Context Adaptation of Web Service Orchestrations

Frederick Seyler Chantal Taconet
Guy Bernard
GET / INT, CNRS Samovar 9 rue Charles Fourier, 91011 Evry, France
Tel: +331-60-764515
Fax.:+331-60-764780
{Frederick.Seyler,Chantal.Taconet,Guy.Bernard}@int-evry.fr

Abstract

With orchestrations, one service may be realized through the cooperation of several services. This cooperation has to be formally described. In this paper, we propose to describe service orchestrations according to UML2 meta-model through three UML2 diagrams. Component diagrams describe each service external interfaces. Collaboration diagrams describe the structural composition of services. And activity diagrams describe the orchestration of services.

The main contribution of this article is to mix the orchestration and composition meta-model with a context meta-model. Thus, we propose to include the descriptions of context awareness into the orchestration and composition meta-model. This approach allows application designers to describe flexible orchestration of services. Furthermore, describing compositions and their context-awareness with a model (conform to a meta-model) allows middleware with model-transformation capabilities to produce ad-hoc compositions in term of adaptation to current context execution and in term of target execution platforms.

We present in this article two kinds of adaptation of context aware orchestrations: deployment time and run time adaptations.

1. Introduction

The meta-model presented in this article is part of a general framework, called CAComp (Context Aware Compositions). This MDE (Model Driven Engineering) framework aims at exploiting context information to perform context adaptation on composition and orchestration models. As several MDE frameworks, such as Ugatze [2, 14], it exploits separation of concerns and non functional properties to perform model transformations. CAComp applies this approach to context properties to obtain compositions adapted to context situations.

Following MDA (Model Driven Architecture) initiative [9, 7], CAComp meta-model is made up by two main abstraction levels, platform independent level for building Platform Independent Model of context-aware composition, and platform specific level. This architecture allows CAComp to produce orchestrations for several platforms (e.g. BPEL [10], XLANG [16] or WSFL [8]).

In CAComp, the composition meta-model is a subset of UML2 [13]. Additionally the context-aware composition meta-model contains context and context awareness definitions. Context adaptations are taken in consideration during the model transformation process. Therefore, CAComp supports a model driven process for context aware web service based applications development.

This article is structured as follows. In Section 2, we present an illustrating context aware orchestration example: The rugby match management process. Then, in Section 3, we firstly define main concepts of Context meta-model, then we present the composition context aware entities we have identified and finally we describe the kinds of adaptations which are likely to be applied to context aware orchestrations of Web Services. The meta-model allows the CAComp framework to host several transformations. We present two of them : transformations for context awareness during the deployment process of the orchestration in Section 4 and transformations for context awareness during the runtime of the orchestrations in Section 5. Finally, Section 6 concludes the article, compares our proposition to related works and draws perspectives concerning context adaptation of service compositions.

2. Context Aware Composition Example

In CAComp, a context aware composition is a composition which is able to present different configurations according to context situations. The variations may be for example

in term of number of services, in term of service interfaces and in term of sequence of activities performed in the orchestration. In this section, we introduce a concrete example, Sport Match Report Form Process which will illustrate the variations of a Context Aware Composition.

Nowadays, the need to develop information systems to report form management is becoming more apparent at amateur level of collective sports (rugby, football or basket ball). Currently, referee, delegate and club representatives all fill a hard copy of the Match Roster. Each club representative copies on a paper form each player information (information found on his hard license). The referee checks each license before the match, and fills the match report form after the match. The League Delegate includes match movements in the report form, and the referee mails it to the local league. Consequently, official match result and tables are published one week after the match.

We can imagine that each participant in the report process (referee, league delegate, club representative) is equipped with different and heterogeneous computers (from workstation to mobile terminals). Thus, it becomes possible to automate the production of match report forms, and so result of matches may be published instantaneously in the Internet through an orchestration of services furnished by the regional league, the clubs, the referee and the delegate.

Additionally, the context situations on the play grounds where the matches take place are variables from one match to another: Namely the network bandwidth, final user terminals, topography, sensors availability. These variations may lead as we will see to various orchestrations.

Presentation of the orchestration through a component diagram presents advantages. (i) The orchestration can be seen as a UML2.0 Component with provided interfaces as its external representation (MatchManagementPort is the entry point of the orchestration). (ii) Additionally the required interfaces which represent its service dependencies are declared (*e.g.* RefereePort, RegionalLeaguePort, HomeClubPort invoke or send signals to external Components). (iii) The component owns an internal behavior (it will be described through an activity diagram) which allows the designer to describe the orchestration of services; (iv) Furthermore, a UML2.0 collaboration diagram specifies with components and connectors the structure of the collaboration.

In UML 2.0, an activity diagram may model a component behavior with token flow semantics (Petri Nets like). An *Activity* is an oriented graph with *ActivityEdge* and *ActivityNode*. An *ActivityEdge* governs data or controls flows between two *ActivityNode*. Figure 1 represents one of the behaviors of the MatchManagementProcess component with an activity diagram.

The activity receives Date, RefereeName as input parameters (*ActivityParameterNode*). The activity is made up of

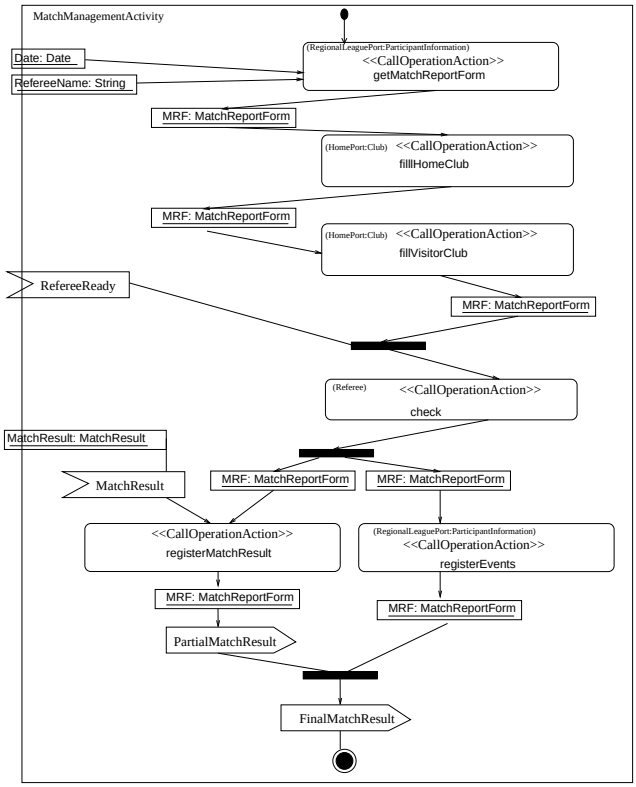

Figure 1. ContextAware Entities in CaComp

several nodes and edges which describe the flow of execution. Firstly, the getMatchReportForm node gets a MatchReportForm from the external RegionalLeague[1]. Then, the orchestration waits that the HomeClub and the VisitorClub both fill the match report form. Afterwards, an *AcceptEventAction* indicates the referee presence for supervising the match. Then, the referee checks the team licenses, and the referee and the delegate annotate the match in parallel. Finally, the match resulting MatchReportForm may be sent to the Regional League via a *SendSignalAction*.

In this section, we have illustrated with a concrete example the use of UML 2.0 concepts for describing Web Service Orchestrations structure and behavior in a component-based platform independent way. In following section, we supplement this description with context awareness (Section 3), which shows context-based variability of structural and behavioral aspects of the model.

[1]Invocation of the getMatchReportForm operation defined in the ParticipantInformation interface of the RegionalLeaguePort.

3. Context and Context Aware Entities

3.1. Context Definitions

In CAComp, we abstract the main elements of the Dey's context definition into meta-classes: "any information that can be used to characterize the situation of an entity". For example, "any information" observed becomes an *ObservableContext*. An *ObservableContext* owns a data-type, which determines the observed value data-type.

Figure 2 illustrates a part of CAComp meta-model focused on Context-Awareness. A *ContextAwareEntity* is a software system entity which may react to changes of its outside context. The set of *RelevantContext* identifies the set of *ObservableContext* that a given *ContextAwareEntity* is aware of. For example, the MatchManagementProcess component is a context aware entity, aware of observable contexts including: ScreenSizeContext, NetworkBandwidthContext, DateOfTheDayContext, LocationDistrictContext and Referee.UserNameContext.

A *ContextSituation* is defined by a composition between ranges of values of several observables contexts. A relevant Situation of a context aware entity must have been composed from a subset of its relevant contexts. At a given time, a Context Aware Entity may be in one or more RelevantSituations, a set of *ContextSituations*, associated to a ContextAwareEntity.

Several UML 2 model elements such as Components, Collaboration, Activity, may be context-aware. In CaComp, the concept of *ContextAwareEntity* establishes the link between Context meta-model and Composition meta-model (ie : Components, Collaboration, Activity, as a subset of UML2). We consider two kinds of context aware entities : *CompositeContextAwareEntity* and LeafContextAwareEntity. *CACollaboration*, *CAComponent* and *CAActivity* are *CompositeContextAwareEntity*, *CAConnector*, *CAActiviyEdge* and *CAActivityNode* are *LeafContexAwareEntity*.

Additionally, the context awareness of each CompositeContextAwareEntity is defined by *Mappings* and *Choices*.

A *Mapping* allows the designer to define a direct mapping between one of the elements of the *CompositeContextAwareEntity* (*targetCAE*) and one of its relevant contexts (*relevantContext*). With Mappings, a Context Aware entity is directly influenced by its relevant contexts.

A *Choice* allows CAComp Designer to choose between several *CaOptionalElements* according to a given relevant situation (*relevantSituation*). Each *Choice* associates a relevant situation and a set of optional elements. We illustrate these kinds of context awareness in the example scenario.

3.2. Example of Context-Aware Entity

Behavioral ContextAware Entities are component behavior (activity, operation, process). We give below examples of behavioral context Aware Entities. The MatchManagementActivity model illustrates mapping of operation parameters in an activity diagram.

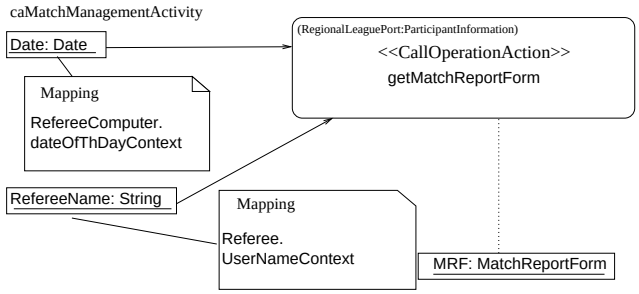

Figure 3. Parameter mapping in a part of CaMatchManagementActivity Model

In Figure 3, caMatchManagementActivity is a context-aware behavior of the MatchManagementProcessComponent. According to Figure 2, caMatchManagementActivity is a composite context aware entity. It is aware of a set of relevant contexts, namely Referee.UserNameContext and RefereeComputer.DateOftheDayContext. Two *ActivityParameterNode* (Date and RefereeName) are considered as mandatory *ContextAwareEntity*. The first *Mapping* associates DateOfTheDay Context to Date activity parameter node. The second *Mapping* associates Referee.UserNameContext Context to RefereeName activity parameter node. Thanks to those mappings, Date and RefereeName are directly given by context values.

caMatchManagementActivity is also aware of the following relevant situations: DelegateAvailable or DelegateUnAvailable. A second part of this context-aware Activity model is illustrated in Figure 6. In this activity, all the context aware entities included in the dashed rectangle are optional elements, associated to a default choice, and to DelegateUnAvailable relevant situation. When caMatchManagement is in this situation, the match event registration is done by Delegate component. The second choice associates DelegateUnAvailable to three activity edges and a *callOperationAction*. In this situation, the registration of Match events in the Match Report Form is done by the Referee. This Context Aware Activity contains a variability of different behaviors parts according to several context situation. Therefore several possible behaviors are represented in a single model.

Context adaptations may be processed at different time in the service lifetime: service development, service packaging, service deployment on the computers, service instan-

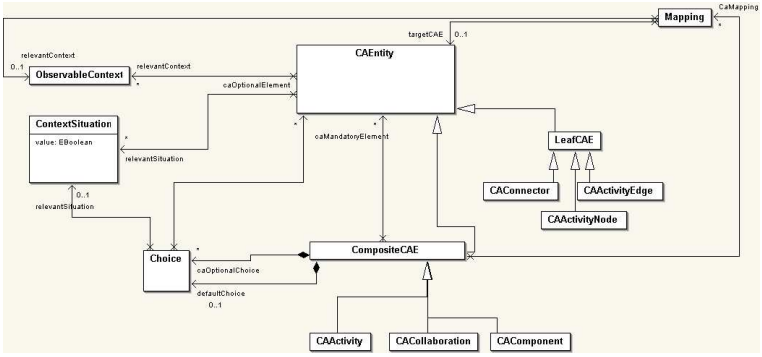

Figure 2. ContextAware Entities in CaComp

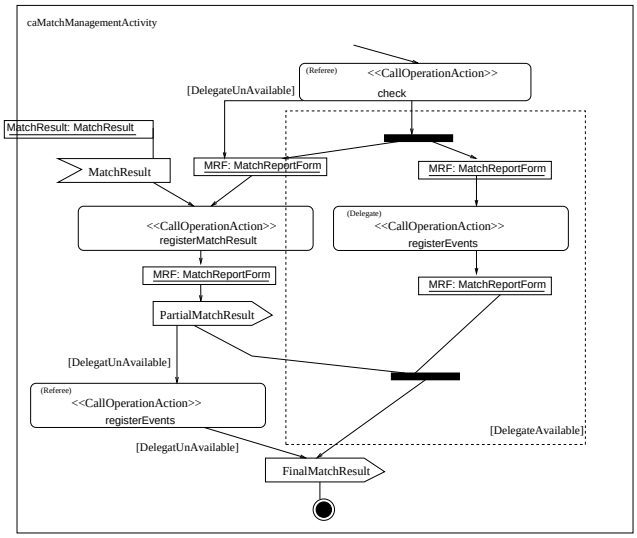

Figure 4. Context Aware Match Management Activity

tiation, service execution. CAComp provides an enrichment of the UML2.0 model for modeling Web Service Orchestration in an abstract way and for adding Context awareness to each viewpoint of the abstract model.

In the following sections, we present two kinds of context adaptations which could be applied on the obtained context aware orchestration model: Deployment-time adaptation which first collect context values to generate a context independent orchestration model (section 4), and runtime adaptation which generates a context-aware orchestration collecting context values at run-time and performing its choices.

4. Deployment-time adaptation of Orchestrations

Context Adaptation meta-model is essential for the CAComp context-aware code generation tool. It enables the composition designer to describe all the strategies and policies used by the generation tool to perform context adaptations according to current relevant situations at deployment time.

This adaptation follows these steps : first the architect builds an orchestration model, next he defines relevant contexts and relevant situations. The next step consists in associating orchestration model and context definitions, with building a context aware orchestration. Then the context aware orchestration model can be adapted according to the relevant context. The adaptation itself is made up of following sub steps : relevant contexts of the orchestration are collected, interpreted, and its relevant situation are detected. The orchestration is then adapted according to associated context values and relevant situations, and deployed to be directly executed.

In Figure 5, a deployment-time adaptation is performed on the caMatchManagementActivity context-aware activity model described in Figure 4 : first context values are collected and stored, thus the Context-aware Activity Model is transformed following context values with mappings and relevant situations. After this adaptation, caMatchManagementActivity can be easily deployed, as a classical web service Orchestration.

With Deployment-time adaptation, context-based mappings and choices are performed during the deployment stage. One one hand with this kind of deployment, the resulting Orchestration is static and do not need any additional context information (and consequently no interaction with any context collector) at runtime. One the other hand context changes cannot influence the execution of this orchestration, that is why we also propose Runtime-time Adaptations in Section 5.

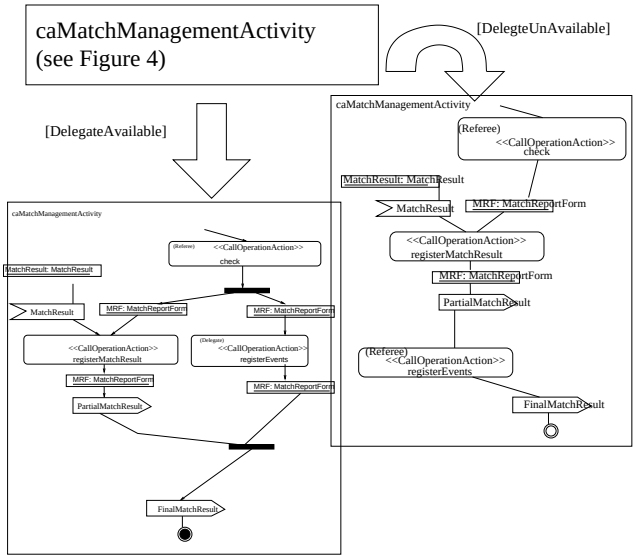

Figure 5. Deployment-time Choice Adaptation in MatchManagementActivity Model

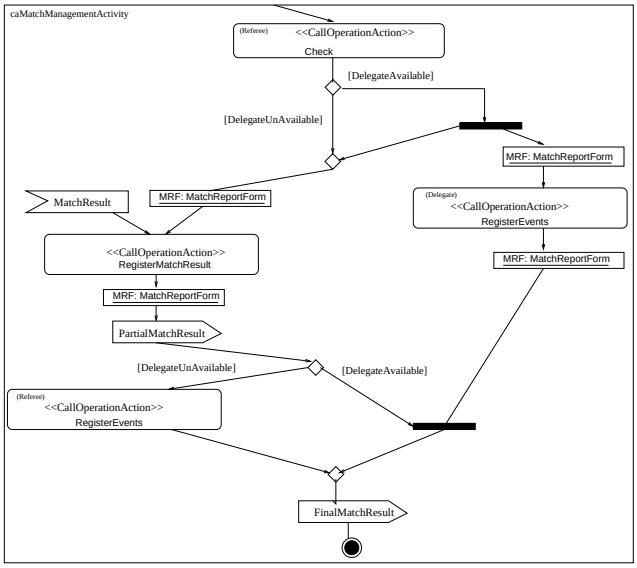

Figure 6. Run-time Choice Adaptation in MatchManagementActivity Model

5. Context Aware Runtime-time Adaptation of Orchestrations

Run-time adaptation consists in generating context aware artifacts able to adapt the orchestration to context values and relevant situations. The firsts step of this adaptation : Context Definition and Context Aware Orchestration building remain the same as in section4. The adaptation process performs the generation and the deployment of a Context Aware Orchestration, with context specific artifacts (new decision nodes, new sub activities, new interactions with a context collector).

For example, the run-time adaptation transforms caMatchManagementActivity with adding new constructs (new decision node, fusion nodes, activity edges ...). Each newly generated *DecisionNode* is associated to a *DecisionInput* called EvaluateRelevantSituation. This sub Activity calls a context collector able to evaluate the associated Relevant Situation. For example, in Figure 6, DelegateAvailable and DelegateUnAvailable are evaluated with this Decision Input.

6. Conclusion

The CAComp adaptation meta-model enables composition designers to define variations in the composition and in the process modeling. During deployment and execution, these information will be used by CAComp to choose the structure and the behavior of the composition and orchestration.

This paper has introduced a meta-model for context-aware composition of software services. This meta-model is made up by three meta-models. Software Service composition meta-model describes software services as UML 2.0 components, their assembly with collaboration and their behavior as UML2.0 activities. The context meta-model is an implementation of CAMidO meta-model citeBehlouli2006, and describes context information, collection and interpretation. The third meta-model is defined as a merge of the two previous meta-models, to describe context-aware Orchestrations of Web Services and their adaptation to relevant context situations. A real example of context-aware orchestration made up by several context aware services cooperating in different contexts illustrates the use of CAComp meta-model.

The proposition allows composition designers to use standard UML diagrams to describe orchestration of services. Furthermore, they can describe the context awareness of their compositions.

Implemented with MDE technologies, the CAComp framework is used at deployment time to produce composition adapted both to the target platforms and to the current context situations. At deployment time it produces platform-specific models and descriptors after having collected and interpreted relevant context, according to mappings and choices. At run time it produces a context aware platform-specific models and descriptors able to collect context values at run time and to perform decisions with additional context artifacts (new decision nodes, new Decision Input).

There are substantial research works on context-aware compositions, but few of them consider software service composition with the scope of model transformations. This section compares some context aware solutions with the CAComp proposition.

Process Definition Meta-model RFP [12] solicits submissions for process definition meta-model. We took particular attention to the following ongoing submission [1]. This proposal reuses a subset of the UML 2.0 meta-model and extends UML 2.0 with a Business Process Definition Package. In this proposition, a process is seen as a UML Component representing the external view of a process, denoting its contractual interface. Additionally, process flows are considered as associated behavior specified by an activity. We add to this proposition the possibility to describe the collaboration between components (i.e. services). This extension is necessary in CAComp because we are interested in the orchestration execution but also in the deployment of the services used by the collaboration.

Several propositions allow to model context information. Above them, the ContextUML [15] and the CML (Context Modeling Language) [6]. The first one is a UML model, it is similar to CAComp Context Meta-model but does not include observable entities and context aware entities. The second one models context, but not adaptations. CAMidO is a Context Aware Middleware based on an Ontology meta-model [5] which includes context and adaptations. CAComp context meta-model is a UML mapping of CAMidO ontology meta-model and an extension of this model for orchestrations.

PLASTIC [3] provides tools and methodologies to develop service-based context aware applications. The meta-model proposed by the authors is based on two levels of software description: service composition as an abstract layer and component compositions as a concrete layer where can be found deployed services. Context information is utilized during service discovery for negotiations between user and provider (trade-off between offered Qos and provided Qos).

CADeComp [4] is a context aware deployment tool for component-based applications. This tool is driven by a dedicated abstract meta-model, based on OMG D&C specifications [11], and follows MDA specifications. CADeComp describes context aware assemblies of components and produces target deployment plan. The context adaptation is driven by a set of adaptation rules executed by CADeComp tool at deployment time. CADeComp manages structural context adaptations of assemblies of components, whereas CAComp addresses adaptation for orchestration of services. CAComp takes into account behavioral adaptation through activity diagrams adaptations, and performs BPEL descriptor generation whereas CADeComp configures CCM Applications.

The meta-model presented in this article will be enhanced with a deployment meta-model on which will be based the model transformations to target platform models. The deployment level will include a context-aware discovery service to manage context-aware discovery of existing services.

References

[1] Business Process Definition Metamodel (BPDM), 2006. In Response to: Business Process Definition Metamodel RFP (OMG Document bei/2003-01-06).

[2] P. Aniorté and F. Seyler. Ugatze: Model driven egineering for component reuse. In *In Proceedings of 25th Latin American Computing Conference, Cali, Columbia*, Oct. 2005.

[3] M. Autili, V. Cortellessa, A. D. Marco, and P. Inverardi. A conceptual model for adaptable context-aware services. In A. Bertolino and A. Polini, editors, *in Proceedings of International Workshop on Web Services Modeling and Testing (WS-MaTe2006)*, pages 15–33, Palermo, Sicily, ITALY, June 9th 2006.

[4] D. Ayed, C. Taconet, G. Bernard, and Y. Berbers. An adaptation methodology for the deployment of mobile component-based applications. In *ICPS'06 : IEEE International Conference on Pervasive Services 2006*, pages 193–202, Lyon, France, June 2006.

[5] N. Belhanafi Behlouli, C. Taconet, and G. Bernard. An architecture for supporting Development and Execution of Context-Aware Component applications. In *ICPS'06 : IEEE International Conference on Pervasive Services 2006*, pages 57–66, Lyon, France, 26-29 June 2006.

[6] K. Henricksen and J. Indulska. Developing context-aware pervasive computing applications: Models and approach. *Journal of Pervasive and Mobile Computing*, volume 2(1):pages 37–64, Elsevier, 2006.

[7] A. Kleppe, J. Warmer, and W. Bast. *MDA Explained. The Model Driven Architecture: Practice and Promise*. Addison-Wesley, 2003.

[8] F. Leymann. Web services flow language (wsfl 1.0), May 2001. IBM.

[9] J. Miller and J. Mukerji. Model Driven Architecture (MDA). Technical report, Object Managment Group, July 2001. http:www.omg.org/docs/ormsc/01-07-01.pdf.

[10] OASIS. Oasis ws-bpel technical commity website. http://www.oasis-open.org/committees/wsbpel/charter.php, 2006.

[11] Object Management Group, Inc. *Deployment and Configuration of Component-based Distributed Applications*, June 2003. Draft Adopted Specification (ptc/03-07-02).

[12] OMG. Business Process Definition Metamodel Request For Proposal (RFP). Technical report, Object Management Group, 2003. bei/03-01-06.

[13] OMG. Unified modeling language version 2.0 superstructure, final adopted specification. Technical report, Object Managment Group, Aug. 2003. document ptc/03-09-15.

[14] F. Seyler and P. Aniorté. A model driven integration process to manage component interoperability. In *Software Engineering Research and Practice*, pages 104–110, 2004.

[15] Q. Z. Sheng and B. Benatallah. ContextUML: A UML-based Modeling Language for Model-Driven Development of Context-Aware Web services. In *The 4th International Conference on Mobile Business (ICMB'05), IEEE Computer Society. Sydney, Australia.*, July 11-13 2005.

[16] S. Thatte. Xlang: Web services for business process design., 2001.