Extending Enterprise Architecture Modeling Languages:
Application to Telecommunications Service Creation

Vanea Chiprianov, Yvon Kermarrec
Institut Telecom, Telecom Bretagne, Université européenne de Bretagne, UMR CNRS 3192
Lab-STICC, Technopole Brest Iroise, CS 83818 29238, Brest Cedex 3, France
vanea.chiprianov@telecom-bretagne.eu

Siegfried Rouvrais
Institut Telecom, Telecom Bretagne, Université européenne de Bretagne, Technopole Brest Iroise, CS 83818 29238, Brest Cedex 3, France
siegfried.rouvrais@telecom-bretagne.eu

ABSTRACT
Enterprise Engineering offers a global view on multiple concerns such as processes, stakeholders, supporting technology. This global view is sustained by Enterprise Architecture frameworks, languages, tools and standards. The current effort has been focused on general purpose Enterprise Architecture frameworks, modeling languages and tools, which allow describing a wide range of domains. While they are expressive enough at the business layer, at the technical layer, where more detail is needed to describe a domain specific system, such general purpose Enterprise Architecture Modeling Languages sometime lack the semantic strength required. The concepts present in the language are too abstract, they need refinement and specification. To provide the necessary specific semantic strength, this paper proposes an approach to extend Enterprise Architecture Modeling Languages with domain specificity. The proposed approach is a model-driven one, allowing a high degree of automation in the building of tools for the language extension. To better show its benefits, the approach is applied on the domain of Telecommunications, for defining an Enterprise Architecture Modeling Language extension for service creation. The so defined language and its associated tools are illustrated on an IP Multimedia Subsystem conferencing service example.

Categories and Subject Descriptors
D.2.2 [Software Engineering]: Design Tools and Techniques.

General Terms
Design, Languages, Management.

Keywords
service and system modelling; technological infrastructure modelling; enterprise architecture frameworks; languages for enterprise engineering; model driven engineering.

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, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

SAC’12 March 25-29, 2012, Riva del Garda, Italy.
Copyright 2011 ACM 978-1-4503-0857-1/12/03 ...$10.00.

1. INTRODUCTION
Enterprises have been considered as complex adaptive systems [12]. They are diverse, made up of multiple interconnected elements and have the capacity to change and learn from experience. To document, understand and master this complexity, architectures are an important means [22]. An Enterprise Architecture (EA) approach consists of a set of models describing the structure and functions of an enterprise. Models are organized into layers, from more generic (e.g. objectives) to more specific (e.g. technology), and into views conforming to viewpoints (e.g., End User, Developer). Layer and viewpoint are used here as defined by ArchiMate [15]. An EA approach is beneficial for: tackling system complexity, more agile business alignment with technology platforms, interoperability and integration of constituting systems, common understanding across the enterprise.

However, there are several EA methods and frameworks (e.g., TOGAF, RM-ODP, Zachman Framework). When applying an EA approach to a real case, only one framework has to be chosen. An EA framework can describe a wide range of domains. It can describe integrated models of the enterprise, which can be understood by all stakeholders. While this is useful at the business layer, at the technical layer, where more detail is needed to describe a system, such a framework lacks the semantic strength required. Lacking semantic strength means that the concepts introduced by the framework are too abstract and they need refinement and specification. To cover this lack, an EA development method [15] proposes to use an EA framework at the business layer, while at more detailed layers (e.g. technology) introduce domain specificity.

This paper introduces domain specificity in EA modeling languages through language extension (cf. Sec. 2). In the meta-modeling approach (cf. Sec. 3), extension consists in extending the meta-models that define the language. The approach is instantiated for Telecommunications, presented in Sec. 4 to tackle the service creation problem (cf. Sec. 4). The obtained Enterprise Architecture Modeling Language extension and associated tools are used to model an IP Multimedia Subsystem conference example (cf. Sec. 4). Due to space considerations, the reader is directed to the section on related work from [8].

2. EXTENDING ENTREPRISE ARCHITEC-
TURE MODELING LANGUAGES
EA frameworks define processes, guides, methods. They are theoretical in nature, and to apply them concretely,
modeling languages are needed. There are many modeling languages, both EA specific (e.g., ArchiMate, IDEF, UML), called Enterprise Architecture Modeling Languages (EAMLs), and general purpose (e.g. UML). UML is a family of graphical modeling languages allowing structural and behavioral specification. While modeling an EA using it is possible, UML is not purposely conceived to offer the EA constructs, which would have to be re-modeled in UML. That is why the EAMLs form the focus of this paper.

In accordance with the EA modeling method proposed by [15], domain specificity (for an argumentation of the need of domain specificity in EA cf. e.g. [8]) can be introduced through Domain Specific Modeling Languages. A Domain Specific Language (DSL) is "a language that offers, through appropriate notations and abstractions, expressive power focused on, and usually restricted to, a particular problem domain" [15]. A Modeling Language is, "a graphical language for visualizing, specifying, constructing, and documenting the artifacts of a software-intensive system" [3]. A Domain Specific Modeling Language (DSML) is therefore taken in this paper to be a graphical language that offers, through appropriate notations and abstractions, expressive power focused on a particular problem domain, to visualize, specify, construct and document the artifacts of a software-intensive system. DSMLs allow experts to express, validate, modify solutions and achieve tasks specific to their domain. A DSML requires less cognitive [13] effort from experts than a more general purpose language, as it offers a higher closeness of mapping between the problem world and the program world. Thus quality, productivity, reliability, maintainability, re-usability can be enhanced.

However, introducing domain specificity raises the question of integration between the DSML and the EAML. Also, defining a new language from scratch is very costly in terms of associated tools (e.g. compiler). To answer this concerns, the DSML is defined in this paper as a profile of the EAML. A profile [1] is a generic extension mechanism for customizing reference languages with constructs that are specific to particular domains, platforms. Although profiles are usually associated with UML, they can be generalized to other languages as well. The term profile is used in this paper in its general meaning, and not in its UML-bound sense.

Profiles allow refining the reference language in a strictly additive manner, so that they can’t contradict standard semantics. This means that profile tools, defined as extensions to existing tools for the reference language, only have to process the additions. The initial cost of DSML tool implementation as profile is lower than that for a standalone DSML. Also, interoperability between several DSMLs, defined as profiles, is facilitated by the reference language. The constructs common between the reference language and each DSML respectively, and the connections between the constructs from the reference language, facilitate defining connections (interoperability) between profiles.

3. LANGUAGE EXTENSION WITH THE META-MODELING APPROACH

The manner in which language extension is done depends on the manner in which the initial language is defined. There are several methods for defining a language. Compiler theory uses: abstract and concrete syntaxes, and semantics. It provides (meta-)tools (e.g., lex, yacc) that generate language-specific tools (e.g., lexical, syntactical parsers). However, it deals with textual languages, whereas EA deals with graphical models, so graphical languages.

In the Meta-modeling approach for graphical modeling languages definition [9], the abstract and concrete syntaxes from compiler theory are described using meta-models (cf. Fig. 1 without the big red rectangle). One way of describing operational semantics is by generating code from the new language to existing (programming) languages, and so producing an executable form of the model. The Meta-modeling approach also provides (meta-)tools (e.g., Eclipse Modeling Framework (EMF), Xpand) that generate language-specific tools (e.g., graphical editor, code generation). The general accepted meaning of a meta-tool is a tool that allows specification and generation of another tool.

Meta-tools reduce significantly the time, effort and cost of constructing language-specific tools. When language definition evolves, it impacts the meta-models describing the syntax and the semantics. Due to meta-tools, language-specific tools can be re-generated to include the new language features, with limited modifications, fast and inexpensive.

The Meta-modeling extension approach is synthesized in Fig. 1 with the big red rectangle. It consists in extending the initial meta-models describing the abstract and concrete syntaxes, and extending the semantics.

4. USING EAMLS TO ANSWER TELECOM SERVICE PROVIDERS REQUIREMENTS

From the 90’s, the telecom service creation industry has undergone radical change [4]. Traditionally, services were offered on equipment from switch manufacturers. Nowadays, services have become more software based. Service creation is a complex activity not only because of the technical activities involved, but also of the number of actors and the difference in each actor’s perspective and objectives. To simplify this situation, roles have been identified to allow separation of the different perspectives from the organizations concerned [14]. End User, Service Subscriber, Service Provider, Network Provider, Manufacturer, Tool Vendor. The focus of this paper is on Service Providers.

Service Provider is the organization that provides value-added services. The Service Provider is primarily interested in considerations related to the End User (e.g. promotion of services in the marketplace) and deployment (e.g. validation of services against requirements). The Service Provider has searched independence from Manufacturer, to create its own services and be more flexible in answering the End User’s expectations, to whom it is closer.
In such a complex stakeholder ecosystem, to assist service creation, a software Service Creation Environment (SCE) is required. As this paper focuses on the role of Service Provider, their requirements for SCEs are:

1. **Req 1 An overall model**: A graphical model describing the service creation process from a market standpoint should include all business, economic activities, network, technical activities, data: Application, ApplicationFunction, ApplicationInteraction, and data: DataObject.

2. **Req 2 Domain specificity**: Tools specialized for the task and domain are essential to decrease the concept-to-market time, like those for specifying service functionality.

3. **Req 3 Rapid prototyping**: The service potential market success must be assessed rapidly.

4. **Req 4 Early validation/simulation**: Early validation, incompatibility and error detection, are needed to guarantee that the operation of the new service will not produce unwanted side effects within a distributed telecommunications context.

To answer the **Req 1 An overall model**, this paper relies on EA frameworks and modeling languages. However, these are general purpose, without any specificity to the tasks and domain of telecom. To answer the **Req 2 Domain specificity**, a DSL, dedicated to telecom specific tasks, is defined. To integrate these two approaches, the telecom DSMI is defined as a profile to an existing EAML. Among the many methods for defining a language, the one which seems to fulfill best the **Req 3 Rapid prototyping** of the Service Provider is the Meta-modeling approach, as it offers code generation and easy integration with existing domain specific tools.

To answer the **Req 4 Early validation/simulation**, a model transformation towards a network simulation tool is defined.

Among EA frameworks, the most promising to date for telecom is TOGAF. A mapping between TOGAF and Frameworx has identified important synergies between them. Frameworx is a telecom-specific framework developed by the TM Forum to organize, specify, design and develop new generation management systems. It provides a standard method, common terminology and harmonized framework for the telecom industry value chain. The Open Group Architecture Framework (TOGAF) provides methods for assisting in the acceptance, production, use and maintenance of an EA. At the core of TOGAF is the Architecture Development Method, consisting of eight phases and providing one of the most complete guiding step-by-step process for creating an EA. TOGAF is sometimes considered one of the strongest frameworks on the business and technical layers, providing much detail for them. As these layers are important for the Service Provider, TOGAF is retained as EA framework.

One EAML is ArchiMate. It shares important concepts with TOGAF, and thus the two are usable together. It models three phases of TOGAF Architecture Development Method into three layers: Business, Application and Technology. It regulates interoperability between them by defining possible dependencies. Due to its proximity with TOGAF, ArchiMate is retained as EAML. The meta-model for the Application layer is presented in Fig. 2 without the concepts in a red circle, which are specific to Telecom. There are structural concepts: ApplicationInterface, ApplicationComponent, ApplicationCollaboration, behavioral ones: ApplicationService, ApplicationFunction, ApplicationInteraction, and data: DataObject.

Figure 2: The ArchiMate telecom Application layer meta-model extension.

5. **EXTENDING EAMLS FOR TELECOM**

A modeling language needs its definition, but associated software tools also.

5.1 Language Definition

To define the Telecom-specific language extension, the abstract and concrete syntaxes, and the semantics are needed. To define the abstract and concrete syntaxes in the Meta-modeling approach, meta-models are used.

For the abstract syntax, the extension is treated separately for each layer. The extensions for the Business and Technology layers of the ArchiMate abstract syntax have been presented elsewhere (cf. [7] and respectively [8]). Following the same practical meta-model extension approach (presented in more detail in [7]) used for those two layers, the meta-model for the Application layer is extended as well. Three new concepts are added to the initial ArchiMate Application meta-model: ServiceFunctionalOperation, ServiceFunctionalComponent and ServiceElementaryFunction. These concepts have been found similar to the ones from which they inherit, but sufficiently dissimilar (i.e. specific to the Telecom domain) to introduce them as new ones.

For each concept and relation introduced in the meta-model, a graphical representation (belonging to the concrete syntax) is defined. An excerpt is presented in Fig. 3. In their definition, closeness to concepts from ArchiMate had been emphasized, so that concepts that inherited those from ArchiMate have inherited their representation as well, with a mark of distinction (e.g. an orange rectangle, cf. ServiceElementaryFunction).

To implement the semantics of the language extension, the choice of using template-based code generation (or model-to-text transformation) has been taken. This was due to the maturity of this approach, and the availability of powerful, mature, free tools. In this approach, the transformation of the input language is captured in rules called templates. The templates are defined using the Xpand template description language, and its editor and compiler, the free Eclipse plug-in OpenArchitectureWare. The static semantics is implemented, resulting in Java classes, with attributes, method signatures (cf. Lst. 1). For the behavioral semantics, method call is implemented, resulting in the possibility of translating sequence-like diagrams into code (cf. lines 15 and 18 from Lst. 1). The rules transform, for example, for the Application layer meta-model:

1. For static semantics:
   a. ApplicationInterface and ServiceElementaryFunction into Java interfaces;
Figure 3: Excerpt from the concrete syntax.

(b) ApplicationService into Java abstract method;
(c) ApplicationComponent and ServiceFunctionalComponent into Java classes;
(d) The composition relation (between ApplicationInterface and ApplicationComponent) into the Java implements relation (between an interface and a class);
(e) The aggregation relation (between ApplicationColaboration and ApplicationComponent) into Java attributes;
(f) ApplicationFunction into a Java method;
(g) DataObject into Java parameter for method signature.

2. For behavioral semantics:
(a) The triggered by relation (between ApplicationFunction and itself) into Java method call.

Obviously, the code generation encompasses not only the Telecom extension, but also the initial ArchiMate language.

5.2 Language Tools

Language-specific tools are usually the editor (for modeling languages a graphical one) and the compiler describing the operational semantics of the language. Meta-tools targeted to the generation of graphical editors used in this project are the Eclipse Graphical Editing Framework (GEF), the Eclipse Modeling Framework (EMF). An existing open source ArchiMate graphical editor, Archi [5], developed as an Eclipse plug-in, has been selected and reused, expanded (more details about the plug-in extension process in [6]) with the Java code needed to describe the concrete syntax of the Telecom extension. The resulted editor presents the classical divisions of an Eclipse-based editor. Archi has a model navigator and an outline of the graphical model. Its central window presents views (defined as tabs) of the graphical model. The palette was extended with telecom specific concepts and relations (cf. an excerpt in Fig. 3), from which the designer can select, drag and drop the desired ones.

For compiler implementation, a meta-tool, OpenArchitectureWare (OAW), has been used. If software compilers, that translate from a high level of abstraction language into machine language, are considered tools, any other means of implementing language translations can be considered tools as well. Consequently, model transformations may be seen as tools. It follows also that language translation implemented as template-based code generation (a model-to-text transformation) may be seen as a tool. So, the templates for translating the Telecommunications extension of ArchiMate into Java, written in Xpand and run with OpenArchitectureWare, describe the configuration of a compiler. The generated skeletons can be further detailed/implemented using Java libraries that abstract Telecom specific protocols, like JAIN [12].

Req 4 Early validation/simulation is highly important for Service Providers. There are many mature and extensively used telecom simulation tools. For example, OPNET is such a network simulation and dimensioning tool, with capabilities of simulating the behavior of a model at different levels of detail. Packet exchange can be visualized, errors in the signaling logic can be detected. To take advantage of these capabilities, a model-to-model transformation between the Telecom ArchiMate extension and the XML internal format of OPNET is defined. The transformation is described in Xpand. So, the editor, the compiler and the transformation towards the simulator (which is an external proprietary software) are all integrated in the Eclipse platform. An overview of the language tools is presented in Fig. 4.

6. MODELING A TELECOM SERVICE WITH AN EXTENDED EAML

To exemplify the power and benefits of the language extension and its associated software tools, they are applied to the description of a telecom service.

A description of a conference service may use two roles: Moderator and Participant (cf. Fig. 5). The concepts of Business: 'Role', 'Event', 'Process', Actor, 'Service' and relations of 'triggering', 'assignment', 'and-junction' used in Fig. 5 are the ones defined by ArchiMate [18]. After taking the decision (cf. Fig. 5a)), the Moderator creates the conference and sends the connection details to the other participants (cf. Fig. 5b)). A Participant, at the appropriate time, enters the conference using the details received from the Moderator, starts the conference, chats, speaks, and/or sends video at the same time, as many times and in any
order (s)he may want, until exiting the conference (cf. Fig. 5c)). Then the Moderator terminates the conference (cf. Fig. 5d)).

Because of space considerations, the focus of this example for the Application layer is limited to the phase of entering the conference. This is the most difficult/important phase, when the signaling is established (cf. Fig. 6). The concepts of Application: 'ServiceFunctionalOperation', 'ServiceFunctionalComponent', and relations of 'triggering', 'assignment' used in Fig. 6 are those defined by ArchiMate and the telecom meta-model extension (cf. Fig. 2 and Fig. 3). The Client part of the conference system launches the console and tries to join the Participant to the conference. If after three tries the connection is refused by the Conference system, an error message is displayed. If everything is alright, the Participant subscribes and joins the conference.

The technical layer description of the service is much bigger, that is why only excerpts of the models are presented here. This is where a clear distinction between static entities and behavior is made. The entities are specific to the IP Multimedia Subsystem (IMS) (cf. Fig. 7). The behavior uses messages specific to IMS as well (cf. Fig. 8, after the signaling provided in [4], represented here, close to a UML sequence diagram).

The model-to-model transformation towards the OPNET internal format is then applied. Fig. 9 shows for example the results of the UDP traffic simulation for one of the IMS nodes, the P-CSCF11. The model-to-code transformation...
public class Clientpart1ofconferencesystem {
    private String name = "Clientpart1ofconferencesystem";
    private String id = "78d717c8";
    public Clientpart1ofconferencesystem ()
    {
        this.name = newName;
    }
    public String getName()
    {
        return name;
    }
    public void setName(String newName)
    {
        this.name = newName;
    }
    public String getId()
    {
        return id;
    }
    public void joinconference()
    {
        public void launchconferenceconsole()
        {
            ConferenceSystem conferencesystem = new ConferenceSystem();
            conferencesystem.checkresponse();
        }
        public void displayerror()
        {
            ConferenceSystem conferencesystem = new ConferenceSystem();
            conferencesystem.checkresponse();
        }
        public void subscribe(conference)()
        {
            this.joinconference();
        }
    }
}

Listing 1: Excerpt of generated Java code for the conference service at Application layer (cf. Fig. 6)

towards Java is then applied, obtaining code (cf. Lst. [1].
In conclusion, this example shows the use of the proposed language and its associated tools from an initial high-level business description to a low-level Java code.

7. CONCLUSIONS AND FUTURE WORK

This paper has identified a real need for extending Enterprise Architecture Modeling Languages (EAML) with domain specificity. It proposed as solution the extension of EAMLs with Domain Specific Modeling Languages (DSML) as profiles.

The language definition solution adopted here was a model driven one. It offers two types of advantages, to the modeler using the EAML: overall model, domain specificity, rapid prototyping, and to the vendor building the language-specific tools: reduced time, effort and cost, easy evolution synchronized with that of the language.

In the future, to further help the EAML modelers, collaborative methods will be integrated, especially for capturing the rationale behind design decisions.

8. REFERENCES