ABSTRACT
Service Oriented Architecture (SOA) is an architectural style that is widely used in distributed and dynamic systems. The Service oriented architecture Modeling Language (SoaML) is an OMG standard for modelling SOA independent of a technology. This paper presents a tool for modelling SOA using SoaML and generating OSGi Declarative Services Models from SoaML models. SoaML metamodel has been implemented as an Ecore model using the Eclipse Modeling Framework (EMF). An Eclipse plug-in that allows architects to graphically design SoaML models has been developed using the Graphical Modeling Framework (GMF). We have also implemented a model transformation using ATLAS Transformation Language (ATL) in order to partially generate Declarative Services models. The generated model is used as a Declarative Services Component Description XML specification which is needed to execute code on the OSGi service oriented platform. In this way, we provide SoaML with Model Driven Architecture support.

Categories and Subject Descriptors
D.2 [Software Engineering]: Tools, CASE

General Terms
Design, Languages.

Keywords
SOA, Modeling, Model Driven Architecture.

1. INTRODUCTION
Nowadays, Service Oriented Architecture (SOA) [16] is becoming one of the most promising architectural styles used on the web, or for distributed and dynamic systems. SOA is based on concepts such as loose coupling and location transparency which allow services to be autonomous and context independent. Services do not know who their clients will be. Owners of these services can change their definitions or implementations without affecting on their clients. Once services are published, they can bind, dynamically, after clients search and discover them.

SOA can be implemented with many application technologies such as J2EE, CORBA or Web Services. Research efforts focusing on developing applications based on SOA concepts is required instead of focusing on technological solutions. In this way, designers can focus on understanding the problem domain rather than dealing with implementation details. In this context, following a model driven approach [14] seems appropriate to fulfill the separation of technological models from technology independent ones. Tools are needed to facilitate the construction of SOA technology independent models and automate the transformation of these models into specific technologies. Model Driven Architecture (MDA) is the Object Management Group (OMG) [10] standard for implementing a model driven approach by providing a set of tools that manage models.

This paper presents a tool for MDA implementations from modeling SOAs and generating platform specific models. The tool is based on using SoaML [16] as a platform independent model, which is an extension of UML 2.0 for modeling services. The tool is an Eclipse plug-in, which provides facilities for the graphical description of service oriented applications. Then, the SoaML models can be partially translated into an OSGi Service Model [13] called Declarative Services [12]. The generated Declarative Services Model XML documents are used for publishing, registering and accessing services in the OSGi platform.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
PESOS’10, May 2-8, 2010, Cape Town, South Africa
Copyright (c) 2010 ACM 978-1-60558-963-3/10/05 .. $10.00

* The work reported in this paper is based on Rukmani and Rajalaxmi’s final year projects which they carried out while visiting Lero between March and September 2009 under EURECA project.
The paper is organized as follows: Section 2 briefly presents the SoaML metamodel. In Section 3, we describe how we have provided SoaML with MDA support. In Section 4, we describe the tool from a user point of view. In Section 5, we discuss the advantages and limitations of our tool. Section 6 presents related works. Finally, Section 7 presents conclusions and further work.

2. Service oriented architecture Modeling Language (SoaML)

SoaML is an OMG specification for modelling service oriented architecture [17]. It consists of a SoaML UML profile and a metamodel that extends the Unified Modeling Language (UML) 2.0 (see Figure 1). The presented tool implements the SoaML metamodel. There are already tools that implement the SoaML UML profile such as the Objecteering tool [9].

This section presents only basic concepts of SoaML. These concepts are based on the revised UPMS submission presented in [6]. In the following, the concepts are presented:

- **ServiceInterfaces** describe the operations used between a service provider and a service consumer from the perspective of the provider. ServiceInterfaces are used as a type of a ServicePoint or a RequestPoint. A ServiceInterface can imply the realization and usage of one or more UML interface.

- **ServiceContracts** define the terms, conditions, interfaces and choreography that interacting participants must agree. They specify how services are provided and consumed based on interactions and behaviours involving the participants.

- **Participants** allow defining the service providers and consumers. When a Participant is a provider it contains at least a ServicePoint. A ServicePoint defines a capability offered by one entity to others. When a Participant is a consumer it contains at least a RequestPoint. A RequestPoint defines the connection point through which a Participant makes requests or consumes services. A participant can be a consumer, a provider or both.

- **ParticipantArchitectures** are the high level view of a SOA that defines how a set of participants work together for providing and using services. ServiceContract instances can be included in an architecture and when they are related to a Participant it implies that the Participant fulfils the contact.

- **ServiceChannels** provide a communication path between consumer Requests (ports) and provider services (ports).

3. Supporting SoaML with MDA

The MDA approach we follow is shown in Figure 2. The models that we have used are: SoaML as a Platform Independent Model (PIM), and the Declarative Services Model of OSGi as a Platform Specific Model (PSM). We have chosen to use the Declarative Services (DS) Model as a Component Model for publishing, finding and binding OSGi services. The transformation between the two models is done through the ATL transformation language.

We have developed an ECLIPSE plug-in for graphically modeling SoaML models (SoaML Editor) and a plug-in for generating DS models (ATL Configurator). In the following sections we explain how we implemented the SoaML and Ecore metamodels, the SoaML graphical Editor and the Transformations.

![Figure 1. Partial SoaML metamodel taken from [6]](image)

![Figure 2. Tool Components](image)
defined our two metamodels: SoaML and OSGi Declarative Services as Ecore models.

The SoaML metamodel specified in [6] was used to create an Ecore model for SoaML. Since SoaML is an extension of UML 2.0, we loaded the UML 2.0 Ecore metamodel provided by EMF in the Ecore editor and extended it with the SoaML concepts. The genmodel was created from it and subsequently the code was generated. Figure 3 shows the SoaML metamodel after it is introduced as an Ecore model and Figure 4 shows the properties window of the RequestPoint SoaML model element.

![Figure 3. Screenshot of the SoaML Ecore model](image)

The SoaML metamodel specified in [6] was used to create an Ecore model for SoaML. Since SoaML is an extension of UML 2.0, we loaded the UML 2.0 Ecore metamodel provided by EMF in the Ecore editor and extended it with the SoaML concepts. The genmodel was created from it and subsequently the code was generated. Figure 3 shows the SoaML metamodel after it is introduced as an Ecore model and Figure 4 shows the properties window of the RequestPoint SoaML model element.

![Figure 4 The properties window of the RequestPoint class](image)

In addition to extending the UML 2.0 Ecore model, we included validation rules. These are needed to enforce that models created by a user are correct. The validation rules that we have implemented for ServiceChannel are the following:

- One end of the service channel should be a RequestPoint and the other end should be a ServicePoint.

- The interfaces required by one end of the channel should be provided by the other end and vice versa, i.e., the request and the service in the channel should be compatible with each other.

Validation rules are enforced by implementing diagnostic methods of the EValidator interface, which delegate to the EMF Validation Framework for evaluating all constraints on a sub-tree of a model. While creating a model from the SoaML metamodel and when validating that model, the EValidator methods are called. The implementation contains a condition that violates the invariant. If the condition is false, then the validation is correct. If the condition is true, then a new BasicDiagnostic object is created and added to the list of diagnosis. On the user end, this will give a warning or an error depending on the severity that is set for the particular diagnosis.

The Declarative Service Metamodel is designed based on the Declarative Services Schema given in [11]. This DS Schema is designed based on the OSGi DS concepts and rules. To define this metamodel in Ecore, we only had to use the inbuilt facilities of the EMF and convert the XML schema into an Ecore metamodel.

### 3.2 SoaML Graphical Editor

It is possible to create a SoaML models Editor using EMF. However, a model is only represented through a tree structure (see Figure 8). Thus, visual characteristics that show how elements are related cannot be noticed in a friendly way. The different entities of the model can only be differentiated by the tags e.g., <Service Contract>. In addition, there are no modeling facilities such as drag and drop. Therefore, a Graphical Modeling Framework (GMF) [6] Editor has been developed for SoaML.

To create an editor in GMF, there are four steps to be followed:

1. Define the metamodel in Ecore;
2. Create a gmfgraph which contains the figures of each element in the metamodel as they need to appear in the canvas;
3. Define a gmf tooling which implements the palette that has to appear in the graphical editor and;
4. Create the mapping definition which combines an Ecore model, a gmfgraph, and the Tooling definition. In this way, we created the SoaML GMF Editor.

### 3.3 Transforming SoaML Models into OSGi Declarative Services Models

The first step we performed to transform the SoaML into OSGi DS is to identify the mappings. To do so, we implemented several simple examples using both SoaML and DS.

Figure 5 explains how we have identified the mapping for a service provision and registration. We have modeled a participant called VoiceTelephonyProcessor in the SoaML model which provides the service (ServicePoint) called CallProcessing. We also used the DS model to represent the VoiceTelephonyProcessor. The SoaML Participant is mapped into an OSGi Component and Implementation, and the SoaML ServicePoint is mapped into a DS provided interface.
Once we identified the mappings, we implemented the model transformations using ATL. We have used the ADT (ATL Development Tooling) environment. Figure 6 shows the ATL rule that is implemented for transforming a Participant into a Component. The rule also transforms the RequestPoints and ServicePoints of participants into DS Reference and Provider interfaces.

The ATL transformation generates DS Ecore models. The EMF facilities can be used to have a textual view of the DS Ecore models. These views are in XML. Since DS Components are specified in XML, these can be used along bundles for registering, publishing and accessing services.

4. Usage of Tool
This section explains how the current version of the tool can be used from a user point of view.

Figure 7 shows the user interface of the SoaML graphical editor. The editor consists of a Palette with the SoaML concepts (marked with 1). The user has to drag the concepts of the Palette into the canvas or drawing area (marked with 2). Not all the SoaML concepts can be dropped directly in the drawing area. For example, ServicePoints or RequestPoints can only be part of participants. If the user points over a Participant the editor shows a box above the Participant to indicate that the user can define ServicePoints or RequestPoints. When the user points over the canvas area (see Figure 7, marked with 3) a box is shown with the concepts that can be added.

--- ATL Grammar Rule to transform Participant Model Element to DS Model Element

``` ATL
rule ParticipantToDocumentRoot {
  from p : Soa!Participant
to d : DS!DocumentRoot
--Binding between DocumentRoot and Tcomponent
  (component<-c),
c : DS! Tcomponent
--Bindings Between Tcomponent and its children
  (name<-p.name, implementation<-i, service<-s, reference<-rList),
i : DS! Timplementation
--Initializing the Timplementation.class value
  (class<-p.name+'Implementation'),
s : DS! Tservice
--Binding between the Tservice and the set of Tprovide
  (provide<-prList),
prList:distinct DS! Tprovide
--For each ServicePoint a Tprovide is created and values are initialized
  foreach(prName in p.getServicePointName())
    (interface<-prName),
  rList : distinct DS! Treference
--For each RequestPoint a Treference is created and values are initialized
  foreach(rName in p.getRequestPointName())
    (name<-rName+'Reference', interface<-rName, bind<- 'set', unbind<- 'unset')
}
```

Figure 6. ATL Transformation rule for SoaML participant
The area marked with 4 in Figure 7 consists of a set of tabs. The tab shown in Figure 7 is the Properties tab. This tab allows the user to see or update information of the diagram. The user selects certain elements and the Properties tab shows the information of that element. In this case, the user is pointing to the canvas which is the model.

The user can also navigate in the different files of the project using the Navigator (see Figure 7, marked with 5). There is a file called default.soaml. This file contains the Ecore view of the SoaML model being graphically designed (see Figure 8). In this view, the user can validate the model or choose to validate specific concepts. If the model has errors a box pops describing the problems encountered (see Figure 9).
When the model is validated and there are no errors, the user can run the ATL model transformation to transform the SoaML model into Declarative Services Model. Once the DS Ecore model is created, the user can view it in the XML view (see Figure 10). This can be used in order to publish, register and access services in OSGi. The user has to implement its bundles in OSGi (the implementation of the components).

![Declarative Services generated from SoaML model](image)

**Figure 10.** Declarative Services generated from SoaML model

By using the OSGi services command from the OSGi console (see Figure 11), we can observe that the ProcessVoiceCall service has been successfully accessed by the MobileDevice Component.

![ProcessVoiceCall service has been accessed by the MobileDevice](image)

**Figure 11.** ProcessVoiceCall service has been accessed by the MobileDevice

### 5. Discussion

There have been a few tools which provide support for SoaML. A list can be found in [18]. These tools provide modeling support using the SoaML UML profile. These tools have implemented the SoaML UML profiles and customized their existing UML editors. The essential difference between these tools and the tool we have implemented is that our tool is based on the SoaML metamodel. The UML profile is a lightweight extension mechanism and is appropriate when the objective is to model using an already existing UML editor. However, the use of stereotypes and tagged values preserve the UML semantics and do not create new languages. The tool presented in this paper gives the option to users to make use of the SoaML metamodel. In this way, SoaML can be extended in order to be used in different domains. Users can reuse our SoaML implementations and extend them to create their own domain specific languages. In addition, code generation using the SoaML metamodel can also be facilitated.

The current version of the tool has several limitations which have to be improved. In the following, we discuss them:

- Some of the concepts are not graphically shown. For example, the realization relationships between types and instances or the parameters of operations. Currently, these are indicated by introducing the needed information as fields in the Properties view.
- The validation of the models is only performed using the Ecore view. The user has to navigate from the diagram view to the Ecore view and use the validation option. This has to be improved by including a validation option in the graphical diagram view.
- The shapes of some of the elements have to be improved. For example, the ServiceContract shape should be an oval shape instead of a rectangular one.
- The icons appearing on the Palette have to be improved. It can be noticed that some Palette elements have the star icon which does not match with the element shape. These have to be implemented in the later versions.
- The graphical modeling plug-in should allow the user to include other UML diagrams which are needed for modeling the behavior of a SOA. For example, including activity diagrams or class diagrams. This will also allow us to generate improved OSGi code.
- A menu bar has to be included from the graphical editor plug-in in order to validate and generate the DS model. This will improve the usability of the tool since currently the user has to navigate to the ATL configuration to generate the DS models.
- The current version should improve the validation facilities. Currently, the tool does not allow specific actions if they are not allowed but some of them are only notified to the user, once he/she chooses the validation option. We would like to add guideline facilities during modeling.

### 6. Related Work

There has been a lot of work done on supporting SOA with Model Driven Development (MDD). IBM provides a tool for supporting the SOMA methodology using MDD [2]. The tool allows business architects to capture information using tabular spreadsheets similar to excel sheets and collaboration diagrams (similar to service contracts in SoaML). The information is then used to build IT SOA systems. The SoaML tool that we have implemented can be used in SOMA in order to model SOAs.

Other approaches have defined UML profiles for SOA such as in [8], [7] and [5]. Some of them demonstrate how their profiles can be mapped into the Web Service Description Language (WSDL) such as [7] and [5]. However, they do not provide any editors for modeling or code generation facilities. Currently, the SoaML UML profile has been implemented and being integrated with other metamodels in the SHAPE project [14]. One of the objectives of the project is to generate semantic web services code from SoaML.

An approach which has provided MDD support for generating OSGi code is the one presented by Cetina et al. in [3]. They
provide a tool which is based on a set of metamodels which are UML like and define their own metamodel for services. Their approach is focused for pervasive systems.

7. Conclusions and Further Work
This paper presents a tool for modeling SOA using SoaML and partially generating DS XML to provide SoaML based MDA support. The presented tool has been developed using Eclipse facilities such as EMF, GMF and ADT to build a SoaML editor and an ATL configurator for generating DS models. This research prototype tool has certain limitations, which have been discussed in section 5. We plan to improve the SoaML modelling editor by improving the user interface and validation facilities.

We are also going to improve the SoaML model transformations to generate improved DS models by mapping service contracts. We also intend to enhance the tool for generating OSGi bundles using platform specific models and MOFscript. The Declarative Services XML is generated by transforming Ecore models into XML using EMF’s facility. However, when the required code is another kind of text as Java, MOFscript has to be used.

One possible extension of the presented tool can be to provide tooling support for Ambient-SoaML [1], which extends the SoaML metamodel with new capabilities for mobile systems.

8. ACKNOWLEDGMENTS
This work was supported by Science Foundation Ireland grant 03/CE2/I303_1 to Lero - the Irish Software Engineering Research Centre. The students of Amrita School of Engineering acknowledge the EURECA project (www.mrtc.mdh.se/eureka) funded by the Erasmus Mundus External Cooperation Window (EMECW) of the European Commission.

9. REFERENCES