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 through several UML2 diagrams. Component model is a tool for describing each service external interfaces. Collaboration diagrams describe the structural composition of services. And activity diagrams describe the orchestration of services.

Furthermore, the second contribution of this article is to mix the composition meta-model with a context meta-model. Thus, we propose to include the descriptions of context aware composition adaptations into the composition meta-model.

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. The article is illustrated through a concrete context-aware composition example. Our proposition is compared with other related works.

1. Introduction

One trend in software service development is to build services by composition of services. Orchestration and choreography of services are examples of composition of services. As far as now, these compositions are described through ad-hoc languages such as BPEL (Business Process Execution Language). In this paper, we propose to use the same meta-model (UML2 Unified Modeling Language version 2 [1]) for component composition and orchestration of services. One advantage of this approach is that the meta-model, through model transformations, is applicable to different platforms (Web Services, Orchestration Engines, Component Platforms).

With the emergence of mobile devices (e.g. PDA-personal digital assistant, smart-phones), it becomes essential that software service based applications offer context-awareness capabilities. Indeed, their configuration and behavior may depend on the execution context (e.g. network connection, end user-terminal capabilities, server engine type, sensor availability). The second issue addressed in this article concerns the expression of context awareness of software service compositions. We consider to include context awareness definition in abstract service composition (choreography and orchestration) through a context-aware composition meta-model. Context awareness involves the description of adaptation of service composition to relevant context situations. The composition adaptation has to be taken in charge by the middleware rather than by additional context adaptation code in the application. We argue that the expression of the service composition and of the composition’s context awareness must be defined during the application modeling process in a platform independent way.

CAComp (Context Aware Composition) is a middleware for the context awareness of compositions. It is based on the CAComp meta-model presented in this paper. The paper is structured in the following sections. Section 2 gives an overview of CAComp as well as its motivations and objectives. Section 3 describes the subset of UML2.0 meta-model necessary for CAComp composition meta-model. Section 4 describes the context meta-model and Section 5 provides a description of the context adaption meta-model, which provides means to build context-aware compositions of software services. Finally, Section 6 discusses some related works and Section 7 concludes the paper.

2. Motivations and illustrating example

In this section, we present our motivations and objectives. Then, we introduce the orchestration example which illustrates the article.
2.1. Objectives

The first purpose of our proposition is to abstractly describe context-aware composition of software services such as web service orchestrations. In this approach, we consider that the application designer may forecast the use of its software in different context situations and describe adaptation actions in these various situations. This abstract description is useful during the code production process, the deployment process and the execution process. We aim to obtain orchestration descriptions which may be used in different platforms and context situations rather than static orchestration descriptions.

During the development process, the meta-model allows the code producer to generate code for different platforms. During the orchestration deployment process, an orchestration engine is selected, then the model may be used to generate the process in the target orchestration language (e.g., Business Process Execution Language (BPEL) [2], XLANG [3] or WSFL [4]). Thus the same description may be used by different platforms. The services which participate to an orchestration may be discovered with constraints suited to context situations. Furthermore context information may be used during the deployment process of software services for configuration purpose. Finally, during execution, different context situations may lead to different execution flows.

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 software models. As several MDE framework, it exploits separation of concerns, and as Ugatze [5, 6] aims at exploiting non functional properties to perform model transformations. CAComp meta-model is made up by two main abstraction levels, following MDA (Model Driven Architecture) initiative [7, 8]: platform independent level for building Platform Independent Model of context-aware composition, and platform specific level.

In CAComp, the composition meta-model is a subset of UML2.0. Additionally the context-aware composition meta-model contains context and adaptation meta-model

In this article, we only present the abstract meta-model. We do not explain how the abstract meta-model is used for transformation and adaptation purpose.

2.2. Context-aware composition example

The meta-model presented in this article is illustrated with the Rugby Report Form Management Process. We briefly present this process in this subsection.

Nowadays, the need to develop information systems to report form management is becoming more apparent at amateur level of collective sports (rugby, football or basketball). 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 licence). The referee checks each licence 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 interned through an orchestration of services furnished by the regional league, the clubs, the referee and the delegate.

Additionally, the context situations on the play ground where the matches take place of several match report form processes are variable, namely the network bandwidth, final user terminals, topography, sensors availability. These variations may lead as we will see to various orchestrations.

3. Software Service Composition Meta Model

We propose to use component orientation to describe software service compositions. This approach enables CAComp to describe the software entities involved in the orchestration, and their connections. These descriptions are useful during the orchestration deployment process.

Component orientation naturally leads to define service orchestrations in both a structural and a behavioral view with UML 2.0 diagrams [1]. This section focuses on Component, Collaborations and Activity elements of UML2.0 meta-model. The UML2 meta-model is used in CAComp in one hand as a base to build a context aware meta-model, in the other hand as a target for context adaptation and model transformations.

We begin this section with service and orchestration definitions. Then, we present Component, Collaboration and Activity diagrams. This section is illustrated with diagrams extracted from the example scenario.

3.1. Software services and software compositions

A Software Service is usually considered as a particular software system that can be published, located and invoked across a network. Web services are example of software services, their interfaces are described in WSDL (Web

\footnote{In this paper, each meta-model element is expressed in italic.}
Service Description Language) [9]. They may also be described by UML interfaces and transformed to WSDL interfaces through model transformations. A set of interfaces offered by the same software entity may be defined in a same component. The component is a unit of deployment [10].

A Software Service Composition is a Service for which some operations necessitate to invoke operations from other Software Services. For example, a Web Service may itself be a composition of other existing web services. Obviously, a "composition language", which relates the series of operation to invoke when one of the composite operations is invoked is necessary. In the web service area, several languages such as Business Process Execution Language (BPEL) [2], XLANG [3] or WSFL [4] may be used to specify the "code" executed as a composite Software Service.

The interface of the composite service in itself does not specify the functional dependencies of the composite service, it does not either describe software execution entities. We propose in the next sections to describe composite services and orchestrations with UML2 diagrams.

3.2. Component diagram

Figure 1 presents a component view of the Match Management Orchestration. The MatchManagementProcess manages the Match Report Form updates during and after the match, and coordinates interactions with other software services.

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 (presented next subsection) specifies with components and connectors the structure of the collaboration.

3.3. Structural Composition: Collaboration

In UML2.0, a Collaboration describes a structure of collaborating elements connected through connectors and which collectively accomplish a functionality. This diagram may be used to identify independent components participating in the orchestration. This diagram is necessary to generate deployment diagrams which associate component instances to target computers.

Figure 5 in section 5 shows a collaboration between the components involved in the MatchManagementProcess. In this figure, the MatchManagementProcess is connected to the Referee, the RegionalLeague, the HomeClub, the VisitorClub, and the Delegate components. This collaboration illustrates a structural composition of Software Services. Internal behavior of each service, and especially of the MatchManagementProcess, can be described via a Behavior Model.

3.4. Behavioral Composition: Activity as behavior of Components

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 2 represents the behavior of the MatchManagementProcess component with an activity diagram.

The activity receives Date, RefereeName as input parameters (ActivityParameterNode). The activity is made up of several nodes and edges which describe the flow of execution. Firstly, the getMatchReportForm node gets a MatchReportForm from the external RegionalLeague². Then, the orchestration waits that the HomeClub and the VisitorClub fill the match report form. Afterwards, an AcceptEventAction indicates the referee presence for supervising the match. Then, the referee checks the teams licences, 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.

²Invocation of the getMatchReportForm operation defined in the ParticipantInformation interface of the RegionalLeaguePort.
In this section, we have illustrated main concepts for describing compositions of software services: components, collaboration, activities, as simplified subsets of the UML2.0 meta-model. We have shown that UML2.0 meta-model may be used to describe abstract services and orchestration of services. The advantage of this approach is that the meta-model may be used for model transformation especially during the orchestration deployment process. In the following sections (Section 4 and Section 5), we supplement this meta-model with context-awareness and context-adaptations.

4. Context Meta-Model

We have the objective to describe processes and compositions which have the ability to vary in their configuration, structure and behavior according to deployment and execution context. For this purpose, we propose to associate context and composition in the modelling process. In this section, we present the context meta-model used in the CACComp framework. The different parts of the context meta-model are illustrated by examples taken from the report form match management process model, and reported in Table 1. We present successively context and observation meta-model, collected and interpreted context meta-model and finally relevant situation meta-model.

4.1. Context and observations

In CACComp, 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, the term “entity” extracted from the definition becomes an ObservableEntity meta-class in the CACComp context meta-model, “any information” observed from an ObservableEntity becomes an ObservableContext. Among ObservableContext, those relevant for a given composition because they may lead to adaptations are RelevantContext. The “situations” which lead to adaptations are RelevantSituation.

Extracted from the match report form scenario, RefereeComputer and Referee are ObservableEntity, linked to a set of ObservableContext including respectively Memory-Context and GPSLocationContext.

Each observable context is linked to Observation. An observation is described by a value and a time of observation. An ObservableContext owns a data-type, which determines the observed value data-type. For example, ScreenSizeContext owns an Integer data-type and GPSLocationContext owns a Location composite data-type made up of two attributes: longitude and latitude. An observable context may either be directly collected (CollectedContext), or interpreted from other contexts (InterpretedContext). The following subsection describes these two kinds of contexts.

4.2. Collected and interpreted context

A CollectedContext is an observable context directly linked to a Sensor from which data are collected: ScreenSizeContext, NetworkBandwidthContext, GPSLocationContext are CollectedContext. For CACComp, a Sensor is a software component which provides an operation to collect context observations. This component may or may not be linked to a physical sensor. The task of a CACComp sensor collection operation may be to always return the same stable value, read from a file, or collected by a physical sensor (e.g. a thermometer, a GPS). The advantage of including sensor descriptions to the meta-model is to give the possibility to easily change the context collection process by changing the association from a CollectedContext to a Sensor from one instance of a sensor to another whenever it is necessary without any code modification. An InterpretedContext observation is obtained by a computation on one or several other source ObservableContext. It is deduced by an InterpretationOperation. The InterpretationOperation is a software operation which receives the source observations.
as input parameters and return the interpreted observation. For example, the GPS Location of the referee GPSLocationContext is used as a parameter of InterpretedContext to obtain the district \(^3\) (DistrictContext).

### 4.3. Context Aware Entity, Relevant Contexts, Relevant Situations

In the two previous subsections, we have defined the concepts of observable entities and observable contexts. In this section, we define more precisely the context entities relevant for a given system (\(i.e.\) a software composition or a process). 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.

At a given time, a Context Aware Entity may be in one or more **RelevantSituations**. Those situations may lead to adaptations. A **RelevantSituation** is a set of **ContextSituations**, associated to a context aware entity. 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. The current situation(s) is(are) obtained by a computation from the current observations of the relevant contexts. This computation is described in the meta-model either by an operation or by a logical expression.

Several relevant situations extracted from the report form match management scenario are described in table 1. These context situations are associated to the context-aware entities MatchManagementActivity and MatchManagementProcess Component. They will be used in Section 5 to illustrate adaptations.

In a software application, each context aware entity can be aware of a set of relevant contexts. The CAComp context meta-model enables composition designers and packagers to describe the context and situations which lead to adaptations in the compositions. The following section analyzes which of the entities studied in Section 3 may be context-aware entities. It also presents the kinds of adaptation that may be defined with the CAComp meta-model.

<table>
<thead>
<tr>
<th>Concept</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Meta-Class Observable-Context</td>
<td>ScreenSize, GPSLocation</td>
</tr>
<tr>
<td>Meta-Class Observation</td>
<td>Observation of GPSLocation (date: 08-02-07, value: (45N,45w))</td>
</tr>
<tr>
<td>Meta-Class ObservableEntity</td>
<td>RefereeComputer, MatchManagementComputer, DelegateComputer, Referee</td>
</tr>
<tr>
<td>Meta-Class Collected-Context</td>
<td>GPSLocationContext</td>
</tr>
<tr>
<td>Meta-Class Sensor</td>
<td>GPSSensor</td>
</tr>
<tr>
<td>Meta-Class Interpreted-Context</td>
<td>DistrictContext deduced from GPSLocationContext input parameter</td>
</tr>
</tbody>
</table>

| Meta-Class ContextAwareEntity | CaMatchManagementCollaboration and CaMatchManagementActivity are examples of ContextAwareEntity. |
| Meta-Association RelevantContexts | dateOfTheDayContext and RefereeNameContext are RelevantContext of CaMatchManagementActivity |
| Meta-Association RelevantSituations | DelegateAvailable and DelegateUnAvailableContextSituation are RelevantSituation of CaMatchManagementCollaboration and CaMatchManagementActivity |
| Meta-Class ContextSituation | ManagementDelegateSameNetworkId is a LeafContextSituation corresponding to MatchManagementComputer.NetworkId = DelegateComputer.NetworkId |

\[ \text{DelegateAvailable is a CompositeContextSituation made up of:} \]

\[ \text{ManagementDelegateSameNetworkId & ComputerUsableNetworkConnection} \]

\[ \text{DelegateUnAvailable is CompositeContextSituation made up of:} \]

\[ \text{!DelegatAvailable} \]

---

\(^3\)specifies rugby territory subdivision
5. Context Adaptation Meta-Model for Service Composition

Context Adaptation meta-model is essential for the CA-Comp 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.

The context adaptation may be processed at different time in the service lifetime: service development, service packaging, service deployment on the computers, service instantiation, service execution. In this article, we put the stress on adaptation taking place during the deployment process. At this time, configuration of the structure of the assembly, configuration of the component and assembly attributes and configuration of the process may take place.

In this section, we present first the composition context aware entities that we have identified. Then, we describe the kinds of adaptations which are likely to be applied to composition context aware entities. For this description, we follow the taxonomy of software adaptation defined in [11]. We present first structural adaptations, then behavioral adaptations.

5.1. Composition Context Aware Entities and Adaptations

In an application model, several model elements can be context-aware. In CAComp, we apply context awareness on both the composition structure and the collaboration behavior. Thus, as depicted in Figure 3, we consider two kinds of context aware entities: CompositeContextAwareEntity which is made up by optional and mandatory ContextAwareEntity. Additionally, Collaboration, Component and Activity are CompositeContextAwareEntity, and Connector, ActivityEdge and ActivityNode are LeafContextAwareEntity.

Figure 3. ContextAware Entities in CaComp

As illustrated in Figure 4, we consider several kinds of Adaptation in CAComp. A Mapping allows the designer to define a direct mapping between one attribute of the model to an observable context current observation. ChoiceAdaptation allows CAComp to choose between several elements according to current relevant situation. Finally, Presence enables the designer to condition the presence of an element in the model to a relevant situation. We illustrate these kinds of adaptations in the example scenario.

Figure 4. Adaptations in CaComp

5.2. Structural Adaptations

A change in the structure of a Collaboration: change of a Component or change of a Connector is called a structural adaptation.

For example, in the Match Management Collaboration (see Figure 5), the MatchManagement component is connected to the Delegate component. During this collaboration, MatchManagementProcess needs the Delegate to register the match events. However, for several reasons (level of the Championships, infrastructure of the local stadium), a Delegate is not always designated by the RegionalLeague. In these cases, the Match Event Form is hand written by designated home representatives, it will be checked and integrated by the referee after the match. Therefore, in the scenario model we consider the possibility that the Delegate may be unavailable in the collaboration.

As shown in Figure 5, this possibility is modelled thanks to a ChoiceAdaptation. CACollaboration is a composite context aware entity which owns several caMandatoryElement: MatchManagement, RegionalLeague, Referee, HomeClub, VisitorClub component and five connectors. It also owns two caOptionalElement: Delegate component and one connector. This ChoiceAdaptation owns two Choice. The default choice is optimistic, it considers that the network connection is good and connects the MatchManagement component to the distant Delegate component. The second choice is explicitly linked to the following relevant situation: CAMatchManagementCollaboration is in DelegateUnavailable: Delegate component and associated connector are used in the collaboration.
5.3. Behavioral adaptation

Behavioral adaptations are applicable on component behavior (activity, operation, process). We give below examples of behavioral adaptations. The MatchManagementActivity model illustrates mapping of operation parameters in an activity diagram. The CheckMRFActivity model illustrates the choice of an operation.

In Figure 6, caMatchManagementActivity is a context-aware behavior of the MatchManagementProcessComponent. According to Figure 3, caMatchManagementActivity is a composite context aware entity. It is aware of a set of relevant contexts (see Section 4.3), namely Referee.UserNameContext and RefereeComputer.DateOfTheDayContext. Two ActivityParameterNode (Date and RefereeName) are considered as ContextAwareEntity. They are associated to a mapping adaptation. Mapping are performed on these parameter nodes. During the adaptation process, DateOfTheDay observation value is assigned to the Date activity parameter node, and Referee.UserNameContext observation value is assigned to the RefereeName activity parameter node.

Figure 7. Choice Adaptation in MatchManagementActivity Model

caMatchManagementActivity is a composite context aware entity. It is aware of the following relevant contexts: MatchManagementComputer.NetworkId, DelegateComputer.NetworkId.

Additionally, caMatchManagementActivity is aware of the following relevant situations: DelegateAvailable or DelegateUnAvailable. A part of this context-aware Activity model is illustrated in Figure 7. In this activity, all the context aware entities included in the dashed rectangle are optional elements: eight activity edges, a fork node, a merge node, a call Operation Action and two object nodes. A ChoiceAdaptation is associated to CheckMRFActivity. The default choice associates DelegateAvailable, to this set of elements. When the MatchManagement 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.

The CAComp adaptation meta-model enables composition designers to define variations in the composition and in the process modelling. These variations may be structural and behavioral. During deployment and execution, this information will be used by CAComp to choose the structure and the behavior of the composition and orchestration.
6. Related work

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.

Model Driven Engineering for software Service Composition

Process Definition Meta-model RFP [12] solicits submissions for process definition meta-model. Submissions have to define an abstract language for specification of executable business processes that execute within an enterprise (with or without human involvement), and may collaborate between otherwise-independent business processes executing in different business units or enterprises. If we consider Software Service Orchestrations and Choreographies as "automated business process", the objective of this RFP are close to our needs in Software Service Composition. We took particular attention to the following ongoing submission [13]. 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.

Context Modelling

Several propositions allow to model context information. Above them, the ContextUML [14] and the CML (Context Modelling Language) [15]. 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[16] which includes context and adaptations. CAComp context meta-model is a UML transformation of CAMidO ontology meta-model and an extension of this model for orchestrations.

Context Adaptation of Software Service Composition

PLASTIC [17] 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).

Model Driven Engineering for context aware deployment

CADeComp [18] 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 [19], 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.

7. Conclusion

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 UML 2.0 activities. The context meta-model is an implementation of CAMidO meta-model, 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 compositions of Web Service 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 will be used at deployment time to produce composition adapted both to the target platforms and to the current context situations. At deployment time it will produce platform-specific models and descriptors. 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


