DYNAMIC INTEGRATION OF CONTEXT MODEL CONSTRAINTS IN WEB SERVICE PROCESSES

Kosala Yapa Bandara, MingXue Wang and Claus Pahl
Dublin City University, School of Computing
Glasnevin, Dublin
Ireland
[kyapa|mwang|cpahl]@computing.dcu.ie

ABSTRACT

Autonomic Web service composition has been a challenging topic for some years. The context in which composition takes places determines essential aspects. A context model can provide meaningful composition information for services process composition. An ontology-based approach for context information integration is the basis of a constraint approach to dynamically integrate context validation into service processes. The dynamic integration of context constraints into an orchestrated service process is a necessary direction to achieve autonomic service composition.

KEYWORDS: Autonomic Composition, Context Model, Context Constraints, Web Service Processes.

1. Introduction

Web services are self-contained, self-describing, and modular applications that can be published, located and invoked across the Web. An increasing amount of organizations only implement their core businesses and outsource other application services over the Internet. Thus the ability to efficiently and effectively select and integrate inter-organizational and heterogeneous services over the Web at run-time is an important step towards the development of these Web service applications. The requirement to address service composition in dynamic environments demands a high degree of flexibility and autonomy. A context notion can capture both business-level and technical requirements for autonomic composition. Dynamic integration of context requirements, i.e. constraints-based monitoring and validation, is an important step in the direction of autonomic service composition.

The notion of context has been defined by Brahim and Yacine as any information that can be used by a Web service to interact with clients and client to interact with Web services [1]. We define context as any static or dynamic client-, provider- or service-related information, which enables or enhances efficient communications among clients, providers and services. This allows us to capture context as used in autonomic service composition.

We can identify a number of shortcomings in current approaches to dynamic service composition:

- Context modelling is a promising approach for capturing context information requirements at design time, however it currently lacks a suitable capabilities for exchanging and integrating context models and instances in heterogeneous systems dynamically [5].
- Constraint languages, such as WSCol (Web Service Constraint Language) or the Java Modelling Language, are used in composition environments like the Dynamo service monitoring platform [2] to support fault handling. However, in terms of service composition, more integration is needed at the context model level to capture business and technical aspects beyond basic fault handling.
- Nowadays, WS-BPEL is the de facto standard for web service composition [6]. However, WS-BPEL remains focused on syntactical aspects with no consideration of semantic composition aspects. The Web service protocol stack does not address the requirements of a successful semantic exchange.

Dynamic integration of context information can achieve context-based semantic bindings among services at runtime. For instance, business rules can change often, which makes the dynamic and efficient adaptation of compositions to the new rules necessary. Their integration in a modular fashion as constraints into the processes is needed [3]. In this paper, we propose an approach to dynamically integrate context model-based constraints into Web service processes.

In this paper, Section 2 illustrates a motivating example. Section 3 illustrates our context model, which has been developed to focus on autonomic semantic Web service composition. In Section 4, we introduce our dynamic context integration architecture in detail. We discuss and evaluate a prototype implementation in Section 5. Finally, we discuss related work and present some conclusions.

2. A Motivating Composition Scenario

Our example focuses on a broker architecture where a client can requests one of his utility bills from a range of devices:

UserBillRequest (UserID, UserName, UserAddress, UtilityType, BillRequestDevice, BillRequestCurrency)

The service broker (e.g. a bank or post office) is responsible for providing the requested utility bill in the requested currency to the requested device. Here it is assumed that both user and service provider have already registered with the service broker.

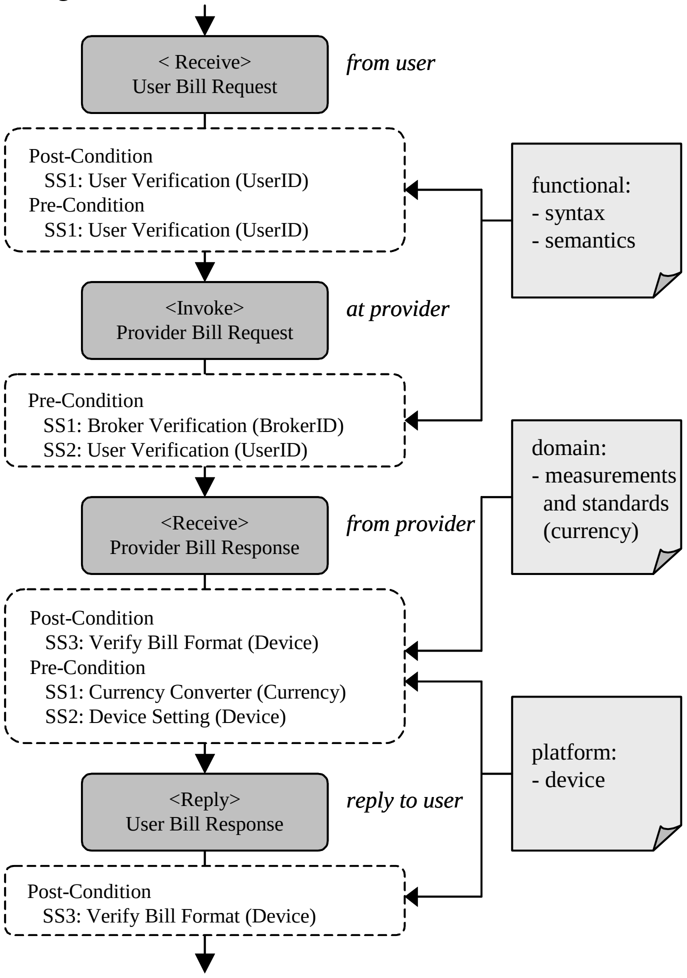
< Receive>
User Bill Request
from user
Post-Condition
SS1: User Verification (UserID)
Pre-Condition
SS1: User Verification (UserID)
functional:
- syntax
- semantics
<Invoke>
Provider Bill Request
at provider
Pre-Condition
SS1: Broker Verification (BrokerID)
SS2: User Verification (UserID)
domain:
- measurements and standards (currency)
<Receive>
Provider Bill Response
from provider
Post-Condition
SS3: Verify Bill Format (Device)
Pre-Condition
SS1: Currency Converter (Currency)
SS2: Device Setting (Device)
platform:
- device
<Reply>
User Bill Response
reply to user
Post-Condition
SS3: Verify Bill Format (Device)

Figure 1. Utility Bill Broker WS-BPEL Process

An initial user request results in a dynamic generation of a service process, composing the application Web services and weaving in appropriate supporting context-dependent constraint validation services. Constraints and their validation support are generated based on context information of the service involved. The user calls the UserBillRequest at the service broker that generates the WS-BPEL process. This integrates ProviderBillRequest, ProviderBillResponse and UserBillResponse application services. All related context constraints are integrated into the Web service process as pre-conditions or post-conditions, i.e. all context constraints are grouped under these two categories by the context constraint generator. Context constraint checking itself is provided as Web services to make this equally reusable and configurable. The constraints SS3 before and after the user bill response are the same, but the bill formats vary depending on the destination context. The user might expect his/her bill on his mobile device (user-friendly format in appropriate resolution) whereas the service broker expects it in machine-processable format (XML).

3. Context Model for Service Composition

Often, context models are developed for specific domains. Researchers like Dey et. al, Schilit et. al, Pascoe et. al and Brown et. al [10,11,12,13] have created context models suitable to their application needs. Brahim and Yacine have proposed a context categorization and context matching approach for Web services [1]. However, a lack of integrative context models that can be used in autonomic service composition led us to develop the following context model. Our model addresses both the user context and the service context, making it more complete and flexible for changing user environments (e.g. mobile applications) and service environments (e.g. dynamic heterogeneous systems).

3.1 Context Model

The user context captures context information about the service user, location information and the platform used (platform context). For example, the location context changes when the user moves from one place to another while working with a system using mobile devices. Service context is mainly divided into four categories.

Functional Context: This describes the operational features of Web services. The functional context is grouped into Syntax, Effect and Protocol context.

- **Syntax** includes the list of input/output parameters that define the operations' messages and the data types of the parameters for invoking the Web service.
- **Effect** includes pre-conditions and post-conditions, i.e. the operational effects of an operation execution.
- **Protocol** refers to a consistent exchange of messages between services involved in an autonomic service composition to achieving their goals. The protocol context includes context on conversation rules and data flow.

Quality of Service Context (QoS): Facilitating end-to-end quality in service compositions is a significant challenge because of two difficulties. One difficulty is to define quality and to determine quality. Then, based on an agreed concept of quality, QoS for individual services needs to be modelled. In addition to single service QoS, end-to-end QoS is critical for business processes, which are composed of both single and compound services. There is no adequate model or method to provide good support for QoS in Web service compositions [8]. In our model, quality of service context is grouped into four major categories [9].

- **Runtime:** enables the measurement of properties that are related to the execution of a service. Performance context: the measurement of the time behaviour of services in terms of response time, throughput etc. Reliability context: the ability of the service to be executed within the maximum expected time frame. Availability context: the probability that the service is accessible.
- **Financial / Business:** allows the assessment of a service from a financial or business perspective. Cost context: the amount requited for execution. Reputation context: measures the service's trustworthiness. Regulatory context: a measure of how well a service is aligned with government or organizational regulations.
- **Security:** describes whether a service is compliant with security requirements. Integrity context: protecting information from being deleted or altered without the permission of the owner. Authentication context: ensures that both consumers and providers are identified and verified. Non-repudiation context: the ability of the receiver of something to prove to a third party that the sender really did send the message. Confidentiality context: protecting information from being read or copied by anyone who has not been explicitly authorized.
- **Trust:** refers to establishment of trust relationships between client and providers – which is a combination of technical assertions (measurable and verifiable quality) and relationship-based factors (reputation, history of cooperation).

Domain Context: Each application domain may need its own requirements met for interacting with services.

- **Semantic:** refers to semantic framework (i.e. concepts and their properties) in terms of vocabularies, taxonomies or ontologies.
- **Linguistic:** the language used to express queries, functionality and responses.
- **Measures and Standard:** refers to locally used standards for measurements, currencies, etc.

Platform Context: The technical environment a service is executed in.

- **Device:** refers to the computer/hardware platform on which the service is provided.
- **Connectivity:** refers to the network infrastructure used by the service to communicate.

3.2 Context Model Ontology

Context model information comes from very different sources. The functional context is derived from the service descriptions; quality and platform contexts are captured based on system and platform data. Domain context is based on meta-level information. This diversity requires an integrating framework, which we provide in the form of a context model ontology.

The context model ontology is an OWL-light ontology that, at its core, captures the context model categories in the format of a taxonomy (concept level of the ontology). OWL provides the necessary interoperability between possibly different source formats. We assume these to be mapped onto the OWL context ontology. We only illustrate a few excerpts here. We use the Manchester OWL syntax here to avoid the verbosity of XML-OWL.

```
Class:          FunctionalContext
SubClassOf:     Context
DisjointWith:   QoS or Domain or Platform

Class:          Syntax
SubClassOf:     FunctionalContext
DisjointWith:   Effect or Protocol
```

Specific linkages – for instance between Trust and Security context – can be formalised by using EquivalentTo instead of DisjointWith at lower taxonomy levels. Specific properties can be formulated, for instance for Syntax:

```
hasInterface MIN 1 and hasInterface SOME string
```

which requires a syntax element to have at least one interface associated to it that must be of type string.

The context ontology also has a second purpose. We capture concrete context model instances as instances of the ontology. We illustrate this using the UserBillRequest service in terms of our context model ontology.

```
UserBillRequest (UserID, UserName, UserAddress,
      UtilityType, BillRequestDevice, BillRequestCurrency)
```

Each parameter has a data type and the Web service has functionality, both specified as context information. In this example, the user information is the user context (UserID, UserName, UserAddress). This is provided to the service in the form of parameters, i.e. the user context becomes syntax for this service. For an Interface element, we can express:

```
hasUserID VALUE 123              and
hasUserName VALUE 'John'         and
hasUserAddress VALUE 'Dublin'
```

UserAddress and UtilityType are the other syntax context elements of this service. BillRequestCurrency is a domain context element (measurements and standards). Parameter BillRequestDevice is device context element, part of the platform context. Concrete values can be attached as above.

4. Dynamic Context Model Integration

4.1 Constraint Integration Architecture

The context model details the properties of users and service providers in the defined categories. The context ontology is used in the respective specifications. The ontology-based context instances, which define and describe a concrete situation are then converted into context constraints and context services that validate them. At composition time (here dynamically at runtime), context constraints (validated through context services) are integrated with the application Web service process.

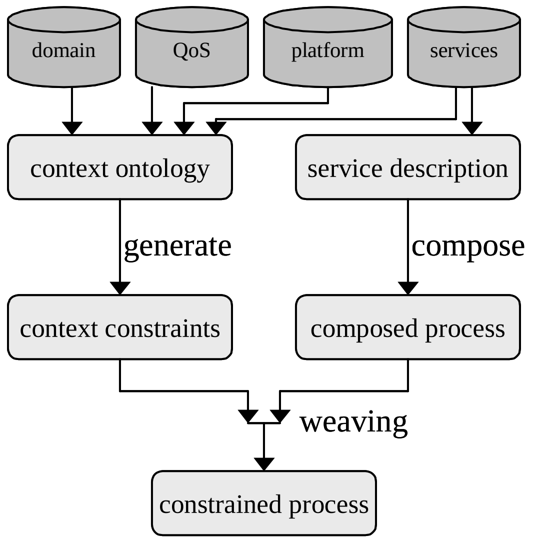

Figure 2. Context Model Integration Architecture

4.2 Constraint Template

When a service, such as the bill request, is called by the service broker, the user is verified first, which is governed by the information provided through the authentication context of the service (QoS context). In order to achieve a uniform approach to context constraint validation, all constraints such as the user verification become post-conditions of the service within the integrated, composed Web service process. A standard constraint checker can be used here. However, for some constraint types, additional information needs to be provided through so-called data collectors.

The supporting constraint services are annotated in the form of a context template. A context template has link, condition and expression elements.

- The **link** is used in supporting service binding. Path expressions (XPath) are used in specifying the location in the service composition.
- The **condition** explains both the type of the supporting service (pre-condition or post-condition) and the order of execution.
- The **expression** specifies the constraint itself to be checked by the supporting context service – formulated using syntax and semantics defined in the context constraint language.

An index file of supporting services maintains pointers to the supporting constraint services of an application service – remember that these were customised by the service-specific context.

4.3 Constraint Language

The Java Modelling language (JML) provides a suitable foundation as our context constraint language. JML is a behavioural interface specification language.

- The keyword *requires* is used in specifying pre-conditions. A precondition is a condition that must be satisfied before calling a service.
- The *ensures* keyword indicates that what follows is a post-condition that must be satisfied.

For instance, the UserBillRequest Service proceeds further only if the user is verified. This is an authentication (the QoS context category). For the userVerification (with parameter UserID), we get:

```
<Link: path expression to the service process/>
<Condition>
          <Type> post-condition </Type>
          <Order> 1 </Order>
</Condition>
<Expression>
  @ensures \returnBoolean(
      Context:userVerification(UserID) ) == True;
</Expression>
```

The @ensures expression requires the return value of the userVerification context service (located at <Context> with parameter is UserID) to be true. Similarly, for the UserBillResponse service, the constraint depends on the user device and device type (platform context):

```
<Link: path expression to the service process/>
<Condition>
          <Type> post-condition </Type>
          <Order> 3 </Order>
</Condition>
<Expression>
  @ensures \returnBoolean(
      Context:compareBillFormat(),
      DeviceType,
      Context:setBillFormat(
          Context:getBillFormat(DeviceType) ) ) == True;
</Expression>
```

This post-condition ensures that compareBillFormat returns true. SetBillFormat calls getBillFormat, located at Context, with the parameter DeviceType to set the bill format. The parameters for the ensure statement are the Context-based CompareBillFormat() service, the DeviceType parameter and the currently set bill format (which is set by calling getBillFormat in setBillFormat) , Then compareBillFormat compares the device type with the set bill format and returns true if the bill format matches the device type. If not, it returns false.

In the UserBillResponse service, the VerifyBillFormat supporting service queries the existing device (e.g. mobile) and sets the appropriate settings for the bill to be displayed on that device. By default, devices uses an XML format so that the bill can be used in other systems for further processing. In the ProviderBillResponse service, the device setting is the default one and the predefined XML format is used in forwarding the bill to the broker.

4.4 Transformation and Weaving

At the centre of the integration of constraint validation into a composed service process is a mapping:

- Attributes of the context model aspects are connected to concrete values at the instance level. These form the context constraints. Thus, attributes like UserID or BillRequestDevice are extracted from the ontology. Pairs of attributes with their associated values form abstract constraints.
- A preparation step for the final mapping is the determination of data collectors (e.g. getBillFormat) and data initialisers (e.g. SetBillFormat) that support the constraint condition (e.g. compareBillFormat).
- The abstract constraints are mapped to JML pre- or postcondition constraints as illustrated in Section 4.3. Hereby, the use of data collectors and data initialisers needs to be considered.

Context service calls for constraint checking are generated based on context ontology instances. These service invocations are based on information given in the constraint templates, i.e. path expression (link), context constraints (condition), context services and context constraint language (expression).

Context constraint validation is implemented using the context services that encapsulate the constraint checker. Context services are services that directly deal with context instances. A service-related context specification, the context ontology, and context service invocation shells can be precomputed at the development stage of the main services. However, depending on the changing environment, the weaving process is executed in parallel with the Web service process planner/generator. For example, the bill format is set for the respective service calls, e.g. ProviderBillResponse or UserBillResponse. The path expression (to the context service) is set during context service planning, thus enabling dynamic bindings.

4.5 Service Process Execution

Based on the service process outline from Fig. 1, application service invocations (the grey boxes) are generated in the format of the corresponding WS-BPEL command:

```
<bpel:invoke name="UserBillRequestInvoke"
        partnerLink="UserBillRequestPL"
        operation="UserBillRequestProcess"
        portType="ns:UserBillRequestDelegate"
        inputVariable="UserBillRequestRequest"
        outputVariable="UserBillRequestResponse">
</bpel:invoke>
```

Constraint validation is woven in by calling the context services that encapsulate the constraint checker.

```
<bpel:invoke name="UserVerificationInvoke"
        partnerLink="UserVerificationPL"
        operation="UserVerificationProcess"
        portType="ns:UserVerificationDelegate"
        inputVariable="UserVerificationRequest"
        outputVariable="UserVerificationResponse">
</bpel:invoke>

<bpel:if name="userVerificationConstraint">
  <bpel:condition>
    <![CDATA[
      $UserVerificationResponse.parameters/return ='true']]>
  </bpel:condition>
```

The execution of the application process will only proceed if the respective constraint is not violated.

5. Prototype Implementation

We have implemented a service process planning and monitoring architecture for which we have developed a prototype. In a context-determined composition approach, a composition planner works in parallel with the constraint validation generator. The WS-BPEL engine running the process needs to integrate the constraint validation checker. We are working with ActiveBPEL engine and the OCL checker here.

Performance is central. Two aspects emerge.

- Firstly, the ontology processing and weaving is time-consuming – depending on the size of the ontology. A strategy that favours early pre-computation of constraint templates (from the ontology as soon as changes to the ontology are known if the application services are determined) is advisable.
- The second aspect is the constrained service execution. Our first experiments with different variants of the architecture in terms of the constraint weaving into the BPEL service process – in particular with respect to how constraint violations are handled – shows an acceptable overhead of around 15-18% for constraint validation through the context services.

While the constraint checking performance is more or less constant for each of the categories – of course there are variations between the different types of data collection involved. However, the solutions vary in terms of the fault handling applied. WS-BPEL fault handling provides support for category-specific, but also engine-independent handling, which turns out to be the most flexible form.

6. Related Work

The related work in this area covers Web service composition approaches and rule-based approaches including rule representation formats and rule integration approaches [3]. Baresi et.al. present work on monitoring directives, called monitoring rules, and a weaving approach for the dynamic inclusion into Web service processes. Dynamic selection and execution of monitoring rules at runtime and also data acquisition and analysis into monitoring rules are core elements of the solution [2]. Monitoring rules are selected by the designer at design-time. However, this approach can be improved in terms of monitoring rule determination. In our approach, we use the context model for modelling and integrating context information and then use context instances in determining context constraints and context services attached to an application service. Our context constraint validators are invoked through the supporting services, making them compatible and reusable in a range of service-based architectures.

The METEOR-S project has focused on constraint-driven Web service composition [4]. It distinguishes data, functional and quality of service semantics, but the approach to capturing and specifying of semantics can be extended. Especially, the METEOR-S approach does not sufficiently support the dynamic binding of constraints – the use SWRL and OWL to provide more descriptive rules for specifying constraints is, however, planned. Our approach is more advanced in capturing, specifying and binding of semantics using the context model ontology with context instances and corresponding sound supporting services and context validators.

7. Conclusions

Our approach focuses on dynamic service composition problems, driven by context constraints validation problems. Context-based composition is an essential ingredient for autonomic Web service composition. We first defined a notion of context in a range of categories, then formalised an approach of specifying semantic context information in a context ontology and finally introduced a technique to integrate context constraints dynamically into Web service processes.

Two central contributions define our approach:

- A context model that captures both business and technology aspects and that, through its ontology-based formalisation, acts as an integrator for context information stemming from different sources.
- A uniform approach to runtime context constraint validation, based on a dynamic mapping on ontology instances to constraints that can be validated during the execution of a dynamically composed service process.

These are two central stepping-stones towards an autonomic service composition platform.

We are planning to enhance the technique in two directions. Firstly, we aim to automate the integration of different information sources into the context ontology. Secondly, in particular the validation of non-functional constraint properties requires a deeper investigation into the provision, selection and use of appropriate data collectors for different qualities.

Acknowledgement

The authors are grateful for the support the Science Foundation Ireland has provides for the CASCAR project.

References

[1] Brahim Medjahed and Yacine Atif, Context-based matching for web service composition, *Journal: Distributed and Parallel Databases*, Volume 21, pages 5-37, Springer Netherlands, 2007.
[2] L. Baresi and S. Guinea, Towards Dynamic Monitoring of WS-BPEL Processes, *In Proceedings of the 3rd International Conference on Service Oriented Computing*, 2005.
[3] F. Rosenberg and S. Dustdar, Business Rules Integration in BPEL- A Service Oriented Approach, *In Proceedings of the 7th IEEE International Conference on E-Commerce Technology* (CEC'05), 2005.
[4] R. Aggarwal, K. Verma, J. Miller and W. Milnor, Constraint Driven Web Service Composition in METEOR-S, *In Proceedings of IEEE International Conference on Services Computing*, 2004.
[5] R. Robinson and K. Henricksen, XCML: A runtime representation for the Context Modelling Language, *In Proceedings of the 5th Annual IEEE International conference on Pervasive Computing and Communications Workshops* (PerComW'07), 2007.
[6] M. Mrissa, C. Ghedira, D. Benslimane, Z. Maamar, F. Rosenberg and S. Dustdar, A Context-Based Mediation Approach to Compose Semantic Web Services, *ACM Transactions on Internet Technology,* Vol.8, No. 1, Article 4, November 2007.
[7] C. Moore, M. Wang, C. Pahl, An Architecture for Autonomic Web Service Process Planning, *The 3rd Workshop on Emerging Web Services Technology* (WEWST-2008), November, 2008.
[8] Z. Wu, J. Luo, QoS-Resource Graph Model for Web Service Composition in Service Oriented Computing, *The Sixth International Conference on Grid and Cooperative Computing* (GCC 2007), IEEE Computer Society, 2007.
[9] B. Medjahed, A. Bouguettaya, A Dynamic Foundational Architecture for Semantic Web Services., *Journal: Distributed and Parallel Databases*, Volume17, Number 2, pages 179-206, Springer Netherlands, 2005.
[10] A.K. Dey and G.D. Abowd, Towards a Better Understanding of Context and Context-Awareness, Lecture Notes In Computer Science; Vol.1707, *Proceedings of the 1st international symposium on Handheld and Ubiquitous Computing*, Germany, pp.304-307, 1999.

[11] P.J. Brown., J.D. Bovey and X. Chen. Context-Aware Applications: from the laboratory to Marketplace. *IEEE Personal Communications* 4, 5, 58-64. 1997.
[12] J. Pascoe, Adding General Contextual Capabilities to Wearable Computers. *In 2nd International Symposium on Wearable Computers (ISWC 1998)*, pp. 92-99. 1998.
[13] W.N. Schilit, A System architecture for Context-Aware Mobile Computing. *PhD thesis*, Columbia University, 1995.