known as composite services. Composition primarily addresses the situation of a user request that cannot be satisfied by any available service, whereas a composite service obtained by integrating available services might be used [3].

Current standards for Web services (e.g. WSDL, SOAP and UDDI [20]) revolve around XML to achieve platform-independence features. Therefore, Web services composition is only achieved at the level of message interactions. This is by far not sufficient, as composition needs also to be conducted at the level of message semantics [1]. The semantic composition guarantees that the information exchanged between Web services is clearly understood. The need for a common semantics is intensified when Web services, which originate from different providers, have to take part in the same composition. If the semantic heterogeneity is not dealt with properly, it constitutes a serious obstacle to Web services composition [17]. To tackle this obstacle the Web services of a composite service have to initially agree on the information they will exchange before further interactions. Our response to the semantic heterogeneity uses ontologies [12] and consists of making Web services dynamically bind to the ontology that is suitable for the situation in which they operate. By situation, we mean the application domain of a composition such as vacation planning and conference organization. Ontology is a specification of the concepts of an application domain and the semantic proximity existing between these concepts.

Besides the information disparity challenge, further challenges could hinder Web services composition such as which businesses have the capacity to provision Web services, when and where the provisioning of Web services

* Corresponding author.
E-mail addresses: zakaria.maamar@zu.ac.ae (Z. Maamar), narendra@in.ibm.com (N.C. Narendra), ss_nitk@yahoo.co.in (S. Sattanathan).

0950-5849/$ - see front matter © 2005 Elsevier B.V. All rights reserved.
doi:10.1016/j.infsof.2005.05.004

occurs, and how Web services from independent providers coordinate their activities so that conflicts can be avoided. To face some of these challenges, we recommended considering the context of the composition and execution of Web services [15]. Context is the information that characterizes the interactions between humans, applications, and the surrounding environment [5,10]. From a Web services perspective, we defined context as a set of common meta-data about the current execution status of a Web service and its capability of collaborating with peers, possibly enacted by distinct providers. For example a Web service assesses its execution status before it agrees on participating in a composition.

Composition of Web services is a very active area of R & D [16]. However very little has been accomplished to date regarding the integration of ontologies and context into Web services composition approaches. Multiple obstacles still hinder this integration such as Web services act as passive components that cannot be embedded with context-awareness mechanisms, existing approaches for Web services composition (e.g. BPEL, WSFL) typically facilitate choreography only, while neglecting context of users, services, and computing resources, lack of appropriate techniques for formalizing contexts of Web services using ontologies, and last but not least scarcity of techniques for protecting contextual information exchanged between Web services, from being intercepted and altered. In this paper, we present our work on using ontologies for specifying and securing contexts of Web services. Our work, which differs from contextualizing ontologies [24], takes advantage of the research findings of the Semantic Web community and its specification language Ontology Web Language-based Web Service Ontology (OWL-S [11], formerly DAML-S). Our objective is to develop a language similar to OWL-S for managing contexts of Web services. We refer to this language as OWL-C standing for Ontology Web Language-based Context Ontology. The elements that are highlighted in this paper and constitute the foundations of OWL-C are the following:

- Web services of type instance are obtained out of Web services (details on Web services instantiation are given in [15]).
- Web services are subject to multiple constraints such as maximum number of Web service instances to make available for concurrent use, and strategy for selecting the ontology.
- Prior to agreeing on participating in a composite service, a Web service assesses its ongoing participations through awareness mechanisms. To perform this assessment, the three types of services (i.e. composite service, Web service, and Web service instance) are each associated with a context of type C-context, W-context, and I-context, respectively.

The rest of this paper is organized as follows. Section 2 presents our context reconciliation approach using ontology. The foundations of OWL-C and a running example on using OWL-C are also presented in this section. Section 3 introduces our strategy for securing contexts during interactions of Web services. Section 4 overviews the status of the proof-of-concept prototype of specifying and securing Web services using ontologies. Section 5 presents related work. Finally, we draw our conclusions in Section 6. It should be noted at this stage that the mechanisms for discovering the component Web services of a composite service, while important, are beyond this paper's scope. UDDI-based mechanisms can for instance be used.

2. Ontology-based context reconciliation

Ignoring the problem of context heterogeneity of Web services has side-effects on the normal progress of their composition. These side-effects are multiple such as adopting the wrong strategy for selecting a component Web service (e.g. favoring execution-cost criterion over reliability criterion instead of the opposite), delaying the triggering of some urgent component Web services, or poorly assessing the exact execution status of a Web service (e.g. service in blocked status while being thought in running status). In the following, we present the foundations on which OWL-C is built upon. Also discussed in this section, the principles that regulate the operation of Web services during context detection, assessment, and reconciliation.

2.1. Overview

Fig. 1 conceptualizes the approach of dealing with the context heterogeneity of Web services. The approach revolves around Web service instances binding to appropriate ontologies. By binding, we mean the compliance of a Web service instance with a specific ontology for data-management needs when it interacts with peers. The creation of Web service instances is subject to accepting the invitations of participation that originate from composite services to Web services (a Web service can always reject an invitation of participation in a composite service; see [15] for more details). Upon invitation acceptance, the composite service informs the multiple component Web services about the ontology that their respective Web service instances need to adopt. The ontology is related to the application domain in which the composition of Web services occurs (e.g. travel domain). We assume that ontologies are available in a repository that administrators maintain (Fig. 1). In the rest of this paper, we also assume that the specification of the component Web services is based on service chart diagrams [14]. A service chart diagram extends a state chart diagram with emphasis on the context surrounding the execution of a Web service rather than only the states that a Web service takes. Our selection of service chart diagrams is motivated by the value-added of state chart diagrams to Web services composition as reported in [26].

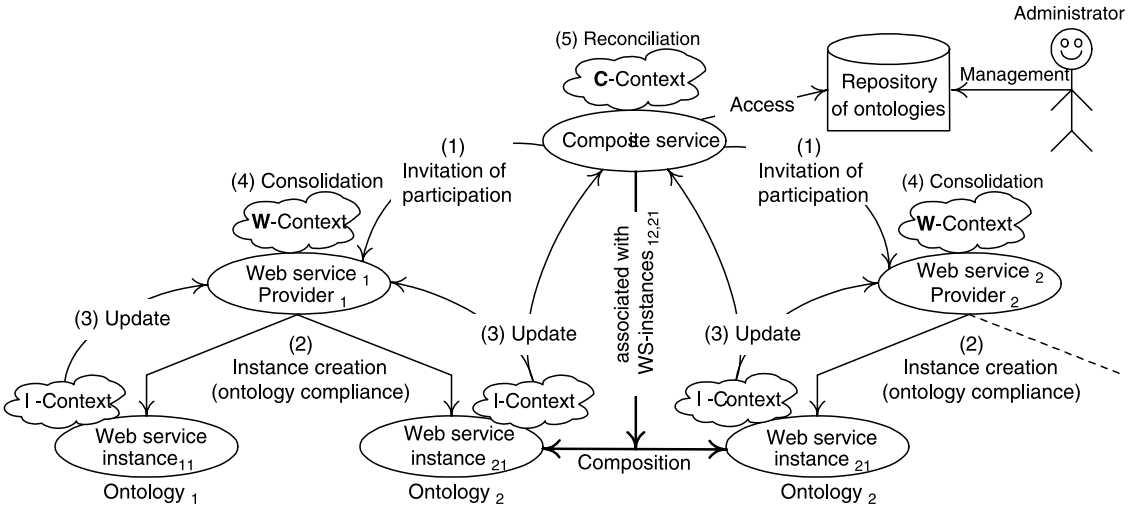


Fig. 1. Context and ontology use in Web services composition.

A service chart diagram represents a Web service from five perspectives [14]: state, flow, performance, business, and information. We focus on the information perspective since it clearly states the ontology that a Web services has to use. The information perspective represents the pre- and post-execution data that are exchanged between the Web services according to a specific chronology. For example, *flight-reservation* Web service needs 'date of departure', 'date of return', and 'destination city' as pre-data. After processing, this service submits the data with a confirmation of the reservation as post-data to *hotel-booking* Web service. Both services are aware of the meaning and structure of these data.

Since a composite service is made up of several component services, its underlying process model is specified as a state chart diagram where states correspond to the service chart diagrams of the Web services, and transitions connect these diagrams through events, conditions, and variable assignment operations. Fig. 2 represents a composite service as a state chart diagram. The composite service is about vacation logistics. Each component Web service has a service chart diagram: *sightseeing* (SI), *weather* (WE), *shopping* (SH), and *transportation* (TR). These diagrams are connected through transitions; some of them have constraints to satisfy (e.g. [confirmed(hot weather)]).

2.2. Operation of Web services

According to the Web services instantiation principle [15], a Web service can be part of multiple composite services through the multiple service instances it deploys. A Web service is instantiated each time it receives from a composite service a request of participation in a composition. Before the instantiation takes place, several aspects related to the Web service are checked. First, the number of Web service instances currently running vs. the maximum number of Web service instances that can be simultaneously run (i.e. instances of the same Web service). Second, the execution status of each Web service instance that is deployed and part of a composite service. Third, the execution progress of the preceding Web service instances per Web service instance to be deployed. This progress is required in case of data or control dependency between the different service instances. We recall that Web service instances originate from different Web services, which accentuates the semantic heterogeneity challenges. Finally, the time that the composite service would like having a Web service instance made available for invocation vs. the time it would be possible for the Web service to have a Web service instance made available for invocation. It happens that a composite service has deadlines to meet, thus, it has to ensure that the Web service instances are deployed on time.

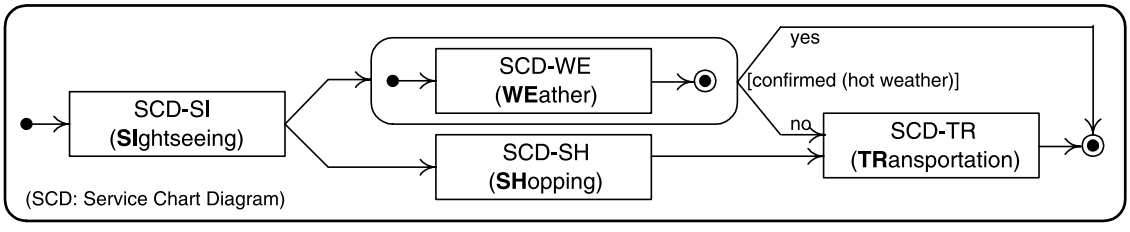


Fig. 2. Specification of a composite service as a state chart diagram.

To monitor the deployment of the specification of a composite service from a temporal perspective (i.e. what occurred, what is occurring, and what might occur), the composite service needs a structure on which a specific processing is performed. This structure is called C-context (context of composite service). The same argument in favor of contexts of Web services (W-context) is adopted. Web services take part in different composite services and require structures to support their tracking per composite service. Web services rely on their respective W-contexts before they make any decision of participation in a new composite service. The decision-making process of a Web service needs to be fed with details that directly originate from its Web service instances. The Web service instances rely on their respective I-contexts to collect and submit details to the W-contexts of their Web services (Fig. 1).

Because of the heterogeneity of the Internet, it is unlikely that a certain context provider would deliver all types of contextual information [21,27]. Therefore, contexts will have a content of different granularities and structures as well. To manage this heterogeneity, contexts of Web services are subject to two operations referred to as consolidation and reconciliation. In Fig. 1, numbers between brackets represent the chronology of these two operations.

- Consolidation at the level of Web services: each time a Web service accepts an invitation of participation in a composite service (Fig. 1-(1)), a Web service instance and I-context are created (Fig. 1-(2)). The transfer of details from the I-contexts of the same Web service instances to the W-Context of their associated Web service is featured by a consolidation of these details before this W-context is updated (Fig. 1-(3,4)). Consolidation means the combination of details that stem from a lower level to a higher level. Once the consolidation is completed, a Web service can determine for each of its Web service instances the following: execution status, the actions it has performed, and expected completion execution-time so that the Web service can commit additional Web service instances for the benefit of other composite services.
- Reconciliation at the level of composite services[1]: since the component Web services of a composite service have multiple providers, the definition of their respective W-contexts (and obviously the definition of the I-contexts of their service instances) varies in terms of structure and content (e.g. different numbers of arguments, different names of arguments). The transfer of details from the I-contexts of the Web service instances to the C-context of a composite service is featured by a reconciliation of these details before the C-context is updated (Fig. 1-(3,5)). For example it occurs that the I-context of a Web service instance of a composite service has *location of execution* argument, whereas the I-context of another Web service instance of this composite service has *site of execution* argument. During reconciliation, *execution site* and *execution location* are considered the same. To ensure that the composite service recognizes the differences between the arguments of contexts, it refers to ontology of context. *Execution location* or *execution site* mean here the computing resource on which the performance of a Web service instance occurs.

In Fig. 1, ontologies during Web services composition participate in two operations. The consolidation operation concerns the data that the Web service instances of a composite service exchange. The information perspective of a service chart diagram indicates the compliance of a Web service with a specific ontology. The reconciliation operation concerns the data that the contexts of Web service instances submit to the contexts of composite services. Before updating the contexts, their heterogeneity needs to be dealt with. This is done through OWL-C, a dedicated language for specifying contexts of Web services. Backing the importance of OWL-C, Wang et al. discuss the motivations of developing context models based on ontologies [8]:

- Knowledge sharing: the use of a context ontology enables different computational entities to have common concepts about context when they engage in interactions with other entities. We argued that component Web services of a composite service will highly come from different origins.
- Logic inference: based on ontology, context-aware computing can exploit various existing logic reasoning mechanisms to infer high-level, conceptual context from low-level, raw context, and to check and solve inconsistent context knowledge due to imperfect sensing. We argued that because of the three types of services (composite service, Web service, and instance service) associating each type with a specific type of context constitutes a normal practice. In addition, this separation of types enables generating contextual information at different levels of abstraction.
- Knowledge reuse: by reusing well-defined Web ontologies of different domains (e.g. temporal and spatial ontology), large-scale context ontology can be composed without starting from scratch. Plus, a context ontology facilitates the sharing, management, and inference of a given context.

2.3. Foundations of OWL-C

Fig. 1 depicts three levels of abstraction, where each level identifies a type of service: Web service instance, Web service, and composite service. Contexts of Web service instances have the fine-grained content, and contexts of composite services have the coarse-grained content. The content of I-context updates respectively the content of W-context after consolidation and the content of C-context

[1] That is how context heterogeneity of Web services is addressed.

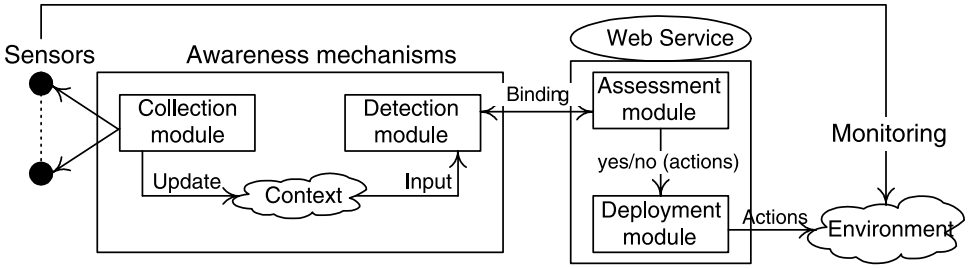


Fig. 3. Connection between Web service and context.

after reconciliation, too. In the following, we discuss the foundations of OWL-C, a language for defining the structure and content of context. It is stated that for a good language for describing services, multiple requirements have to be fulfilled such as editable, semantically expressive, automatically comparable, and flexible. We consider that a language for describing contexts has to fulfill the same requirements. In addition, by taking OWL-S as a starting point for defining OWL-C a compatibility with the existing standards for Web services is achieved.

Context specification using OWL-C includes two parts. The first part is about the arguments that define the structure of context. The second part is about the capabilities associated with context. In the first part, context is an additional argument of a Web service. Web services regularly exhibit their arguments (e.g. *identifier, execution cost*) to the external environment (through UDDI). Because context is a multi-argument structure, OWL-C assists in the stipulated semantics of these arguments. As a result various parties agree now on a common representation of the content of the context of Web services. With regard to context capabilities, a service needs to be embedded with awareness mechanisms, so that it can take advantage of the information that context caters. These mechanisms gather any contextual raw data from sensors and detect any change in the environment. A change needs to be evaluated by the Web service through an assessment module, so that it takes appropriate actions through a deployment module (Fig. 3).

Fig. 4 represents the context ontology as a directed graph. Nodes represent the ontology's concepts and edges represent relationships between concepts. Edges are annotated with cardinality. For illustration purposes, some of the concepts and relationships of the ontology context of Web services are discussed below:

- Context is the core concept of the ontology. A context type is associated with a specific type of service namely Web service, Web service instance, or composite service.
- Label (1:*n*) on edge (Web service → Web service instance) indicates that a Web service has one or more Web service instances. In addition, label (1:1) on edge (Web service instance → composite service) indicates that a Web service instance belongs to one composite service.
- Edges (Web service → constraint) and (Web service instance → constraint) illustrate respectively the constraints on a Web service (e.g. maximum number of Web service instances that can be created) and the constraints on a Web service instance (e.g. ontology of the domain that the instance has to comply with).
- Edge (context → type) indicates that a context has exactly one type namely I, W, and C. Each context type has a set of sensors, purpose, description, name, and a set of

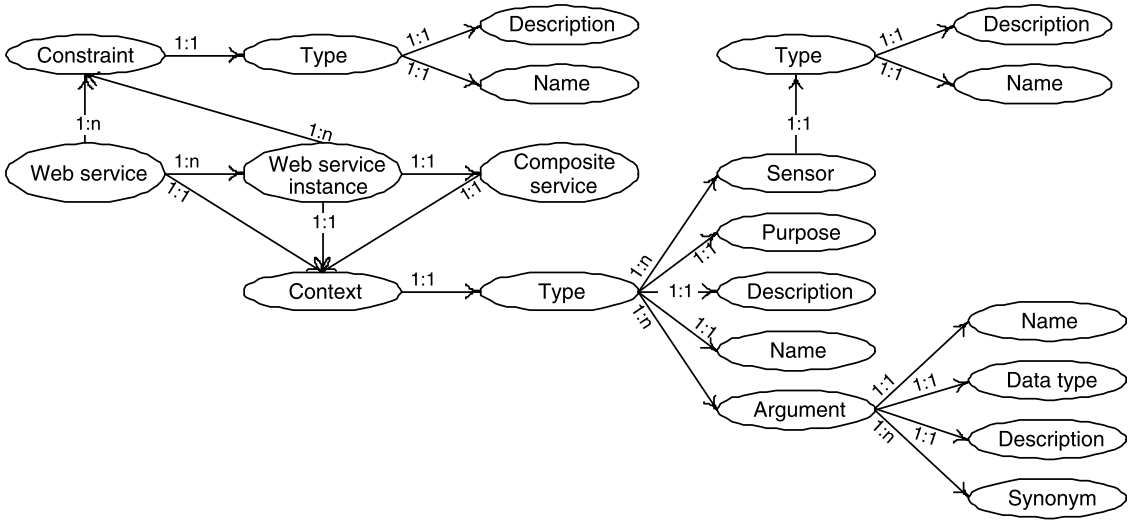


Fig. 4. Ontology-based description of context of Web services.

Table 1
Some arguments of I-context

Label: corresponds to the identifier of the service instance
Status: informs about the execution status of the service instance (in-progress, suspended, aborted, and terminated)
Previous service instances: indicates if there were service instances before the service instance. These service instances have already completed their execution
Next service instances: explicates whether there will be service instances to be executed after the service instance. The Web services of these service instances will be called for execution
Regular actions: illustrates the actions the service instance performs
Begin time and end time (expected and effective): informs when the execution of the service instance has started, and when this execution is expected to terminate/has effectively terminated. End-time expected is user dependent whereas end-time effective is execution dependent
Reasons of failure: informs about the causes that fail the execution of the service instance
Corrective actions: illustrates the actions the service instance performs if the execution fails
Date: identifies the time of updating the arguments above

Table 2
Some arguments of W-context

Label: corresponds to the identifier of the Web service
Number of service instances allowed: corresponds to the maximum number of service instances that can be created from the Web service
Number of service instances running: corresponds to the number of service instances of the Web service that are currently running
Next service instance availability: corresponds to when a new service instance of the Web service will be made available
Status/Service instance/Composite service: corresponds to the status of each service instance of the Web service that is deployed (based on status argument of I-context)
Date: identifies the time of updating the arguments above

Table 3
Some arguments of C-context

Label: corresponds to the identifier of the composite service
Previous Web services: indicates which Web services of the composite service have been executed with regard to the current Web services
Current Web services: indicates which Web services of the composite service are currently under execution
Next Web services: defines which Web services of the composite service will be called for execution with regard to the current active Web service(s)
Begin time: informs when the execution of the composite service has started
Status/Web service instance: corresponds to the status of the Web service instance of the composite service that is deployed (based on status arguments of I-context)
Date: identifies the time of updating the arguments above

arguments. An argument has a name, data type, description, and synonyms.

As mentioned earlier, a set of arguments constitute the structure of context. The number of arguments varies depending on the context type. We discuss below the arguments of the structure that can populate each context type. First of all, we start with the structure of I-context of a Web service instance (Table 1): *label*, *status*, *previous service instance*, *next service instance*, *regular actions*[2], *begin time*, *end time (expected and effective)*, *reasons of failure*, *corrective actions*[3], and *date*.

For the structure of W-context of a Web service, which will be built upon the I-contexts of its Web service instances, Table 2 suggests some arguments: *label*, *number of service instances allowed*, *number of service instances running*, *next service instance availability*, *status per service instance per composite service*, and *date*.

For the structure of C-context of a composite service, which will be built upon the W-contexts of its component Web services, Table 3 suggests some arguments: *label*, *previous Web services*, *current Web services*, *next Web services*, *begin time*, *status per Web service instance*, and *date*.

In the structure of a context, an argument has a name, description, type, and a list of synonyms. Name and description arguments are already explained in Tables 1–3. In the following, we provide examples on the type and set of synonyms of an argument using Table 4. In I-context *status* is the name of an argument, its description is in Table 1, its data type is string, and *stage* and *state* constitute both its list of synonyms. A similar description applies to all arguments of W-context and C-context.

OWL-S organizes the description of a Web service along three categories [6]: profile, process model, and grounding. While developing OWL-C, we aimed at making sure that the description of a context happens along the same categories. The profile describes the arguments and capabilities of context (what does the context require and provide?). The process model suggests how context collects raw data from sensors and detects changes that need to be submitted to the service.

Table 4
Examples of arguments of contexts

Context	Argument	Value
I-context	Name	Status
	Description	Table 1
	Data type	String
	List of synonyms	Stage, State
W-context	Name	Number of instances allowed
	Description	Table 2
	Data type	Integer
	List of synonyms	Maximum number of instances authorized
C-context	Name	Begin time
	Description	Table 3
	Data type	TIME
	List of synonyms	Start time

[2] Regular actions argument of I-Context illustrates the actions that a Web service instance has executed according to a certain context. This helps track the execution trace of the Web service instance.

[3] Corrective actions argument is in relation to regular actions argument.

Finally, the grounding defines the bindings (protocol, input/output messages, etc.) that make context accessible to a service. Because context is an argument of the structure of a service, the context profile can be used during the selection of Web services for composition. Discovering Web services using context is important since Web services are provisioned according to specific preferences such as user location (service selection in a context-sensitive manner) [22].

2.4. On using OWL-S

To illustrate the feasibility of OWL-S, a running example is provided. Imagine a travel-agent service (to be mapped onto a composite service) that has to put together an itinerary for a tourist. Two of the most significant component services of this composite service, would be *taxi booking* (mapped onto Web service$_1$ of provider$_1$) and *hotel booking* (mapped onto Web service$_2$ of provider$_1$ and Web service$_2$ of provider$_2$). The instantiation process of both Web services occurs as follows: (i) a hotel-booking service instance for a 5-star hotel (Web Service Instance$_{21}$); and (ii) two taxi-booking service instances, one that books a taxi from the airport to the hotel (Web Service Instance$_{12}$), and one that books a taxi that takes the tourist around the city (Web Service Instance$_{11}$).

Some arguments of the I-context of the airport taxi service instance could be (as per Table 1):

- Previous service instances: airport-taxi service instance could have been preceded by another service instance, which finds out the arrival details of a tourist.
- Regular actions: airport-taxi service instance executes some actions once a tourist is picked up from the airport by informing for example the central reservation system. The composite service is also informed about the progress of airport-taxi service instance so that it can update its C-context (after reconciliation).
- Begin and end time: airport-taxi service instance informs its respective Web service when its execution (i.e. ride from the airport to the hotel) has started and terminated. The Web service is also informed about the progress of airport-taxi service instance so that it can update its W-context (after consolidation).

Similar information would also be sent out from the I-context of the tourist-taxi service to the composite service. With regard to the W-context of taxi-booking service, this one would maintain the data regarding the running airport and tourist-taxi service instances, and also the status of all running service instances (as per Table 2). With regard to the C-context of travel-agent service, this one would maintain the information regarding the taxi and hotel-booking services and service providers, and also the service instance data received from the I-contexts via reconciliation (as per Table 3).

3. Security of Web services contexts

In Section 2.4, we strengthened the necessity of dealing with the content heterogeneity that features the contexts of Web services. An unawareness or poor consideration of this heterogeneity result in a lack of the quality and relevancy of the information that is deemed appropriate for tracking composition, monitoring its execution, and handling its exceptions. In what follows, we focus on the security breaches that threaten the integrity of the contexts of Web services, and proposes appropriate means to achieve this integrity.

3.1. Current situation

In Fig. 1, the three contexts have a double role: support tracking the execution status of each Web service instance of a composite service, and support deploying the appropriate corrective actions in case of exceptions. Besides the risk of intercepting SOAP messages that are passed between a Web service invoker and a service provider in a Web services transaction scenario, altering contextual information during its transfer between the various parties (composite services, Web services, and Web service instances) has negative consequences on the normal progress of a composition. Because current security mechanisms for Web services interactions are pre-defined, they, unfortunately, do not take advantage of the contextual information that might help select the appropriate security mechanism. In addition, these security mechanisms do no distinguish between the specific requirements that each type of service (composite, Web, or instance) has when it comes to securing its contextual information.

In Fig. 1, the context model associated with Web services composition spreads over three levels. This model, however, does not integrate any security measures during context exchanges. Because composition involves many Web service instances, communication between the three levels happens according to different patterns: between service instances of a composite service, from service instances to Web services and vice-versa, from service instances to composite services and vice-versa, and (iv) from Web services to composite services and vice-versa. We categorize the threats that affect Web services into three categories (adapted from [28]): identity threats where an attacker impersonates a legitimate Web service or user; content-borne threats where an attacker attacks Web services directly (e.g. buffer overflow); and operational threats that render Web services unusable (e.g. denial of service attacks).

3.2. Proposed security model

The encryption of a content to be submitted over a transport middleware is the most widely means that permits facing interception and alteration threats. Digital certificates, unique identifiers, and heuristic-based protections are used

during encryption. However, these techniques are too complex for Web services security, could be unwieldy, and affect negatively the execution performance of Web services. Our model for securing Web services interactions during context exchange features three security contexts: ISec-context for security of Web service instance, WSec-context for security of Web service, and CSec-context for security of composite service. These security contexts are defined along with the regular service contexts (i.e. I-, W-, and C-context). A security context has a major role in highlighting the security strategy that a service adopts. Any change in this strategy is automatically reflected in the security context so that other peers are aware of the change in case they have to comply with it. Stating the security strategy in a security context is backed by Kagal et al. [18], who mention that a Web service could clearly indicate what it can perform[4] and what it requires from invoker such as authentication and use of XML for communication. The distinction between service context and security context gives better flexibility in the management of the aspects that each type of context is concerned with. A service context focuses on the changes that apply to a service (whether composite, Web, or instance) like availability and commitment, whereas the security context focuses on the strategy of securing the interactions of services during data-context exchange.

At the Web-service instance level, the primary use of I-context is to track its execution status (Table 1). If a service instance was subject to threats that attempted altering its context, this should be reported in its respective ISec-context so that corrective actions are planned for the forthcoming service instances. Each Web service instance has its copy of ISec-context. Table 5 contains samples of the arguments that populate ISec-context.

At the Web-service provider level, whenever a Web service receives a request of participation in a composition, it validates its current capabilities using C-context and checks its security requirements. If both are satisfactory, the Web service creates a new service instance. Table 6 contains samples of the arguments that populate WSec-context.

At the composite-service level, the C-context traces the execution of the composite service and its respective component service instances, and tries to identify potential conflicts (in terms of resources, shared variables, shared log files) between these service instances. The CSec-context ensures that the essential security property of non-repudiation of the messages sent and received by the composite service during its interactions with Web service providers and Web service instances. This can be achieved by maintaining message logs along with the identities of senders and recipients. Table 7 contains samples of the arguments that populate CSec-context.

Table 5
Some arguments of ISec-context

Label: identifies a service instance
Signature: establishes the identity of the service instance so that messages to other components (instance, Web service, composite service) are identified
Security mechanism: sets the encryption/decryption mechanism needed for authenticating messages received from other components
Security status: Indicates the status of authenticating the received message in terms of success or failure
Security violation: indicates the type of security violation that a message was subject to (applies when security status is failure)
Corrective actions: illustrates the actions that the service instance takes when security status is failure
Date: last time of update of the above security arguments

Table 6
Some arguments of WSec-context

Label: identifies a Web service
Signature: establishes the identity of the Web service so that messages to other components (instance, composite service) are identified
Security mechanism: sets the encryption/decryption mechanism needed for authenticating messages received from other components
Security status: Indicates the status of authenticating the received message in terms of success or failure
Security violation: indicates the type of security violation that a message was subject to (applies when security status is failure)
Corrective actions: illustrates the actions that the Web service takes when security status is failure
Security status/service instances: indicates the security status of the interactions for each completed/failed service instance of the Web service. This is obtained from security-status argument of ISec-context. Date: last time of update of the above security arguments

Table 7
Some arguments of CSec-context

Label: identifies a composite service
Signature: establishes the identity of the service so that messages to other components (instance, Web service) are identified
Security mechanism: sets the encryption/decryption mechanism needed for authenticating messages received from other components
Corrective actions: illustrates the actions the composite service takes in case security status is failure
Security per previous Web service instances: indicates the encryption/decryption mechanisms of the Web service instances of the composite service that have already been executed
Security for current Web service instances: Indicates the encryption/decryption mechanisms of Web service instances that are currently under execution
Security per next Web service instances: Indicates the encryption/decryption mechanisms of Web service instances that will be called for execution
Date: last time of update of the above security arguments

[4] For illustration purposes, if the security mechanism of a Web service instance uses the Blowfish algorithm (www.schneier.com/paper-blowfish-fse.html) in its interaction, then all messages sent to this service instance from either its Web service, other service instances, or composite services, should be encrypted using this algorithm. This, of course, assumes that each Web service instance contains access to all the encryption mechanisms needed by other Web service instances.

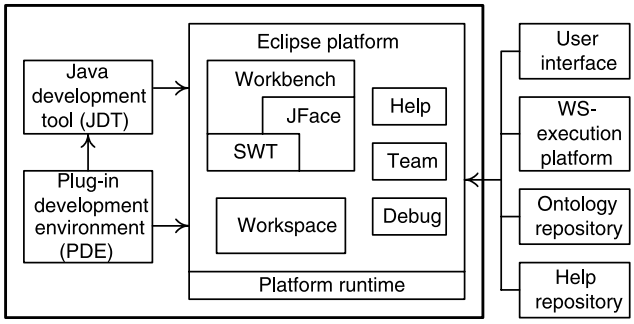


Fig. 5. Architecture of the proof-concept prototype.

4. Implementation

We report on the status of our prototype that demonstrates the feasibility of using ontologies for specifying and securing contexts of Web services. The prototype has a set of plug-ins that runs on top of Eclipse (www.eclipse.org). Eclipse is a platform-independent, open, and extensible workbench, and provides well-designed and well-documented extension points for developers to build domain-specific applications. These plug-ins can be integrated within the workbench without the technical limitations imposed by most proprietary development environments (www.phpeclipse.de/tiki-index.php?page=About+Eclipse).

In the prototype, four plug-ins are developed: *user interface*, *WS-execution platform*, *ontology repository*, and *help*. Fig. 5 illustrates the way these plug-ins connect to Eclipse. The user-interface plug-in extends the workbench in terms of perspective, wizards, views, and editor. The WS-execution platform plug-in extends the workspace in terms of project nature (i.e. context-based Web services) and builder (i.e. context assessment, validation, and reasoning). The ontology repository plug-in stores and retrieves the context ontologies by extending them. Finally, the help plug-in provides the necessary documentation for using the proof-of-concept. We use Eclipse PDE (Plug-in Development Environment) for developing these plug-ins, and SWT (Standard Widget Toolkit) and JFACE (User Interface tool kits) classes for developing user interfaces.

The class diagram that represents the integration of context into Web services and security is presented in Appendix A. The connection that exists between OWL-C and OWL-S is represented in Fig. 6. Next to information about capabilities, access methods, and protocol of a Web service, a new section is added to OWL-S, which contains context information about the service and its security means (depicted in yellow). Basically, the OWL-S document of W-context will be defined as presented in Appendix B.

For prototyping requirements, we have created Book Finder and Book Payment Web services, as well as Book Purchase composite service. Book Finder service identifies the details about a given book, and Book Payment service performs payment related operations (e.g. credit card verification, account debiting). Book Purchase (noted Book Service in the various interfaces) composite service integrates these Web services sequentially. We have set the limit of Web service instances of Book Finder and Book Payment to 3 and 1, respectively. When an instantiation request for either service is received after its respective limit has been reached, the prototype displays an error message, and the service is not instantiated.

4.1. Context consolidation

As reported earlier, consolidation happens at the level of Web services and means the combination of details that stem from a lower level (Web service instances) to a higher level (Web services). Fig. 7-(a) presents the initial values of the W-context parameters of *Book Finder*. In this figure, the focus is on *InstanceRunning* parameter (highlighted in green). After the acceptance of two instantiation requests, the consolidated version of W-context shows two Web service instances under execution (Fig. 7-(b)). When one of the instances completes its execution successfully, the number of running instances drops to 1 (Fig. 7-(c)). *InstanceAllowed* parameter corresponds to the maximum number of service instances that a Web service can concurrently deploy.

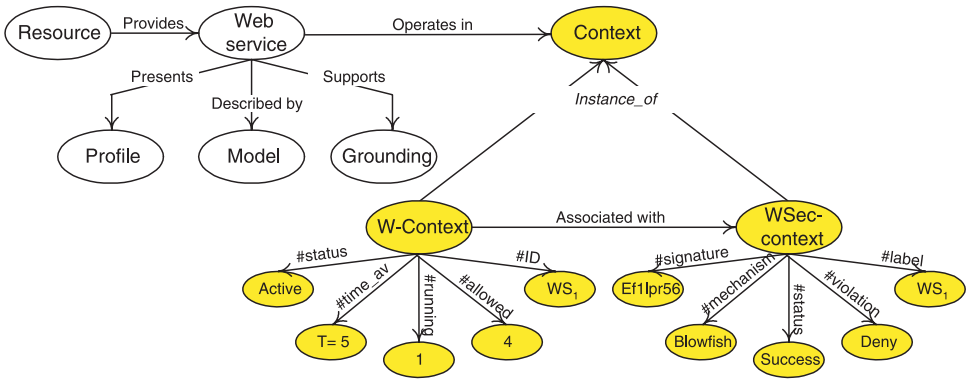


Fig. 6. Extensions to OWL-S ontology.

(a)

W-Context Parameters	Values
Label	Book_Finder
InstanceAllowed	3
InstanceRunning	0
NextInstanceAvilability	true
Status	Active

(b)

W-Context Parameters	Values
Label	Book_Finder
InstanceAllowed	3
InstanceRunning	2
NextInstanceAvilability	true
Status	Active

(c)

W-Context Parameters	Values
Label	Book_Finder
InstanceAllowed	3
InstanceRunning	1
NextInstanceAvilability	true
Status	Active

Fig. 7. Consolidation of contexts.

4.2. Context reconciliation

As reported earlier, reconciliation happens at the level of composite services, since the component Web services of a composite service have multiple providers, and the definition of their respective W-contexts (and obviously the definition of the I-contexts of their service instances) varies in terms of structure and content (e.g. different numbers of arguments, different names of arguments).

A part of the reconciliation as supported by the prototype is shown in Fig. 8. Fig. 8-(a,b) shows the initial status of both the W-context of *Book Finder* and the C-context of *Book Purchase* after *Book Finder* accepts the request of *Book Purchase*. *PreviousWebService*, *CurrentWebService*, and *NextWebService* parameters are significant for the demonstration. It can be seen for instance that *Book Purchase* will sequentially execute *Book Finder* and *Book Payment*. In Fig. 8-(b), *Book Purchase* is under execution whereas *Book Payment* is expected to be initiated upon completion of this execution. Fig. 8-(d) presents the I-context of *Book Payment* service instance (highlighted in green). This instance has a waiting status, i.e. waiting for the completion of *Book_Finder_Service_Instance_1*. Once the execution of this instance is over (Fig. 8-(c)), *Book_Payment_Instance_1* will be changed to active (Fig. 8-(e)). The appropriate parameters of the C-context of *Book Purchase* are also updated according to the execution success of its component (Fig. 8-(f)).

5. Ongoing work: supporting adaptation of Web services

Given the dynamic nature of the environments in which Web services operate, their adaptation during execution is deemed appropriate. As part of our ongoing work, we are studying Web services adaptation and analyzing existing approaches based on workflow solutions (e.g. [3,14]) to achieve this adaptation.

5.1. Motivation and proposed solution

The execution of Web services, like any other program, is expected to happen according to a predefined specification technique. In this paper, we adopted state/service chart diagrams to conduct this specification (Fig. 2). Besides both diagrams, a composition specification can be modeled as a workflow where tasks map onto Web services.

(a)

W-Context Parameters	Values
Label	Book_Finder
InstanceAllowed	3
InstanceRunning	0
NextInstanceAvilability	true
Status	Active

(b)

C-Context Parameters	Values
Label	Book Service
Status	Active
PreviousWebService	Nil
CurrentWebService	Book Finder
NextWebService	Book Payment

(c)

I-Context Parameters	Values
Label	Book_Finder_Service_Instance_1
Status	Active
PreviousServiceInstance	Nil
NextServiceInstance	Book_Payment_Instance_1
RegularAction	Finding a Book
ReasonsOfFailure	Nil
CorrectiveAction	Nil

(d)

I-Context Parameters	Values
Label	Book_Payment_Instance_1
Status	Waiting
PreviousServiceInstance	Book_Finder_Instance1
NextServiceInstance	Nil
RegularAction	Getting Payment for Book
ReasonsOfFailure	Nil
CorrectiveAction	Nil

(e)

I-Context Parameters	Values
Label	Book_Payment_Instance_1
Status	Active
PreviousServiceInstance	Book_Finder_Instance1
NextServiceInstance	Nil
RegularAction	Getting Payment for Book
ReasonsOfFailure	Nil
CorrectiveAction	Nil

(f)

C-Context Parameters	Values
Label	Book Service
Status	Active
PreviousWebService	Book Finder
CurrentWebService	Book Payment
NextWebService	Nil

Fig. 8. Reconciliation of contexts.

Adaptation of Web services execution is needed not only in case of execution failures, but also for any other reason that changes the context of a service whether instance, Web, or composite. Some reasons could be the insertion/deletion of a Web service, and modification in a Web service execution arguments. We are investigating the suitability of the 3-tier adaptive workflow model of [4] for Web services adaptation:

- Adaptation at the workflow-instance level: only the workflow instances need to be modified due to a potential improved efficiency.
- Adaptation at the workflow-schema level: the workflow definition itself needs to be modified, causing major changes to the subsequent workflow instances. However, the workflow instances that are already under execution, are not affected by this modification.
- Adaptation at the planning/goal level: the goals of the workflow execution may have to be changed, necessitating radical changes at the workflow schema and instance levels.

A natural mapping of the above 3-tier adaptive workflow model onto the 3-level context model is possible. Indeed, one of the main uses of context is during adaptation since changes can be detected through awareness mechanisms. We advocate that the following information would typically need to be maintained during workflow definition and execution at the level of the different contexts.

C-context	Two types of workflow information regarding the composite service, need to be stored here: Goals of the workflow, along with a version number that needs to be updated during adaptation The version numbering is needed for tracking the goals in case they are changed. The overall workflow definition (also known as workflow schema in [4]) derived from the goals; again, this workflow definition also needs to version numbered, for use during adaptation
W-context	Here also, two types of workflow information regarding the individual Web service, need to be stored: Sub-goals for the individual component Web Services, derived from the overall goals; this can also be version controlled for identification during adaptation Sub-workflow schema, derived from the overall workflow schema stored at the C-context, again version controlled for identification during adaptation
I-context	As shown in Table 1, this context would capture and store data related to individual instances (i.e. which can be mapped onto an individual workflow task which belongs to a sub-workflow schema stored at the W-context) execution, and this data is then transmitted to the C-context as part of the sub-workflow execution data

5.2. Running example for exception handling

Before we describe our exception handling approach for Web services, we present the state transition diagram of a workflow task as suggested in [4] (Fig. 9). This diagram is complementary to a service chart diagram, i.e. during execution each service instance would transition from ACTIVATED state to either DONE or FAILED state.

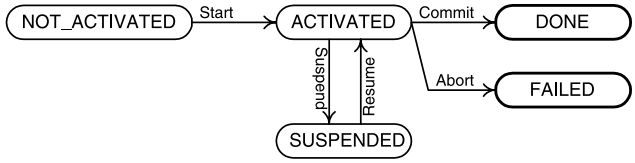


Fig. 9. State transition diagram of a Web service instance.

In case of exception handling of a service instance, several possibilities arise (let the instance in question be called WSI_1, its Web service provider be called WS_1, and composite service be called CS).

Possibility A: before execution, WSI_1 sends a message to CS about its inability of performing its task.

- The message reaches CS in time. CS performs one of the following operations without disrupting the execution of the rest of the Web service instances:
  (a) Try to renegotiate with WS_1 about a replacement possibility of WSI_1, perhaps with different QoS that still guarantee the user requirements.
  (b) If the outcome of the renegotiation in (a) is not satisfactory to CS, then negotiate with a new Web service provider $WS_{j(j<>1)}$ for a replacement.
- The message reaches CS before WSI_1 was supposed to begin execution, but this still has an impact on the rest of the execution of the Web service instances: same analysis as case (b).

Possibility B: during execution, WSI_1 sends a message to CS about its inability of completing its execution, which leads CS to suspend the execution of WSI_1, change its state to SUSPENDED, and perform aborts and roll-backs of other instances that are affected by the failure of WSI_1:

- Web service instances in ACTIVATED state can be either aborted or rolled back.
- Web service instances in DONE or FAILED state can only be rolled back.
- Web service instances in NOT_ACTIVATED state can only be aborted; this abortion will not have any effect on the other service instances, since the instance in question has not even started executing.

The difference between roll-backs and aborts resides in the type of actions that are taken after exception. In a roll-back scenario, we ensure that a service instance, which compensates the failure of a peer, is available. For example, a compensating instance of a credit-card payment service is to reverse the credit card charges to the customer. In an abort scenario, an instance in ACTIVATED state is aborted by simply stopping its

execution, without any compensation. Abort actions are to be implemented only when a service instance execution does *not* change the state of any other Web service component participating in the execution. For example, the credit card payment service execution can be aborted at any time before it actually deducts the payment from the customer's credit card account.

6. Related work

Our research is at the crossing point of several initiatives on Web services, composition, ontology, and context. While these concepts are independently studied (except for Web services composition and ontology), our research aims at their combination. In what follows, some of the initiatives that have backed our thoughts are discussed [7,21,23,25,27,30].

Strang and Linnhoff-Popien present interoperability of services at four levels: signature, protocol, semantics, and context [7]. The signature level focuses on the syntax of a service interface. The protocol level defines the relative order in which the methods of a service are called. The semantic level addresses the problem of a divergent understanding and interpretation of the information exchanged. Finally, the context level uses information that characterizes the state of an entity in order to identify its relevant aspects. These four levels illustrate the value-added of semantics and context to service interoperability. We consider interoperability and composition as the same because of their common objective: making distributed and heterogeneous Web services collaborate. In addition, we consider that our work is at a higher level of abstraction since semantics has driven context reconciliation instead of data reconciliation only.

One of the relevant uses of context is during Web services selection. Verheecke et al. argue that another limitation encountered in the field of Web services is that Web services can only be selected based on the functionality they offer [25]. WSDL-based Web services documentation does not support the explicit specification of the non-functional requirements such as constraint-based on QoS, access rights, and management statements. While we back the statements of Verheecke et al., we advocate that context is suitable for hosting non-functional requirements that are dynamic by nature. Context has also a dynamic nature as backed by Lonsdale and Beale in [30]. Both consider context not as a static but dynamic phenomenon. Lonsdale and Beale applied context to learning activities and noticed that context is constructed through the learner's interactions with the learning materials and the surrounding world over time.

While we strengthened the importance of reconciling context using ontologies, some authors such as Keidl and Kemper do not see any motive to that reconciliation [23]. For both authors, context encompasses all the information about the client of a Web service that may be utilized by the Web service to adjust execution and output delivery so that the client can benefit from a customized and personalized behavior [23]. In addition, they differentiate between context and the parameters of a Web service. We advocate that there is no need to exchange contextual information if the recipient Web service does not understand this information and hence, is not able to adapt its behavior according to the context of other Web services. A common understanding of the information exchanged is required, which backs our context reconciliation efforts.

7. Conclusion

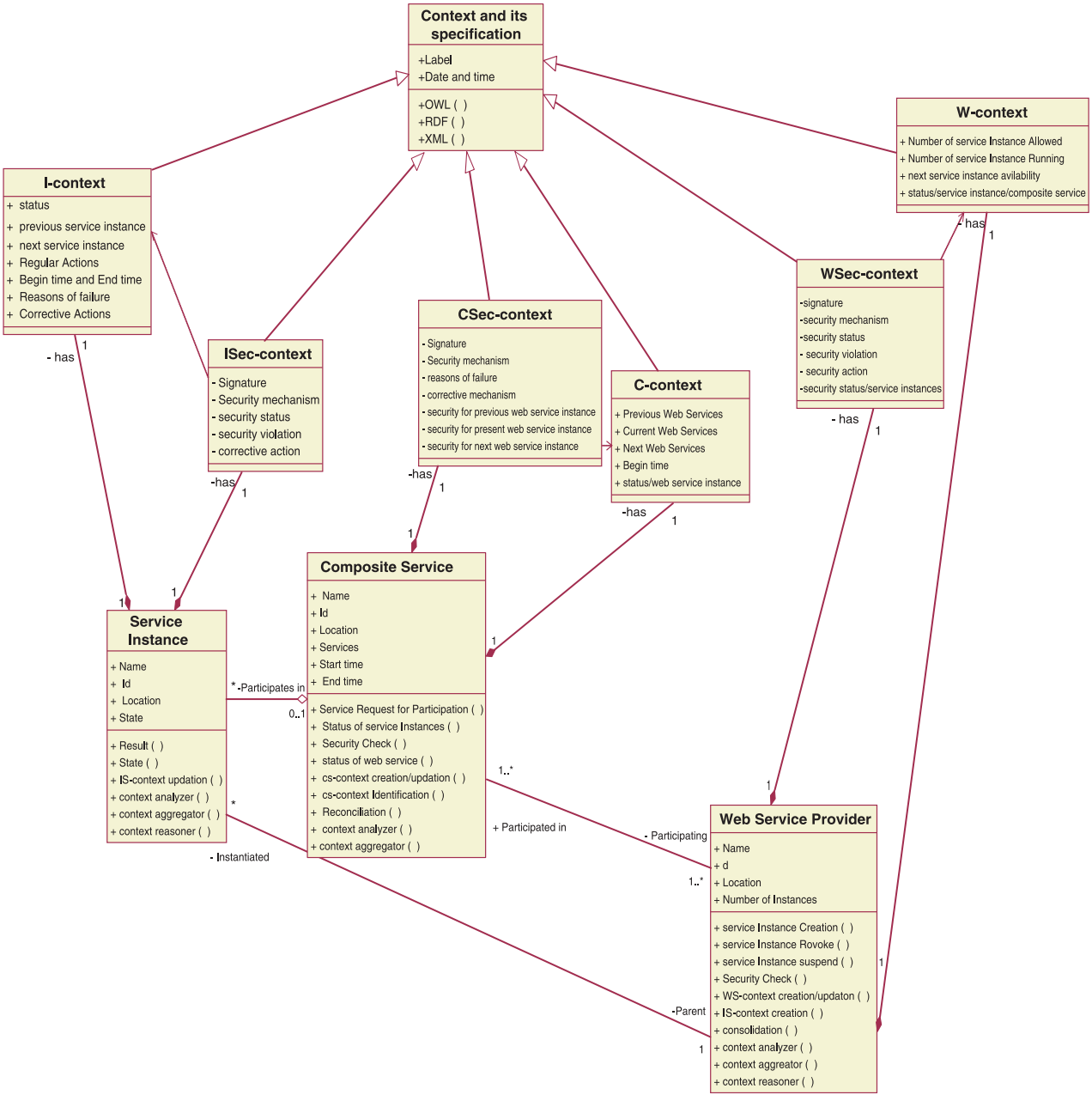
In this paper an ontology-based approach for the specification and security of contexts of Web services has been presented. Because multiple providers supply Web services for potential compositions, a reconciliation of their respective contexts was deemed appropriate. Besides the multiple origins of Web services, disparities between contexts at the granularity level also exist as the three types of contexts (I-, W-, and C-context) have shown. The importance of having a language such as OWL-C for context specification and management was stressed. Although Patil et al. claim that semantically described services will enable better service discovery, allow easier interoperation, and composition of services [19], we claim that semantically described context of services will enable better tracking and promote easier interoperability of Web services.

Despite the widespread use of Web services, we have shown that they still lack the capabilities that propel them to the acceptance level of traditional integration middleware. Web services are still unaware of the environment in which they operate. However, there are several situations that call for Web services self-assessment so that scalability, autonomy, and stability requirements are met. By scalability, we mean the capacity of a Web service to interact with a small or large community of Web services without having its expected performance either disrupted or reduced. By autonomy, we mean the capacity of a service to accept or reject demands of participation in composite services. Finally, by stability, we mean the capacity of a Web service to resist change while maintaining function and recover to normal levels of function after disturbances. To satisfy these requirements, Web services have to assess first, their current capabilities and ongoing commitments, and second, their surrounding environment prior they bind to any composition. Web services need to be context-aware.

Acknowledgements

The second author wishes to thank his manager, K. Muralidharan, for his support. The third author is supported by the Center for Advanced Studies (CAS) program of IBM Software Labs India. The third author would also like to thank Prof. K.C. Shet of NITK for supporting his doctoral work. Last but not least, the authors acknowledge the contributions of Prof. Willem Jan van den Heuvel from Tilburg University in The Netherlands, to the work presented in this paper. Other company (i.e. non-IBM), product and service names may be trademarks or service marks of others.

Appendix A. Class diagram for contexts of Web services and security



Appendix B. Excerpt of OWL-S document of W-context

```
<?xml version='1.0'?>
<!DOCTYPE rdf:RDF [
<!ENTITY rdf "file:/C:/ss/eclipse/workspace/myswede/test.owl">
<!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#">
<!ENTITY owl "http://www.w3.org/2002/07/owl#">
<!ENTITY base "file:/C:/ss/eclipse/workspace/myswede/test.owl">
<!ENTITY dces-ont "http://orlando.drc.com/SemanticWeb/OWL/Ontology/DC/ver/0.1/dces-ont#">
<!ENTITY  drc-ves-ont "http://orlando.drc.com/SemanticWeb/OWL/Ontology/VES/ver/0.1/drc-ves-
ont#">
]>
<rdf:RDF
  xmlns:rdf ="&rdf;"
  xmlns:rdfs ="&rdfs;"
  xmlns:owl ="&owl;"
  xml:base ="&base;"
>
<owl:Ontologyrdf:about=""xmlns:dc="&dces-ont;"xmlns:ves="&drc-ves-ont;">
<ves:versioning>
<ves:VersionData>
<dc:title>W-context</dc:title>
<ves:version>1.0</ves:version>
<dc:creator>sattanathan</dc:creator>
<ves:releaseDate>17102004</ves:releaseDate>
<ves:status>Experiment</ves:status>
</ ves:VersionData>
</ ves:versioning>
<rdfs:comment>
This demonstrates the structural specification of W-context in OWL-S format.
</ rdfs:comment>
<owl:importsrdf:resource="&dces-ont;"/>
<owl:importsrdf:resource="drc-ves-ont;"/>
        ...
</ owl:Ontology>
<OWL:Contextrdf:about="Sample_W-context">
<Context:W-Context>
<context:Label> 129323 </context:ID>
<context:Instance-Allowed> 4 </context:Instance-Allowed>
< context:Instance-Running> 1 </context:Instance-Running>
<context:Next-instance-avaiability> 0 </context:Next-Instance-Avilability>
<context:status> active </context:status>
<context:date> 27.30-17/10/2004 </context:date>
</Context:W-Context>
</OWL:Context>
</rdf:RDF>
```

References

[1] B. Medjahed, A. Bouguettaya, A. Elmagarmid, Composing Web services on the semantic web, The Very Large Data Base Journal, Special Issue on the Semantic Web, Springer Verlag 12 (4) (2003).

[2] B. Benatallah, Q.Z. Sheng, M. Dumas, The self-serv environment for Web services composition, IEEE Internet Comput 7 (1) (2003).

[3] D. Berardi, D. Celanese, G. De Giacomo, M. Lenzerini, M. Mecella, A foundational vision for E-Services, In Proceedings of the Workshop on Web Service, E-Business, the Semantic Web (WES') held in conjunction with the 15th Conference on Advanced Information Systems Engineering (CaiSE'2003) Klagenfurt/Velden, Austria, 2003.

[4] N.C. Narendra, Design Considerations for incorporating flexible workflow and multi-agent interactions in agent societies, J Assoc Inform Syst (1) (2003).

[5] P. Brezillon, Focusing on context in human-centered computing, IEEE Intell Syst 18 (3) (2003).

[6] J.J. Bryson, D. Martin, S.I. McIlraith, L.A. Stein, Agent-based composite services in DAML-S: the behavior-oriented design of an intelligent semantic web in: Ning Zhong, Jiming Liu, Yiyu Lao (Eds.), Web Intelligence, Springer Verlag, 2003.

[7] T. Strang, C. Linnhoff-Popien, Service interoperability on context level in ubiquitous computing environments, In Proceedings of SSGRRR, The International Conference on Advances in Infrastructure for Electronic Business, Education, Science, Medicine, Mobile Technologies on the Internet, L'Aquila, Italy, 2003.

[8] X.H. Wang, D.Q. Zhang, T. Gu, H.K. Pung. Ontology based context modeling and reasoning using OWL, In Proceedings of The Second IEEE Conference on Pervasive Computing and Communications Workshops (PerCom'2004), Orlando, Florida, US, 2004.

[9] J. Lilly. Tips and Tricks: Web Services Attacks and Defenses (White Paper). January 2004 (osdn.bitpipe.com/detail/RES/1080320572_938.html), visited June 2004.

[10] A.K. Dey, G.D. Abowd, D. Salber, A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications, Human-Comput Interact J, Special Issue on Context-Aware Computing 16 (1) (2001).

[11] M. Sabou, D. Richards, D. van Splunter, An experience report on using DAML-S, In Proceedings of the 12th International World Wide Web Conference (WWW'2003), Budapest, Hungary, 2003.

[12] T.R. Gruber, A translation approach to portable ontologies, Knowledge Acquisition, vol. 5, Academic Press, 1993.
[13] P. Lonsdale, H. Beale, Towards a dynamic process model of context, In Proceedings of The First International Workshop on Advanced Context Modelling, Reasoning, Management held in Conjunction with The Sixth International Conference on Ubiquitous Computing (UbiComp'2004), Nottingham, England, 2004.
[14] Z. Maamar, B. Benatallah, W. Mansoor, Service chart diagrams—description and application, In Proceedings of The Alternate Tracks of The 12th International World Wide Web Conference (WWW'2003), Budapest, Hungary, 2003.
[15] Z. Maamar, S. Kouadri-Mostéfaoui, H. Yahyaoui, Towards an agent-based and context-oriented approach for web services composition, IEEE Transact Knowledge Data Eng 17 (5) (2005).
[16] M. Papazoglou, D. Georgakopoulos, Introduction to the special issue on service-oriented computing, Commun ACM 46 (10) (2003).
[17] B. Medjahed, A. Rezgui, A. Bouguettaya, M. Ouzzani, Infrastructure for E-government Web services, IEEE Internet Comput 7 (1) (2003).
[18] L. Kagal, M. Paolucci, N. Srinivasan, G. Denker, T. Finin, K. Sycara, Authorization and privacy for semantic Web services, IEEE Intell Syst 19 (4) (2004).
[19] A. Patil, S. Oundhakar, A. Sheth, K. Verma, METEOR-S Web service annotation framework, In Proceedings of the 13th International World Wide Web Conference (WWW'2004), New York, USA, 2004.
[20] F. Curbera, R. Khalaf, N. Mukhi, S. Tai, S. Weerawarana, The next step in Web services, Commun ACM 46 (10) (2003).
[21] H.G. Hegering, A. Küpper, C. Linnhoff-Popien, H. Reiser, Management challenges of context-aware services in ubiquitous environments, In Proceedings of the 14th IFIP/IEEE Workshop on Distributed Systems: Operations and Management (DSOM'2003), Heidelberg, Germany, 2003.
[22] A. Dogac, G. Laleci, Y. Kabak, A context framework for ambient intelligence, In Proceedings of eChallenges Conference, Bologna, Italy, 2003.
[23] M. Keidl, A. Kemper, Towards context-aware adaptable web services, In Proceedings of the 12th International World Wide Web Conference (WWW'2003), Budapest, Hungary, 2003.
[24] P. Bouquet, F. Giunchiglia, F. van Harmelen, L. Serafini, H. Stuckenschmidt, C-OWL: contextualizing ontologies. In Proceedings of the 2nd International Semantic Web Conference (ISWC'2003), Florida, US, 2003.
[25] B. Verheecke, M.A. Cibran, V. Jonckers, AOP for dynamic configuration and management of web services, In Proceedings of The International European Conference on Web Services (ICWS'2003), Erfurt, Germany, 2003.
[26] Q.Z. Sheng, B. Benatallah, M. Dumas, E. Mak, SELF-SERV: a platform for rapid composition of web services in a peer-to-peer environment, In Proceedings of The 28th Very Large DataBase Conference (VLDB'2002), Hong Kong, China, 2002.
[27] R. Power, D. Lewis, D. O'Sullivan, O. Conlan, V. Wade, A context information service using ontology-based queries, In Proceedings of The First International Workshop on Advanced Context Modelling, Reasoning, Management Held in Conjunction with The 6th International Conference on Ubiquitous Computing (UbiComp'2004), Nottingham, England, 2004.