Abstract—Service Oriented Architecture (SOA) is a promising approach for designing and developing mobile applications. However, SOA concepts need to be combined with mobile principles to fully accommodate their requirements. Previously, a metamodel called Ambient-SoaML has been defined, which combines the Service Oriented Architecture Modeling Language (SoaML) and Mobile Ambients concepts inspired from Ambient Calculus. In this paper, a modeling tool that supports designing service oriented architecture of mobile systems based on Ambient-SoaML is presented. The tool allows users to graphically design Ambient-SoaML architectures and ensures that they are correctly built. The tool is an eclipse plug-in and has been implemented following Model Driven Architecture (MDA) principles and associated technologies. A running example is used to illustrate the features of the tool.

Keywords— Service Oriented Architecture Modeling Language, Model Driven Architecture, Mobile Ambients, Platform Independent Model (PIM), mobile systems.

I. INTRODUCTION

Mobile applications are characterized with the need for continuos change. When locations of devices, or people change, services that were offered at their previous locations cannot be accessed anymore and alternative services have to be found at the new location. Thus, understanding and designing software architecture with these characteristics will help the development process. However, traditional software architecture cannot cope with many of the characteristics of mobile applications.

A promising architectural style for designing mobile applications is the Service Oriented Architecture (SOA) [6]. Several of its principles such as dynamic composition, loose coupling, and well defined interfaces are suitable to define the change and heterogeneity encountered in mobile applications. However, to properly define mobile characteristics SOA concepts have to be combined with mobile ones.

An approach which provides mobility primitives is called Ambient Calculus [3]. Ambient Calculus introduces a concept called ambient which represents a boundary where computation happens. An ambient can be a mobile phone, a network, a laptop, a building or a room. Ambients can have mobility capabilities which allow them to enter sibling ambients and exit parent ones. Previously, a metamodel called Ambient-SoaML has been defined to provide the design of service oriented architecture of mobile systems in a technology independent way [1]. Ambient-SoaML combines service oriented architecture concepts based on the Service Oriented Architecture Modeling Language (SoaML) [7] and extends it with concepts of Ambient Calculus.

In this paper, a modeling tool that supports the graphical design of Ambient-SoaML service oriented architecture of mobile systems is presented. Users of the tool can design different ambients and configure the service oriented architecture based on them. Users can also ensure that they are correctly building the Ambient-SoaML models. The tool is an eclipse plug-in and has been implemented following Model Driven Architecture (MDA) principles and associated technologies [8]. Before implementing the tool, the metamodel presented in [1] has also been refined. To illustrate the features of the tool, a running example is used.

This paper is structured as follows: Section 2 gives an overview of the SoaML and Ambient-SoaML metamodel concepts. Section 3 presents how the Ambient-SoaML tool has been developed following MDA standards. Section 4 presents the usage of the tool. Section 5 discusses related work. Finally Section 6 highlights conclusions and our further work.

II. AMBIENT-SOAML

This section presents an overview of Ambient-SoaML. Ambient-SoaML [1] is an extension of SoaML [7] with new capabilities for mobile and distributed systems. Mobile systems need to dynamically adapt to new environments or locations as users and mobile devices frequently move across boundaries. Thus, an important principle of designing service-based systems is that boundaries are explicit. This fact becomes more important when a mobile device moves, because it is crossing a boundary and it is entering another. When it is in the new boundary, its required services have to be provided.

The Ambient-SoaML metamodel is presented in Figure 1. The concepts that are in white are SoaML ones, and the ones in colour are the ones enriched by Ambient-SoaML. In the following we briefly describe several SoaML concepts [7].

- ServiceInterfaces describe the operations used between a service provider and a service consumer from the perspective of the provider. 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.

- **ServiceArchitectures** are the high level view of a SOA that defines how a set of participants work together for providing and using services.
- **ServiceChannels** provide a communication path between consumer Requests (ports) and provider services (ports).

In the following, several Ambient-SoaML concepts are described:

- **Ambient**
  An ambient is a bounded place which defines locations where different participants can be located and where they can request and provide services. An ambient can locate other participants and ambients, and the boundary presented by an ambient can be physical, geographical or other kinds of boundary.

  An ambient is introduced as a special kind of participant because it can provide and request mobility services. If it requests mobility services, then it is a mobile ambient. Ambients can be connected by service channels. In addition, the participants located in an ambient can also request services from its parent ambient or its siblings (a participant or an ambient). This means that there can be Service Channels between a parent ambient and its children or among its children. However, there cannot be Service Channels between participants of different ambients.

- **MobilityService**
  MobilityService inherits from the SoaML concept Service. They are the mobility capabilities provided by ambients. The MobilityService is separated into two kinds: one is Entering Service and the other is Exiting Service. The Entering Service allows an ambient to accept a sibling ambient into it, to become its child. The Exiting Service allows a child ambient of an ambient to become its sibling.

- **MobilityRequest**
  MobilityRequest inherits from the SoaML concept Request. It allows an ambient to request mobility capabilities. Ambients that have MobilityRequests are mobile ambients. They can be connected to a MobilityService of another ambient through a Service Channel. There are two kinds of Mobility Requests. The Entering Request allows an ambient to request to its sibling that it needs to become its child, and the Exiting Request allows a child ambient to request its parent ambient to become its sibling.

- **MobilityServiceInterface**
  MobilityServiceInterface inherits from the SoaML concept ServiceInterface. A MobilityServiceInterface describes the operations needed for providing the mobility capabilities. A MobilityServiceInterface can be used as the type of a MobilityService or MobilityRequest, and it can also implement the realization and usage of one or more Interfaces.

- **MobilityServiceContract**
  MobilityServiceContract inherits from the SoaML concept ServiceContract. It defines the specification of the agreement between a consumer ambient and a provider ambient for providing a mobility service. The specification includes the conditions, terms, interfaces and choreographies etc. that the interacting ambients must follow. A MobilityServiceContract must include at least two ambients as its parts, and the role of a part can be designated by a MobilityService or a MobilityRequest.

- **Intermediary Concepts**
  Ambients don’t have to only provide and consume mobility services. For example, the MobilityServiceContract is an agreement between a consumer ambient and a provider ambient of a mobility service. However, there may be agreements among ambients for security purposes. Thus, the AmbientServiceContract exists in order to facilitate the extension of these kinds of service contracts among ambients that are not only for mobility purposes. AmbientServiceContract is an abstract class, which means that it cannot be instantiated. When there is a new kind of service contract among ambients, it can be specified by defining a new class which inherits from the AmbientServiceContract class.

In the same way, some other abstract classes are defined to deal with the different concerns of ambients. These include: AmbientServiceInterface, AmbientService and AmbientRequest.

III. A MODELING TOOL FOR AMBIENT-SOAML

We have developed the Ambient-SoaML editor as an Eclipse plug-in for graphically modeling Ambient-SoaML models (Ambient-SoaML Editor). Two main steps have been followed in order to create the modeling tool for Ambient-SoaML. In the following, we briefly explain them:

First, we have used the Eclipse Modeling Framework (EMF) [4] for defining the Ambient-SoaML metamodel. The Ambient-SoaML metamodel has been defined as an instance of Ecore. An Ambient-SoaML Ecore model was created by adding each of the Ambient-SoaML concepts and relating them to each other. In addition, a set of validation rules have
been implemented in order to guarantee that users create correct Ambient-SoaML models.

Second, we used the Graphical Modeling Framework (GMF) [5] to associate a graphical notation for the Ambient-SoaML Ecore model and finally create a graphical editor.

In the next sections, we explain how we have followed these steps in detail.

A. Ambient-SoaML metamodel as ECore model

The EMF[4] permits metamodels to be defined using a language called Ecore and generates a genmodel which is a set of Java classes representing each metamodel concept. The genmodel is used for creating editors, creating code, or managing models created from the metamodels. In this section, we explain how we have defined our Ambient-SoaML metamodel.

Ambient-SoaML is an extension of SoaML i.e., the new concepts of Ambient-SoaML are derived from SoaML. Therefore, we loaded the SoaML Ecore model previously defined in [2]. Based on the Ambient-SoaML specification presented in section II, for each concept of Ambient-SoaML, the Ecore classes were created making use of the Ecore editor.

Figure 2 presents a screenshot of the Ambient-SoaML Ecore model. For example, it can be shown that Ambient extends the MParticipant class. The Ambient has five EReferences i.e., associations with five other Ecore classes: EnteringRequest, ExitingRequest, EnteringService, ExitingService and Ambient.

From the Ambient-SoaML Ecore model, the EMF Generator model is created. EMF generator models are used to generate the Java reflective code that allows the creation and persistency of models, and also permits the creation of an eclipse plug-in for editing models in a tree like hierarchy.

B. Validation Rules

In addition to creating the Ambient-SoaML Ecore model, we included validation rules. These rules will enforce users to create correct models when designing ambient systems.

The tool provides two mechanisms for enforcing the design of correct models: hard and soft constraints. Hard constraints are the ones that a user should never do. Soft ones are the ones that allow the user to design incorrect models temporarily because the models are incomplete and should be corrected when they are completed.

We have implemented 17 validation rules. In the following, we present several of them.

1) Ambients: An Ambient should contain at least one Participant. This means that ambients should be created to locate other participants which can also be ambients.

2) MobilityService: This concept has three validation rules associated to it. These are:
   a) A MobilityService should be typed with an Interface or MobilityServiceInterface.
   b) If the type of a MobilityService is an Interface, this Interface should be provided by the MobilityService.
   c) If the type of a MobilityService is a MobilityServiceInterface, the Interfaces "used" by the MobilityServiceInterface should be required by the MobilityService and the Interfaces 'realized' by the MobilityServiceInterface should be provided by the MobilityService.

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. The EValidator methods are called while creating a model from the Ambient-SoaML metamodel and when validating that model. 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 partial code of the implementation of the validateContents operation is shown in Figure 3. This is the validateContents operation shown in Figure 2. It can be...
shown that if the ambient does not contain any participants a message will appear to the user.

C. Ambient-SoaML graphical editor

As mentioned in section III.A, EMF creates an editor that allows users to design Ambient-SoaML models represented in a tree hierarchy. Thus, visual characteristics that show how elements are related cannot be noticed in a friendly way and there is no graphical notation. In addition, there are no modeling facilities such as drag and drop. Therefore, the GMF [5] has been used to develop a graphical modeling Editor for Ambient-SoaML.

The mapping definition connects the Ambient-SoaML Ecore model, the graphical definition and the tooling definition (the palette). It defines which concepts can be created directly on the canvas, and which ones as children or links.

IV. USAGE OF THE TOOL

This section explains how a user can graphically design Ambient-SoaML architectural models using the Ambient-SoaML Editor. First, we will give an overview of the tool, and later an example of the mobile telephony is modeled in Ambient-SoaML by using the tool.

A. Overview of Ambient-SoaML Modeling Tool

Figure 5 shows the user interface of the Ambient-SoaML graphical modeling editor. The editor has a Package Explorer (1) which shows the structure of the projects within the Eclipse workspace. Users can create new projects, and edit the different files of the project. When a new Ambient-SoaML model is created, the tool creates two files: one with .soaml_diagram extension and another with .soaml extension. The .soaml file contains the Ecore view of the Ambient-SoaML model being designed. In this view, the user can design Ambient-SoaML models and validate them in a tree structure (and tabular way). Moreover, most users prefer to use the .soaml_diagram to graphically design Ambient-SoaML since it is more user friendly. Independently, of which is being used, the tool will always maintain the models of these two files consistent.

The area marked with (2) is the palette that contains Ambient-SoaML concepts with visual icons and different tool groups.
The third area (3) is the canvas where elements dragged from the palette can be dropped. Not all the elements can be dropped into the canvas. For example in the metamodel, the EnteringRequest is part of the Ambient concept, therefore, it can only be dropped into an Ambient figure. When pointing over the canvas area or the figure of a concept, a box (4) is shown with the concepts that can be added to the canvas or the current concept. Users can simply click the icons in the box to add new elements to the current model (4).

The area marked as 5 contains several useful tabs. The Properties tab shows the detailed information and properties of the current selected element in the model being edited. In Figure 5 the element selected is the MobileDeviceAmb and the properties view shows its properties. The Problems tab shows the errors and warnings about the validity of the model being edited.

The tool provides and guides the user for hard constraints by not allowing him/her to perform them. Figure 6 shows how the tool does not allow the user to make specific actions. For example in (a), the user is trying to drag and drop an EnteringRequest in the canvas. An EnteringRequest can only be dropped in an ambient. In (b), the user is trying to connect a MobilityServiceInterface (MConsumer) to a part of a ServiceContract (Protocol). A MobilityServiceInterface can only be part of a MobilityServiceContract.

To support soft constraints, the tool provides a Validate option in the Diagram menu (see Figure 7). If the model violates the specific validation rule, the message that describes the problems (errors or warnings) will be displayed in the Problems view (marked with 4). In addition, a symbol will appear in the figure of the model, which shows the model has problems. If the problem is an error, the symbol will be a red cross (marked with 3) and if the problem is a warning, the symbol will be a yellow attention symbol (marked with 2).

The validation of models can also be performed using the Ecore modeling editor (by using the .soaml file). Users can validate the whole model or choose a specific element by clicking the Validate option. If the model is correct, a dialog pops up indicating that the validation was completed successfully. If the model violates a specific validation rule, a dialog pops up describing the problems (errors or warnings) encountered. Detailed information about the validation is displayed in the dialog including, the name of the element, its type and the error message.

B. Modeling an Example

In this section, an example of mobile telephony network is used to illustrate the capabilities of the tool and the expressiveness of Ambient-SoaML. The example is based on the GSM service.

Figure 6. Forbidding the creation of incorrect models

Figure 7. Validating an Ambient-SoaML model in the graphical editor
The GSM network is composed of mobile networks and mobile devices. A mobile network offers two services to mobile devices: a voice telephony service and the Short Message Service (SMS). A mobile device can be a mobile phone or PDA that has a Subscriber Identity Module (SIM) card. A mobile device can enter to visiting mobile networks, if the mobile network accepts the mobile device. The process of accepting a mobile device is as follows: the mobile device requests to enter a mobile network, then the mobile networks checks if the SIM card is issued for that network. If the SIM card is issued for a different network, then the network registers the device as a visitor, and offers it the voice telephony service. However, the SMS is offered by the original network.

In the following, we describe how the tool is used to design the Ambient-SoaML architectural elements of the example. It is important to note that the tool does not restrict any order for defining the elements.

1) Mobility Service Interfaces:

Based on the scenario, the entering services depend on three different roles which are three Mobility Service Interfaces (see Figure 8): EnteringRequester, EnteringProvider and NotificationReceiver. The EnteringRequester is the role used by a mobile ambient to request the entering into another. It realizes the interface AcceptingToEnter, and uses the interface EnteringInterface. The EnteringProvider is the role used by a destination ambient that allows other ambients to enter into it. It realizes the Interface EnteringInterface, and uses the Interface AcceptingToEnter. The EnteringInterface contains an operation called requestEntering, and the AcceptingToEnter interface contains an operation called acceptanceToEnter. The drag and drop features of the modeling tool are used to define the interfaces, MobilityServices, the uses, and relationships. However, the parameters of the operations cannot be defined using the current version of the graphical tool. Instead, the Ecore view (.soaml) model has to be used to define them.

2) Mobility Service Contracts:

A Mobility Service Contract called EnteringToMobileNetwork defines the agreement between the parties that involve how a mobile device moves into a new mobile network (see Figure 9). There are three parts: the enteringRequester, the enteringProvider and NotificationReceiver. These are instances of the Mobility Service Interfaces defined in Figure 8. The properties view of each part is used to indicate their types e.g., the type of enteringRequester is EnteringRequester. It is a composite mobile service contract because its parts use a Mobility Service Contract instance called mobilityNotificationToOrigin, which is needed for notifying the original mobile network when a mobile device has moved. The MobilityNotificationToOrigin is a simple mobility contract (see Figure 10).

Another MobilityServiceContract called DistributedSms was created (see Figure 11). It contains two roles called smsReceptAmb and smsOriginAmb and a service contract called sms service. The DistributedSms was defined to show that when a mobile device enters a new ambient, the sms service becomes distributed. It is important to notice that a Mobility Service Contract can be composed of Service Contracts that are not related to the mobility concern such as the sms service. This allows modifying the behavior, without having to modify the SMS functionality.

3) Ambients:

In the example, we have defined two kinds of ambients: a Mobile Device Ambient (MobileDeviceAmb) and a Mobile Network Ambient (MobileNetwork). The boundary of the MobileDeviceAmb is determined by the physical mobile device; and for the MobileNetwork, the boundary is determined by the radio network of the Base Transceiver Stations.

The functional parts of the mobile device have been modeled as a participant (see Figure 12). This participant is located in the MobileDeviceAmb ambient. The MobileDeviceAmb has an EnteringRequest port which is needed because the MobileDeviceAmb is mobile and needs
to request from its parent ambients mobility services. In addition, it also contains a port called FunctionalityPort which redirects calls to the MobileDevice participant.

A service architecture GSMMobileNetwork with ambients was created for this example. A possible configuration of the architecture is shown in Figure 14. Two different instances of MobileNetwork ambients (vodafoneIreland and movistarSpain) are included in the architecture, as well as an instance of MobileDeviceAmb called MP1. In this configuration, MP1 that originally is located in movistarSpain has entered vodafoneIreland, a visiting ambient. It can be noticed that MP1 is connected to smsPMov of movistarSpain through the ports of vodafoneIreland and movistarSpain. On the other hand, MP1 is connected to voiceTelephonyProcessor because both MP1 and voiceTelephonyProcessor are in the same ambient.

At the end of the design, the user has to validate that Ambient-SoaML architecture has been well defined and that all the constraints have been satisfied. This includes ensuring that the configuration satisfies the requirements specified in the ambient hierarchies. In this example, the architecture is correctly defined and the tool indicates that the validation is successfully completed.

The example presented in this section has been previously modeled using a UML 2.0 editor as described in [1]. However, the use of the Ambient-SoaML modeling tool allows the users to have the needed primitives for ambient awareness and mobility explicitly. This does not only increase the user friendliness but also ensures that the models are correctly built before they can be manipulated for other purposes such as code generation or model transformations.

Figure 14. GSMMobileNetwork Architecture when MP1 has entered VodafoneIreland

V. RELATED WORK

There has been many approaches for designing service oriented architecture in a technology independent way and following a MDA approach [2] [9][11]. The work presented in this paper, makes use of MDA and focuses on how mobile systems can be designed in a technology independent way using service oriented architecture.

Ambient-PRISMA approach [10] provides a similar modeling tool to Ambient-SoaML which is based on model driven engineering. However, the main difference between Ambient-SoaML tool and Ambient-PRISMA is that Ambient-PRISMA defines architectures based on aspect-oriented and component based software development and not on the service oriented architectural style.
Tergujeff et al [12] discuss the extension of SOA for mobile devices from a technological point of view, specifically using Web Services. They mention that mobile devices can be consumers and providers of web services. In a similar way in Ambient-SoaML, mobile devices (or any kind of ambient) are supported as participants that can be a consumer or provider. They also mention that mobile device vendors take different approaches for implanting Web Services. This can indicate that using a model driven approach that is technology independent can be useful.

Loke et al. [14] use service domains (similar to our concept of ambient) for referring to geographical boundaries that are associated with a set of services. Each service domain stores mapping tables which stores ambient services (services in a boundary) and operators are used for computing services of a user profile. The implementation techniques used in Loke et al. could be used in Ambient-SoaML in future works for calculating services of user profiles. However, our approach is more abstract.

Choudhury et al. [13] discuss challenges for deploying service oriented architecture on Mobile Adhoc Networks. They describe that mobile nodes can provide services to other nodes. This is represented in Ambient-SoaML because different ambient can be mobile or not, but they participate in the provision and consumption of services. They emphasize that many of issues present require an attention to modeling the architecture. We believe that the Ambient-SoaML modeling tool focuses on this direction.

VI. CONCLUSIONS AND FURTHER WORK

This paper presents a tool for modeling SOA of mobile systems using Ambient-SoaML. The presented tool has been developed using Eclipse and MDA facilities such as EMF and GMF. A running example has been used for illustrating the features of the tool.

In the near future, we are working on improving the graphical facilities and support the scalability of Ambient-SoaML models. In the current version of the tool, only structural characteristics are provided. We also would like to include a visual representation for the mobility dynamic behavior, e.g., to show how a mobile ambient instance moves from one ambient to another and show the different ambient configurations that are generated as a result of this mobility. This will aid users in understanding how their designs will be affected as ambients move. In addition, our plan is to empirically evaluate the usefulness of the tool.

Our further work includes supporting Ambient-SoaML with a complete Model Driven Architecture approach. We are currently working on implementing a lightweight middleware for mobile devices. This middleware will map Ambient-SoaML to a technological specific platform. Our objective is to extend the Ambient-SoaML modeling tool with automatic code generation capabilities. We will have to implement mappings between the Ambient-SoaML metamodel and the middleware using Query/View/Transformation (QVT) in order to transform from AmbientSoaML platform independent models (PIMs) into platform specific models (PSMs) [8], and use MOF script for generating executable code.

ACKNOWLEDGMENT

Fei Chen was a master student at UL and Carlos Solis was affiliated to Lero at the time of developing this work. This work was supported, in part, by Science Foundation Ireland grant 10/CE/11855 to Lero - the Irish Software Engineering Research Centre (www.lero.ie).

REFERENCES