A Modeling Approach for Adaptable Service Oriented Systems based on Domain Specific Language Engineering

M. Lethrech\textsuperscript{1}, A. Kenzi\textsuperscript{2}, I. Elmagrouni\textsuperscript{1}, M. Nassar\textsuperscript{1}, A. Kriouile\textsuperscript{1}

\textsuperscript{1}SIME laboratory, ENSIAS
Mohammed V Souissi University
Rabat, Morocco
{mohammed.lethrech, issam.elmagrouni} @ um5s.net.ma
{nassar, kriouile} @ensias.ma

\textsuperscript{2}ENSAF
Sidi Mohamed Ben Abdellah University
Fes, Morocco
adil.kenzi@gmail.com

Abstract— Nowadays, Model Driven Software engineering (MDSD) is a very dynamic field of research. This approach has introduced a major evolution to software engineering. Nevertheless the full code generation always stays a not reached objective.

On the other side the simultaneous use of Context Aware Computing (CAC) and Service Oriented Computing (SOC) paradigms must raise many challenges. Particularly, the challenge of engineering such systems, which consists of the definition of modeling approaches, processes, techniques and tools to facilitate construction of these systems.

The use of Domain Specific Modeling (DSM) approach for Context Aware and Service Oriented Systems brings an answer to these concerns. In this paper, we propose a DSM approach for adaptable service oriented systems named CADSSO (Context Aware, Domain Specific and Service Oriented) modeling approach.

Our modeling approach is based on five models: domain specific services model (conforms to a domain specific services meta-model) which represents the domain specific services; service variability model (conforms to our service variability meta-model), it formulates services forms of adaptation; domain specific context model (conforms to our context meta-model), symbolize the services context of use; adaptation rules model, which is the joint between service variability model and domain specific context model; and domain specific business rules model, used to model domain specific business.

Keywords- DSM; SOC; CAC; MDSD; Adaptation

I. INTRODUCTION

Currently Model Driven Software engineering (MDSD) is a very dynamic field of research. Certainly, this approach has introduced a major evolution to software engineering. Nevertheless the full code generation always stays a not reached objective.

On the other side the simultaneous use of Context Aware Computing (CAC) and Service Oriented Computing (SOC) paradigms must raise many challenges. Particularly, the challenge of engineering such systems, which consists of the definition of modeling approaches, processes, techniques and tools to facilitate construction of these systems.

CAC has recently emerged as a new computing paradigm promising adaptable systems development. It provides techniques for developing pervasive computing applications that are flexible, adaptable and can react to context changes [4].

Also, SOC is a new software engineering approach, favourable for the rapid development of interoperable and scalable distributed applications [2][8]. These services are loosely coupled by nature, allowing addition, modification and suppression of services in a fast and efficient manner.

The use of Domain Specific Modeling (DSM) approach for Context Aware and Service Oriented Systems brings an answer to the concerns quoted at the beginning of this introduction. Adoption of DSM approach in software engineering provides a lot of advantages, essentially a better reactivity to business rules and technological changes in addition to a high productivity and an excelling quality of a generated code [2][9][10]. DSM approach is mainly based on two principles. Firstly, elevation of abstraction level by modeling the solution with a Domain Specific Language (DSL); the DSL uses directly the concepts and business rules of a specific domain. Secondly, the full automatic generation of the final solution from the high level specifications.

In this paper, we propose a DSM approach for adaptable service oriented systems named CADSSO (Context Aware, Domain Specific and Service Oriented) modeling approach. Our approach rests on five models. The first one is the domain specific services model; it takes care of services modeling. The second is services variability model; it formulates services forms of adaptation. The third is the domain specific context model, which is responsible for modeling the system context of use. The fourth is adaptation rules model; it makes the join between services variability model and domain specific context model. The last one is domain specific business rules model, which is in charge of domain specific business rules modeling. For a specific domain, the language developer must produce the domain specific service meta-model as an extension of our generic service meta-model (abstract language), the language (concrete syntax), the domain framework and the code generator.
This paper is organized as follow; the second section presents the related work, the third section describes our CADSSO modeling approach. The latter is illustrated with a tax case study in the fourth section. Finally the article ends with a conclusion and outlook.

II. RELATED WORK

In our DSL and SOA explorative study [2], we have studied a lot of DSLs for SOA, concerning a variety of specific domains; security, orchestration, quality of service QoS…. The main conclusion of our comparative study is that adaptation mechanism is almost absent in all studied DSLs for SOA.

For Vale and Hammoudi [15], a specific context owns its context-aware tasks which are in relation with services. Service variability and adaptation rules are absent in their work.

Soo Ho Chang et al. [12][13][14] focused on service variability modeling. Service and context modeling are out of the scope of their work.

Boukadi [6] uses Aspect Oriented Programming (AOP), with an UML profile, for modeling services. She uses an ontology for context modeling of inter-enterprises cooperation domain. She did not deal with service variability modeling.

Kenzi, El Asri, Nassar and Kriouile [17][18] defined an UML profile (VSoaML) for modeling adaptable service oriented systems. They treat the service adaptation with multi-view service concept. Context modeling is out of the scope of their work.

In [16], the authors also defined an UML-Based Modeling Language for Model-Driven Development of Context-Aware Web Services. They defined the element CAObject (Context Aware Object) which is a generalization of service, operation…. Their work doesn’t cover service variability modeling.

In [11], the authors specify a service adaptation approach based on UML, AOP and MDA with a context meta-model.

Hafidi et al. [19] defined a pattern for modeling context-awareness of services, based on an UML profile, AOP and a key value context meta-model.

The latest version of SOAML [20] doesn’t support variability and context awareness.

III. CADSSO MODELING APPROACH

CADSSO MDSD approach aims to facilitate modeling and development of adaptable service oriented systems, using DSM approach. In order to have adaptable services, service variability should be modeled and designed at the first stages of a modeling approach. Also, separation of concerns is the cornerstone of system flexibility. In fact, our modeling approach ensures a separation between service variability, context and service adaptation rules.

In this paper, we have been focusing on the modeling side of our approach: context modeling, service modeling, service variability modeling, adaptation rules modeling and domain specific business rules modeling.

A. Context modeling

We adopted the context definition of Dey and Abowd [21]:

“Any information that can be used to characterize the situation of an entity. An entity is a person, place or object that is considered relevant to the interaction between a user and an application, including the user and the application themselves.”

There is a growing body of research on context meta-models [3][15][16][19][11][22][6][24]. Our proposed context meta-model is directly joined to the services variabilities. Each service is adaptable based on his service variation points.

For us a Context is composed of ContextElements and a ContextElement is composed of ContextParameters. We have also defined the ServiceContextElement entity which is a ContextElement specific to one Service. The ServiceContextElement can contain ContextParameters from different ContextElements, it can also contain its own ContextParameters which is specific to the service. Each ServiceVariationPoint (from service variability meta-model) is joined to one instance of ServiceContextElement.

The context can be divided into a lot of sub-contexts. This division can be based on the source of sub-context elements, such as: environment (temperature, position, battery level...), user (user profile), computational, sensed …etc [4][11][6]. We have left the context meta-model open to support any type of sub-context. Our context meta-model is illustrated in the figure 1.

B. Service modeling: Generic service meta-model

We have categorized the service entity into three categories: Business service, Utility service and Scheduling service. The latter simply uses the already implemented BusinessServices through ShedulingStrategies (for example see CTRestitutionProcess in the case study). The BusinessServices (for example, see CTCalculationService in the case study) implements business activities and the UtilityServices constitute non functional activities. The ShedulingStrategy determines the services variabilities and the orchestration strategy used by a SchedulingService. Each Service has at least one ServiceVariationPoint which represents a variability of the adaptable service. A service has one or more ServiceOperations
and a ServiceOperation has ServiceParameters; a Service also has one or more Service Level Agreements (SLA). According to service attribute “type”, the implementation will change (big web service, restful web service…etc). Each SLA has conditions which are boolean expressions. Each condition has a ServiceVariationPoint, which represents a service user, uses a service through a SLA. A ServicePackage groups a lot of services. The ServiceInterface make it possible to explicitly model the operations provided by a service using this interface. A service has just one BaseServiceInterface which groups operations provided for all service users. A ServiceOperation has a set of ServiceParameters. A Process is a “SchedulingService”. We use the “process-as-a-service” approach in all the life-cycle of a process. Finally, BusinessServices can use UtilityServices. The relation “uses” (from service A to service B) means that the service A can call the service B. Our generic service meta-model is represented in the figure 2.

The language developer adds his domain elements to our generic service meta-model to have his domain specific meta-model. Our extension mechanism is based on specialization relation. We have illustrated our approach by tax meta-model (see the case study). All our meta-models are in conformity with the MOF (Meta-Object Facility) model [23].

C. Service variability modeling

Service Variability is a characteristic (logic, graphical presentation, persistence … etc) which may vary within a service, according to service context of use. The service variability is represented with variation points. A variation point is a place in service where the difference occurs.

Each service has a lot of variation points [2][12][13][14]. We have categorized these variation points into two categories: FunctionalVariation and TechnicalVariation. The former is divided into three classes: Logic, for the business logic variation; Interface, for methods signature and methods offered -by the service- variation and SchedulingStrategy, for the variation of the sequence of invocation of services used by a process. TechnicalVariation is divided into three classes:

D. Adaptation rules modeling : adaptation logic

In this section we treat the junction of the service variability model to the context model. The adaptation rules are modeled using matrixes. The columns represent the ServiceContextElement parameters; which includes ContextParameters from different ContextElements. The lines contain the service variation points. We have one matrix by service. The same is for processes adaptation rules, each process (service) has an adaptation matrix; the columns represent ServiceContextElement parameters and the lines contain the process SchedulingStrategies.

It is possible to use the “logical or”, the “logical and” and the parenthesis to form a logical expression using ServiceContextElements parameters (see the case study).
For domain specific business rules modeling, the domain developer must produce a domain specific language.

IV. CASE STUDY: TAX RESTITUTION

In this paper, we have chosen the tax domain to illustrate our approach. The tax calculation and restitution algorithms are constantly changing based on the budget law. Therefore, tax information system must be as flexible as possible.

A. Domain specific service meta-model construction

For a specific domain, the language developer must produce its domain specific service meta-model as an extension of our generic service meta-model (defined in the section III.B). Thus, we have to add our tax elements to our generic service meta-model to produce our tax restitution extension of our generic service meta-model (defined in the section III.C).

This latter is divided into Generic service Meta-model, Domain Specific Meta-Services and Domain Specific Meta-Elements.

1) Domain specific meta-services: Tax Restitution

For tax restitution we have the following BusinessServices: DemandDeposit, AdvanceRestitution, RestitutionRecordVerification and TaxRestitution. The DemandDeposit service allows deposit of restitution demand; AdvanceRestitution handles an automatic restitution based on the ratepayer category (A, B or C). The RestitutionRecordVerification service performs verifications of the restitution folder (invoices …etc); TaxRestitution service performs the final tax restitution.

2) Domain specific meta-elements

A Tax can have a lot of TaxDeclarations (Declaration and Bundle for corporation tax); a Ratepayer can file a lot of TaxDeclarations and a RatePayer can have a lot of Exemptions.

TaxRestitutionSLA represents the SLA of tax restitution. The figure 4 represents our tax calculation and restitution service meta-model.

According to our tax meta-model, we can create as much models as taxes. The above example is the service model of Corporation Tax (CT) Restitution.

B. Domain specific Models

1) Corporation Tax restitution service model (conforms to our domain specific meta-model constructed in section IV.A)

The restitution of corporation tax is a SchedulingService named CTRestitutionProcess. It uses four business services: CTRestitutionDemand, CTRestitutionRecordVerification, CTRestitutionFolderVerification and CTRestitution. They have the same role like described in the meta-model but specific to corporation tax. CTRestitutionProcess has its specific SLA called CTRestitutionSLA. Our corporation tax restitution service model is illustrated in the figure 5.

2) Corporation Tax service variability model (conforms to our context meta-model defined in section III.C)

In this section we will identify the variation points of each service. Each service has at least one service variation point. For a client request, just one service variation point is used for a given service.

a) CTRestitution: has two logical variation points:

• CTRestitutionCatA: is used for taxpayers whose category is “A”. 80% of advance restitution;
• CTRestitutionCatB: is used for taxpayers whose category is “B”. 50% of advance restitution.

b) CTRestitutionProcess: has three sequential scheduling strategies. The three strategies are differentiated by the use of CTRestitution service:

• CategoryAStrategy: uses the CTRestitutionCatA variation point of the CTRestitution service;
• CategoryBStrategy: uses the CTRestitutionCatB variation point of the CTRestitution service;
• CategoryCStrategy: doesn’t have advance restitution.

The figure 6 gathers all corporation tax services variabilities.

3) Corporation Tax context model (conforms to our context meta-model defined in section III.A)

Our TaxContext model is composed of one subContext, CTContext for corporation tax and obviously we can add other sub-context for other taxes.

The CTContext is composed of two ContextElements, RatePayer and CTRestitution. The former is composed of four ContextParameters: Resident, Exemption, TaxRegim and Category. The latter contains just one ContextParameter named...
With Annex. The corporation tax context model is represented in the figure 7.

From the above model, and for corporation tax, we have identified for each service a ServiceContextElement: CTAdvanceRestitutionSC and CTRestitutionProcessSC. Each ServiceContextElement is responsible, respectively, of variations of CTAdvanceRestitution and CTRestitutionProcess services. Our ServiceContextElements are illustrated in the figure 8.

4) Adaptation rules model
In this section we will model adaptation rules of each service. Each line of the matrix joins a service variation point to its context parameters values.

a) “CTAdvanceRestitution” adaptation Rules
For corporation tax restitution we have two different treatments, one for ratepayer A category and the other for B category (see table I).

<table>
<thead>
<tr>
<th>Service variation point</th>
<th>RatePayer.Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>CTAdvanceRestitutionCatA</td>
<td>A</td>
</tr>
<tr>
<td>CTAdvanceRestitutionCatB</td>
<td>B</td>
</tr>
</tbody>
</table>

b) “CTRestitutionProcess” adaptation Rules
Also restitution process has three Scheduling Strategies, one for each ratepayer category (A, B or C) (see table II).

<table>
<thead>
<tr>
<th>Service variation point</th>
<th>RatePayer.Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>CTAdvanceRestitutionCatA</td>
<td>A</td>
</tr>
<tr>
<td>CTAdvanceRestitutionCatB</td>
<td>B</td>
</tr>
<tr>
<td>CTAdvanceRestitutionCatC</td>
<td>C</td>
</tr>
</tbody>
</table>

5) Tax restitution business rules model: Corporation Tax
For domain specific business rules modeling, the domain developer must produce a domain specific language. Finally our fifth (domain specific services model, services variability model, domain specific context model, adaptation rules model and domain specific business rules model) models must be transformed by the code generator the produce the full source code of the modeled system.

The domain framework essentially removes duplication from the generated code, hides the target environment and facilitates integration with existing code [9]. The figure 9 illustrates our CADSSO MDSD approach.

V. CONCLUSION AND OUTLOOK
We have proposed a DSM approach for adaptable service oriented systems named CADSSO (Context Aware, Domain Specific and Service Oriented) modeling approach. To develop such Context Aware services, the service variability must carefully be analyzed and designed in the first stages of a modeling approach.

In this paper we have focused on the modeling side. This latter is composed of five models: domain specific services model, domain specific context model, service variability model, adaptation rules model and domain specific business rules model. For a specific domain the language developer must produce his specific service meta-model based on our generic service meta-model.

We are working on the production of the code generator and framework for our tax calculation and restitution domain. Also we will produce the tax calculation concrete syntax and a graphical modeling tool for our domain specific.
REFERENCES


