MMSA: Metamodel Multimedia Software Architecture

Makhlouf Derdour, Philippe Roose, Marc Dalmau, Nacira Ghoualmi-Zine, Adel Alti

Hindawi Publishing Corporation
Advances in Multimedia
Volume 2010, Article ID 386035, 17 pages
doi:10.1155/2010/386035

Research Article

MMSA: Metamodel Multimedia Software Architecture

Makhlouf Derdour,[1] Philippe Roose,[2] Marc Dalmau,[2] Nacéra Ghoualmi Zine,[1] and Adel Alti[3]

[1] *Computing Department, University of Annaba, 19000 Annaba, Algeria*
[2] *Computing Department, LIUPPA—IUT of Bayonne, Bayonne, 64600 Anglet, France*
[3] *Computing Department, University of Setif, 19000 Setif, Algeria*

Correspondence should be addressed to Makhlouf Derdour, m.derdour@yahoo.fr

Received 1 February 2010; Revised 28 May 2010; Accepted 2 July 2010

Academic Editor: Martin Reisslein

Interoperability explains how two or more systems or components exchange and process information. The heterogeneity communication mechanisms of the components (GPRS, WIFI, Bluetooth, ZigBee, etc.), transmission speed, as well as the variety of the media (sound, video, text, and image) they manage have a strong influence on the interoperability. That requires the management of the adaptation to an abstract level in order to avoid ad hoc nonreusable, and/or generalizable solutions. In this paper we propose a metamodel for architectures with heterogeneous multimedia components. It enables the description of the software architectures as a collection of components manipulating various types and formats of data, and interacting between them via specific adaptation connectors.

1. Introduction

With recent progress in software and material technologies, multimedia systems become increasingly sophisticated and complex. Today, companies require multimedia applications that combine a variety of data sources, such as audio, video, text and image, and of the multiparty interactive communications. The multimedia communication needs services able to face with heterogeneity on several levels: the context, the access devices, the communication network, the user, and so forth. It is necessary to integrate solutions to deal with the data heterogeneity problem, and to answer to the changes of the context caused by the user, the application, the network, or the access device. The future multimedia ubiquitous systems must have adaptation capabilities, and be able to modify the system configuration and/or the multimedia contents at any time. This requires taking into account the data flows and the components interactions in the early development phases of application.

Among the software architecture for pervasive applications, it exists component-based architectures that allow the reasoning about complex software systems at an abstract level, that is, ignoring the details of design and of implantation. Architecture is an abstract and modular description of a system. At this level, the architecture is perceived as a collection of components *(in the sense of software entities)*, a collection of connectors *(to describe interactions between components)*, and of configurations *(assemblies of components and connectors)*. The separation of concerns *(functional/non-functional)* can deal with the components as well as the assemblies themselves. They cover the structural and the dynamic aspects of applications. The adaptation is one of the concerns that we consider non-functional and serves to ensure the interoperability of heterogeneous components.

Multimedia technology is increasingly being used to create reliable and effective communication environments. However, the design of multimedia applications is currently driven more by intuition than by empirically or theoretically derived design guidelines. In a multimedia application, the software architecture is defined as a set of components manipulating various multimedia data types with specific constraints that we must take into consideration at the architectural design. For instance, the problem of heterogeneity if based or the exchange of multimedia data flows. In this paper, we propose Meta-model Multimedia Software Architecture *(MMSA)*, an approach for multimedia software, which enables the description of software architectures expressing a multimedia software system as a collection of

components which handle various types and formats of multimedia data, and interacts with them via adaptation connectors.

The remainder of this article is organized as follows. After exposing our objectives and our motivations, in Section 3 we present the model MMSA. Section 4 presents models of multimedia data and adaptation techniques. Section 5 presents the adaptation in MMSA and its architectural concepts. Section 7 summarizes the related work. Finally, Section 8 concludes this article and presents some perspectives.

2. Motivations

Our main motivation is to propose a meta-model for maintaining data consistency in configurations constituted of various components exchanging heterogeneous data. We propose new types of graphic interfaces and connectors with a richer semantic.

The use of these graphics interfaces allows the automatic detection of heterogeneity points between components, while the use of adaptation connectors allows the resolution of these heterogeneities. The systems are built by assembling *(functional)* components and *(non-functional)* connectors, where each element is correctly placed in the architecture configuration. In most of the ADL *(Architecture Description Language)* we find the following.

(i) The choice of the available connectors in the environment is limited to the primitive connectors, no compounds connectors.

(ii) The management of the non-functional concerns of the components is ensured after the definition of architecture and configuration of the components.

(iii) The management of assembly does not take into account the behavioral heterogeneity *(semantic)* of the components of software architecture.

(iv) Few models are able to define new connectors with different treatments that ensure the non-functional concerns of the components *(security, communication, conversion, etc.)*.

(v) There is no direct and automatic correspondence between architectures *(models)* and the applications conceived following these architectures *(instances)*.

In order to solve these problems, we propose MMSA to describe multimedia components-based software architectures. Based on the definition of four types of interfaces according to existing data flow *(image, sound, text, and video)* and to strategies to make multimedia flows *(type, format, property)* to three levels, we propose a model to solve the problem of components data exchange heterogeneity. It is developed in order to reach the following objectives.

(i) Ensure a high level of abstraction for the connectors in order to make them more generic and more reusable, and therefore reconfigurable.

(ii) Take into account the semantics of communication links between components in order to detect points of heterogeneity and insert the adaptation connectors in those points.

(iii) Favor the maintenance and the management of the adaptation QoS and of the communication ensured by the connectors by providing the following possibilities: adding, suppression and substitution of adaptation services.

The contributions of this paper are different from the previous related works. Firstly, the paper gives the connector a central role in dynamic architectures (i.e., dynamic adaptation service for managing QoS). Secondly, it solves the problem of heterogeneity in the conceptual level. Finally, it allows taking into account the capabilities of hardware components, by moving the adaptations processes in other machines.

3. Data Multimedia and Adaptation Techniques

The multimedia environments are increasingly heterogeneous. The interoperability of component-based multimedia applications and automatic deployment of such components are very difficult. Indeed, the diversity of languages, protocols, platforms, and media *(images, text, sound, and video)* induces important incompatibility. Moreover, the instantiation and the configuration of multimedia applications guided by user's preferences, requirements of the context, and characteristics of multimedia components is not an easy task to achieve.

The development of multimedia applications requires two complementary models: a multimedia data flow model allowing the representation of various types of media exchanged between components and their relationships, and an architecture model based on the concepts of ADLs extended to multimedia and integrating adaptation connectors. The main idea of this proposal is to take into consideration the standard concepts of multimedia data as well as the nonfunctional concerns *(data adaptation, communication protocol, security, etc.)* of the components by connectors at the software architecture level. The objective is to propose a generic, clear, and complete description. In the following parts we present different concepts for multimedia represented by models. For each model we detail the relations between its concepts.

3.1. Data Flow Model. In pervasive environments (mostly heterogeneous and mobile), the devices can require for any contents type, going from textual contents to the complex and rich multimedia documents. Ensuring the delivery of the adapted data to each peripheral requires adaptation techniques which take into consideration the media and the flows structuring. Therefore, their modeling is necessary. It facilitates the adaptation work between media of the same type (e.g., image to image) or between different media types (e.g., text to sound).

The hierarchic structure of media is expressed in UML using a class diagram (cf. Figure 1). The media are classified in two categories: continuous media, such as video or sound, which are characterized by temporal dependencies, and

Table 1: Adaptations of media.

(a)

Category	Video	Sound
Transcoding	Format conversion	Format conversion
Transforming	(i) frame rate (ii) spatial resolution (iii) temporal resolution (iv) color depth	(i) change sampling
Transmoding	(i) video to image (ii) video to text (iii) video to audio	(i) audio to text

(b)

Category	Text	Image
Transcoding	Format conversion	Format conversion
Transforming	(i) font size (ii) police, color, etc.	(i) data size (ii) dimension (iii) color depth (iv) color to grayscale
Transmoding	(i) text to Audio (ii) text to Image	(i) image to Text

discrete media such as image or text. Each type of media has a set of encoding formats and some specific properties like the resolution (in the case of image or video), the frequency (in the case of the sound), and so forth. we distinguish three types of structural links between media: temporal (to describe the temporal dependences between units), logic (to describe the logical organization of a flow in hierarchy form of media), and spatial (to describe the disposition of the multimedia-flow elements).

Currently, the multimedia data flows must be executed on many platforms *(Smartphones, PDA, Laptop or Desktop PC, etc.)*. These various peripherals and uses require the adaptation of flows according to their execution context, which are sometimes unforeseeable at the time of preparation and design of data.

3.2. Adaptation of Data Flow. Each media can undergo three types of adaptation. The first one is known as the format conversion *(Transcoding)*. It allows conversion in the same type according to a different encoding format *(e.g., BMP to JPEG)*. The second one allows a handling of the media characteristics *(e.g., modification of image resolution)*. This type of adaptation *(Transforming)* depends on the media format, since each format authorizes the change of some characteristics in the form of parameters. The third and more complex transformation is called conversion of types *(Transmoding)*. It allows passing from a media type towards another *(e.g., text to sound for blind people)*. This conversion of the type can also act on media structures by removing the temporal dependences *(e.g., the video to the images)*. Each adaptation has an impact on the data quality. Thus, the conversion of an image from a JPEG format towards a GIF one implies a reduction in the number of colors to 256, the opposite implies the suppression of component "transparency," which according to the use context can be problematic, even crippling.

The adaptation is a process (cf. Table 1) allowing a modification the type of media (*transmoding*), the format of encoding (*transcoding,*) and/or the media content (*transforming*) in order to adapt it to the component recipient. The class diagram (Figure 2) shows the various classes of association allowing the passage of a media type to another, or of a media format to another format.

We classify media adaptation processes into three categories.

(i) *Transmoding* consists in changing the modality of a media. As an example, consider the transformation of a sequence of text to an image if the client terminal does not provide with the required police.

(ii) *Transcoding* means changing the encoding format of a given media. For example, video may be transcoded from the MOV video format to the AVI video format.

(iii) *Transforming* a given media does not change the modality neither the format. This process transforms the content by, for example, reducing the size.

The relation between association classes of transmoding with the association class of transcoding explains that the transcoding class can be called upon by the transmoding class to participate in achieving the task of the latter. Although the relation between transcoding class and the transforming is a relationship of dependence, this relationship explains that each format has a set of parameters to manage the various qualities of media. The transforming is a particular type of transcoding which keeps the same format of media with changes of characteristics *(e.g., conversion of a color JPEG picture to a black and white one)*.

4. The MMSA Meta-Model

The adaptation problem covers *(but not limited to)* the heterogeneity of content information. Many new features have been integrated with new advanced encoding techniques. It exists now content in the form of images, vector graphics, animations and videos, and so forth. Designers must respond to the problem of heterogeneity caused by the evolution of information content. They need an abstract level in order to offer generic and reusable solutions allowing a good architecture design of multimedia application.

MMSA meta-model describes the software architecture of the system as a collection of components interacting with connectors. Components and connectors have the same abstract level and are defined explicitly by the separation of their interfaces and their internal configurations (Figure 3).

Architecture Description Language (ADL) can be classified in three different categories [1]: ADL without connectors, ADL with a preset set of connectors, and ADL with explicit types of connectors. In the last case, the ADL provides connectors as first-order elements of the language such as: Wright [2, 3], ACME C2 [4], xADL [5], AADL [6], and so forth. All these languages seek to improve

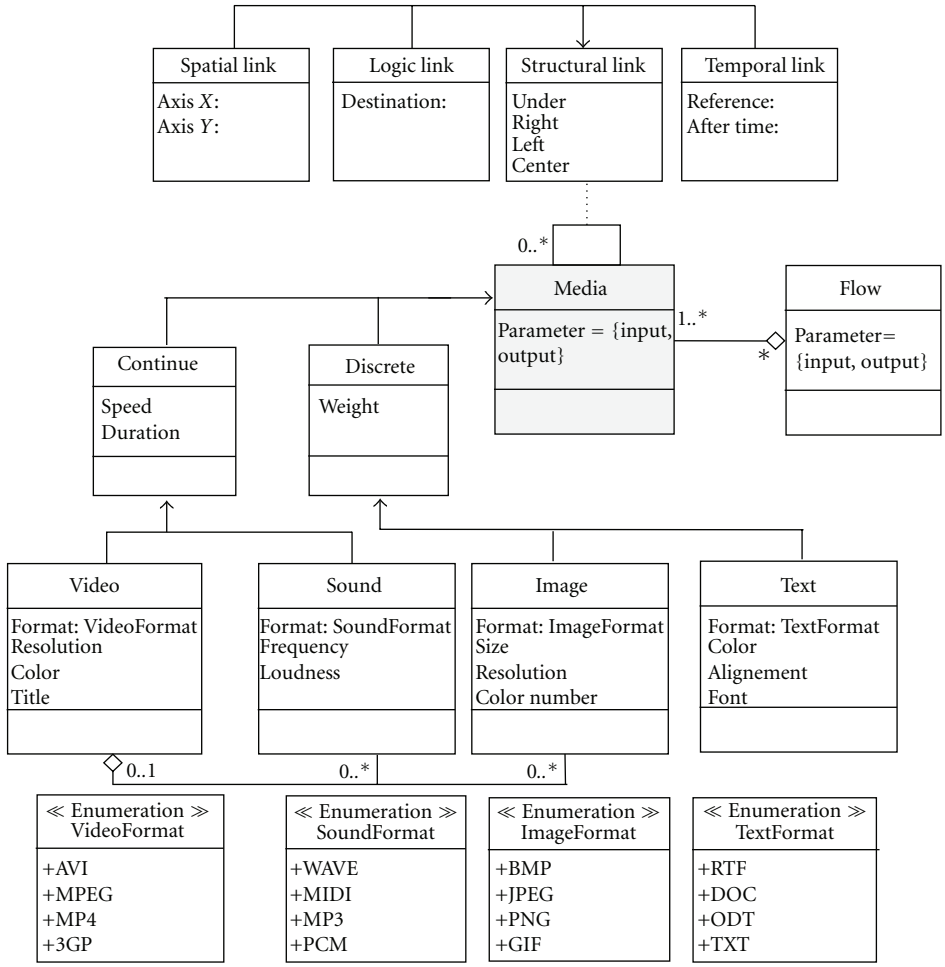

Figure 1: Multimedia flow model for MMSA.

the reusability of the components and the connectors by separating the calculation and the coordination. In our approach, we choose the explicit category of connector. Thus, in MMSA meta-model, we present a generic and explicit type of connector that the system can specialize according to the architecture and the components needs.

An MMSA component is a computation unit having a state. A component may have several implementations (business parts) (Figure 4). A component can be primitive or composite. Each component may have an interface with multiple ports and multiple multimedia services. The interface consists in a set of interactions points between the component and the external world that allow the invocation of the services. We distinguish between an "Output" interface exporting data of components, and the "Input" one importing data to components. Each interaction point of a component is called a port. Ports are named and typed. Each port can be used by one or more services.

Most of existing ADLs do not support multimedia ports; however describing architectures without multimedia typed ports may clutter the outcome design and makes it hard to understand its overall structure. So, we have typed each port with a type of media *(sound, image, video and text)*, so we have distinguished each MMSA port type. This distinction of the ports by data type can simulate the behavior of a component, in order to detect the heterogeneity points between components and to treat them at this level. This gives a better verification of the consistency and validity of the configurations of software architectures.

In MMSA, a connector is a configuration of three components *(communication, adaptation and QoS)* ensuring connection between the components. It ensures the nonfunctional concerns of components *(quality of service, data transformation, communication)*. This allows a possible change of the adaptation services during the execution of the application *(dynamic and real time adaptation)*, and preserves the abstract specification of the component.

A MMSA connector is defined by two interfaces "Input" and "Output" and a glue unit represented by three managers: communication, adaptation and QoS (cf. Figure 5). They manage the data transfer between components and allow adaptations to be made. A required/provided interface of connector is composed of a set of roles. Each role serves as a point through which the connector is connected to the component.

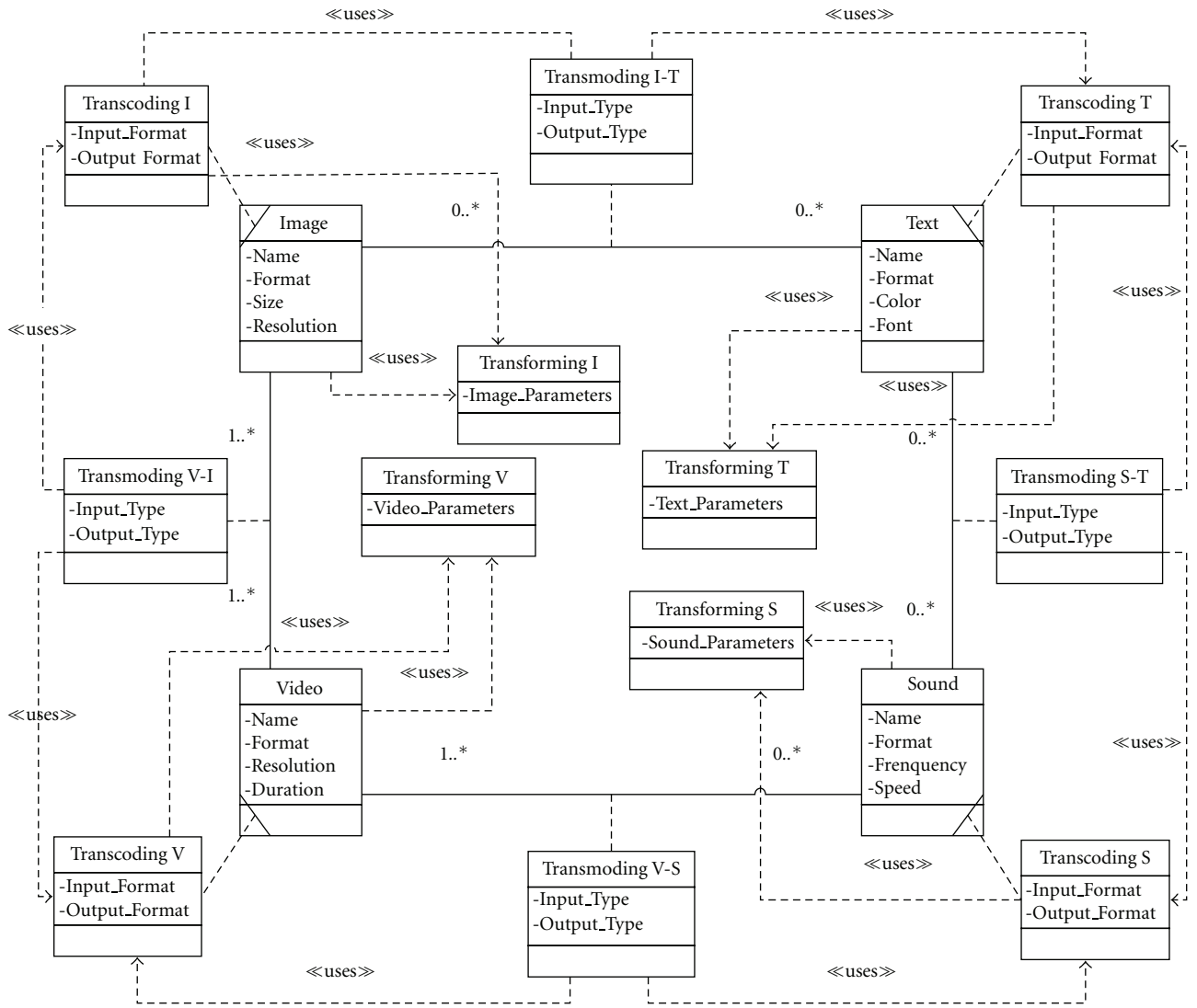

Figure 2: The transformation relationship between different media.

Examples 1 *and* 2. In the first example, the components exchange the same type and format of data (Figure 6). They need a communication connector, but not adaptation is why the adaptation manager and the QoS manager are deactivated *(gray)*. While the second example shows the possibility to connect two heterogeneous components *(one component provides a JPEG image, the other requires a PNG image)* (Figure 7). This requires an adaptation ensured by one or more connectors depending on the complexity of adaptation. In this example, it is ensured by two connectors *(JpegToBmp and BmpToPng)*.

To improve the specification of connection points, we have enriched the notion of role according to their data flows into a connector. We have also extended the glue by an adaptation manager which cooperates with a QoS manager to ensure the adaptation task. An adaptation manager is a set of adaptation services that cooperate to realize adaptations. Two types of adaptation can be realized in software architectures. The semantics adaptation (conversion of type) related to the constraints of the data handled by components and the technical adaptation (conversion of format and adjustment of media characteristics) related to the capacity of components (*memory, display, etc.*). The QoS manager controls the adaptation manager in order to change the parameters of adaptation services to provide the adequate quality to the correspondent at runtime. The QoS manager participates both in selecting parameters of technical adaptation services of data flows (*e.g., reduction of resolution, reducing the number of images per second*) and even the adaptation services of type or format at runtime (*e.g., choice of compression ratio in the transformation from BMP to JPEG*).

Configuration is a connected graph of components and connectors that describe architectural structure. This information is needed to determine whether appropriate components are connected, their interfaces mach, connectors

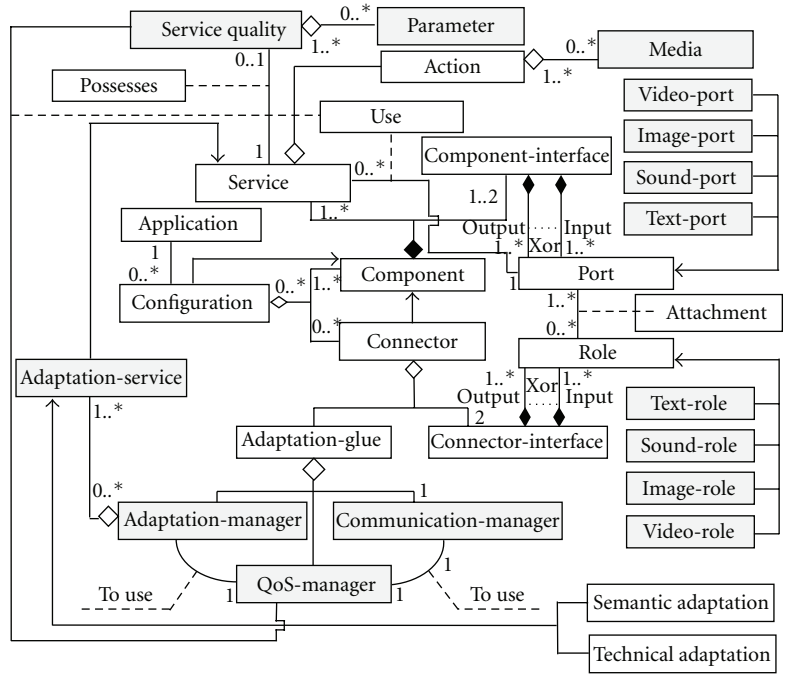

Figure 3: Class diagram of software architecture MMSA.

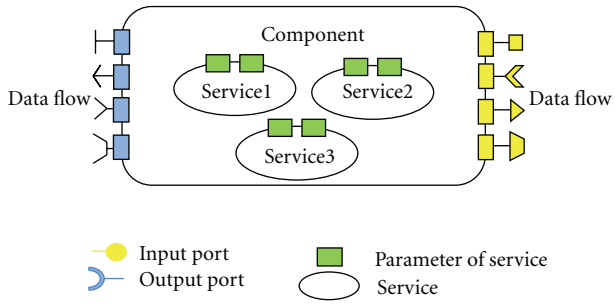

Figure 4: MMSA component.

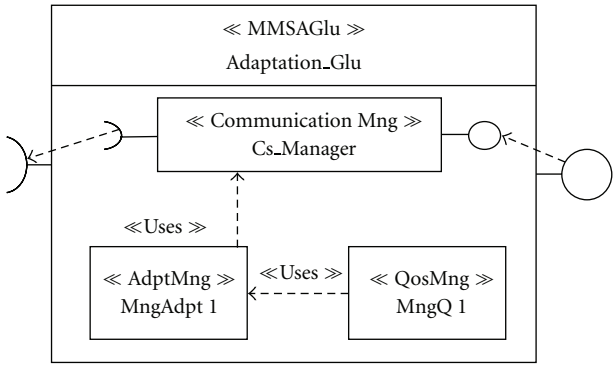

Figure 5: Internal structure glue.

enables proper communication, and their combined semantics results in desired behavior.

The key role of configurations in MMSA is to abstract the details of different components and connectors. They depict the system at a high level that can be potentially understood by actors with various levels of technical expertise and familiarity with the problem at hand.

An architecture configuration has a name and is defined by interfaces (ports and services), which are the visible parts of the configuration and support the interactions among configurations and between a configuration and its components.

In MMSA model, each component or connector is perceived and handled as a primitive element. But they can be primitive, or composite with a configuration which encapsulates all the internal elements of this composite. These configurations are first-class entities. A configuration may have ports similar to components ports, and each port is perceived like a bridge (binding) between the internal environment of the configuration and the external one. In MMSA, this binding is realized using connectors. Generally configurations can be hierarchical where the internal components and connectors can represent subconfigurations with their proper internal architectures.

In Figure 8, the configuration contains two components, which can be connected by one or more connectors, that is, a component needs at least one connector to communicate with another one. It can use several connectors depending on the complexity of the adaption task.

The configuration of Figure 8 contains a video acquisition component and another one providing video restitution. The need for adaptation is explained by one handicap (deaf) of the user of the restitution component. There, we need to transform sound to text and integrate it with the video, through three adaptation connectors *(Video to Sound + Image (A), Sound To Text (B) and Text + Image to Video (C))*.

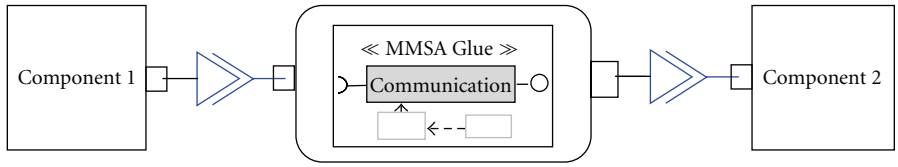

Figure 6: Communication connector.

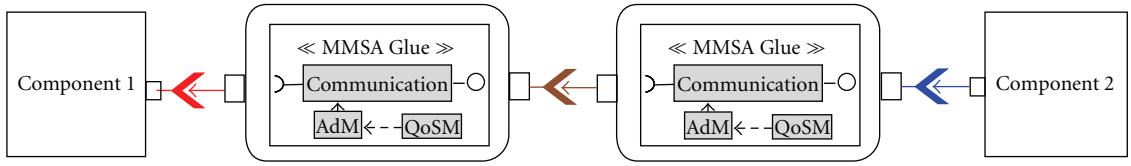

Figure 7: Two connectors of image transcoding JPEG to BMP to PNG.

Table 2: Port of multimedia interface.

Type	Input	Output	Format	
Text		DOC	Docx	ODT
Image		JPEG	BMP	PNG
Sound		WAVE	RM	MP3
Video		MPEG	AVI	MP4

5. The Adaptation in MMSA

During the creation process of architecture, in order to solve the heterogeneity problem of architectural elements *(component, connector, and configuration)*, the adaptation is made in three successive stages: (I) adaptation of the types, (II) adaptation of the formats, and (III) adaptation of the properties (Figure 9).

The data flow is a main constituent of the functional components, it is often specified as a constraint to associate with a functionality of communication involving several components.

The constraints of data flows such as the type, the format, and the media parameters must be specified at the architectural level. For that, we consider a new type of component intended to ensure a non-functional concern that of the adaptation, which one calls the adaptation connector related to the component which provides and/or requires the data multimedia. We propose a graphical notation of the ports of multimedia interfaces allowing to visually identify the heterogeneity points per media type and to highlight the need for the search of adaptation connectors (Table 2).

The detection of heterogeneity is done automatically by the checking of the constraints of forms and colors.

Adaptation of Type (Transmoding). The heterogeneity of components that manipulate the media of different types is detected by the use of different forms to represent the components ports *(step 1,* Figure 9*)*. Therefore, two components which have different ports *(e.g., text port and sound port)* (Figure 10) can be connected only by the use of one or several adaptation connectors of media type. This problem will be solved by the integration of the transmoding connectors at the architectural level.

Adaptation of Format (Transcoding). The heterogeneity of the components that manipulate the same type of media but with two different encoding format *(step 2,* Figure 9*)* can be detected by the presence of color differences between the formats of the same type. Therefore, two components which have different colors for the same port *(e.g., red port for MPEG video and blue port for 3GP video)* (Figure 11) can be connected only with the use of one or several connectors of format adaptation. This problem will be solved by the integration of the connectors of transcoding at the architectural level.

Adaptation of Media Properties (Transformation). The heterogeneity of components that manipulate the same media type with the same format *(step 3,* Figure 9*)* but with different properties *(e.g., resolution and color for image, sampling and speed for video, etc.)* cannot be expressed visually in our architecture, due to the parameters that depend on the media and on the adaptation service *(parameters of the service)*. Therefore, two components which have the same color for the same port *(e.g., image port)* can be connected with a simple communication connector, and during the execution, the adaptation manager and the QoS manager both manage together the adaptation if necessary. At this level the problem of heterogeneity is resolved at runtime, by the manipulation of the parameters of the adaptation service; if this service is configurable, regarding the parameters of flow.

The adaptation service is configured (Figure 12), in order to allow an adaptation in various situations; it is applied in several contexts, for example, with image resolution adaptation.

6. The Architectural Concepts of MMSA

It is largely accepted that the component can be accessed only via well-defined interfaces [7]. Interfaces link the components with the environment. The component-based

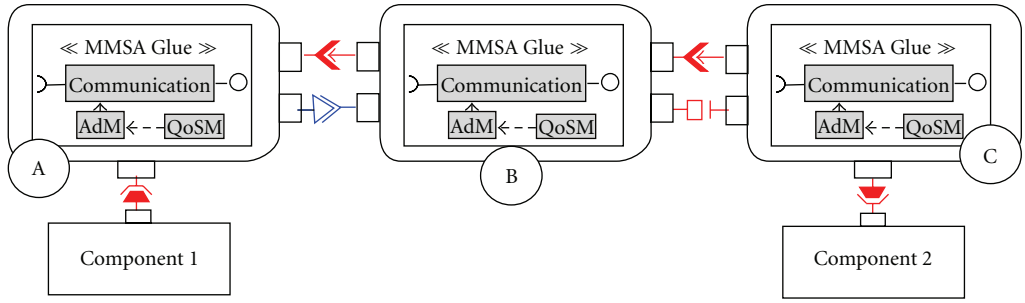

Figure 8: A configuration with multiple connections working in parallel and in sequence.

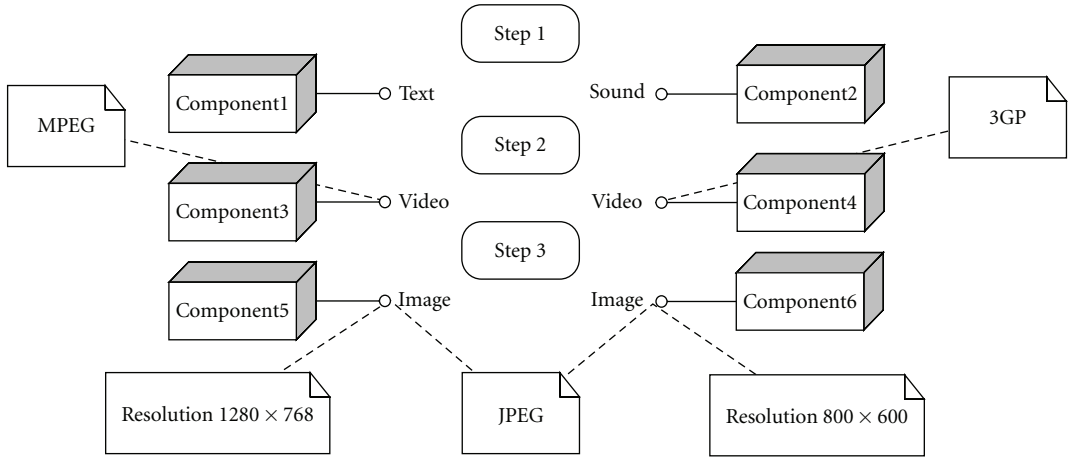

Figure 9: Heterogeneity between components.

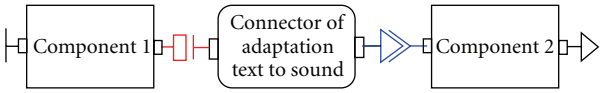

Figure 10: Transmoding connector of text toward sound.

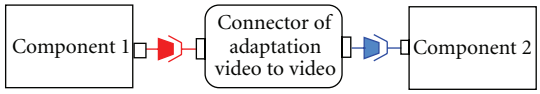

Figure 11: Transcoding connector of MPEG to 3GP.

languages offer different concepts to describe elements interfaces, such as services, ports, interfaces, protocols, and so forth. sometimes with different meanings. For example, in Fractal [8] or Enterprise JavaBeans [9], the concept of port and interface are mixed, so we only speak of interfaces. In UML components diagram [10] the two concepts of ports and interface exist, as in ArchJava [11], where the interface is called port of interface. That is why we have chosen to explain clearly the choices we have made for MMSA. In MMSA, a component provides or requires services from the ports described in the interface provided/required. Thus, MMSA offers typing ports to differentiate them by type of media manipulated *(text, sound, video, and image)*.

6.1. MMSA Component. The components are atomic elements from which an MMSA application is created. Like atoms, components MMSA behave in a coherent way, and they can be assembled in different configurations. MMSA understanding begins with understanding its basic components of application.

A component is an instance of an application that has been properly configured. The implementation is the code that envisages indeed the functions of the component, as a Java class or a BPEL process. MMSA components provide functionalities called services (Figure 13). Basically, a service is a subprogram defined in a program, as a method in object-oriented model.

6.1.1. Interfaces. Generally, the interfaces are a support of description of component to specify how they can be assembled or used within architecture. The interfaces are located both at a local level *(associated with a port)* and at a global level *(associated with a component)*. The interfaces of MMSA components are seen as the connection points of components and a support of services invocations. The concept of port is used to represent the exchange of data via component interfaces.

6.1.2. Ports. "A component is a static abstraction with plug-in" [12]. The ports represent these plug-in which are

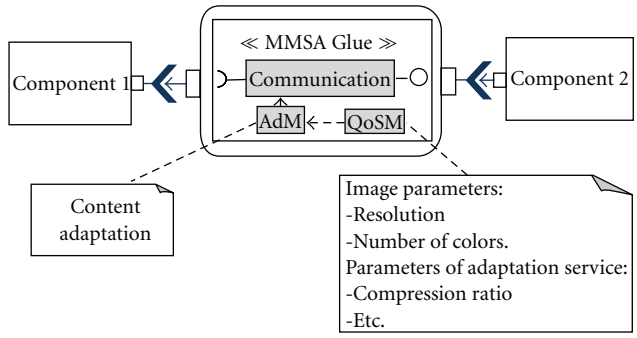

Figure 12: Adaptation connector of image content.

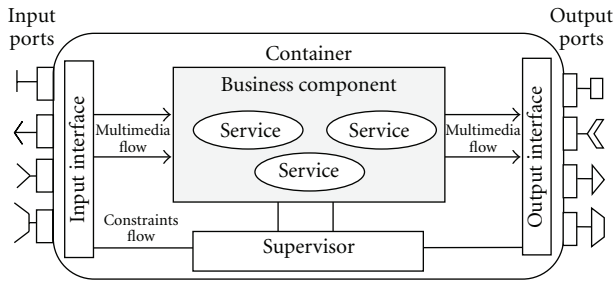

Figure 13: Multimedia component model.

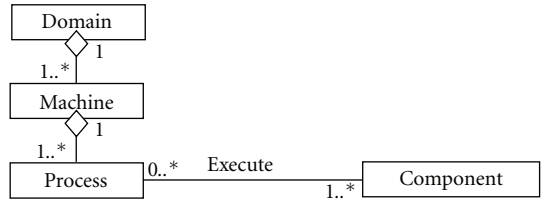

Figure 14: Modeling of domain concept.

the points of components interaction. This means that everything goes through these ports, like the invocation of services, for example. The port is presented in almost all models of components but with different semantic. In component models where the ports are supported, they are unidirectional or bidirectional. For unidirectional ports like ComponentJ [13] or Fractal [8], a component provides or requires all services via its ports. In ArchJava [11] or UML 2.0 [10], the ports are bidirectional and a component requires and provides services through the same port. In MMSA the ports are unidirectional, because a port can provide/require data via/from the connectors. The latter can apply adaptations to the data, and generally the adaptation services are not bidirectional *(e.g., the adaptation service of text to sound is not the same service to adapt sound to text)*. The definition of specific ports, each one oriented to support a specific data's flow will produce more organized architecture specifications where each data flow is considered in an independent manner.

6.1.3. Service. A service is ensured by the component. It has a set of parameters that allows controlling their outputs and set parameters of call. All these parameters describe the service interface. It communicates with the outside via the ports provided/required of a component.

We can say that a service is a function defined inside a component and offered to components. The service parameters and arguments passing when invoking service raises many questions: What is a parameter? Does one really need parameters? What is the difference between argument and parameter?

In MMSA, the arguments are the necessary elements to the execution of service *(the variable, term, or expression to which a service operates)*, while the parameters are the control elements of QoS *(e.g., a service that allows the transcoding of an image BMP into JPEG, this service receives in argument the path of BMP image, and parameter like the compression ratio)*.

6.1.4. Domain. Domains are an important concept; it defines the provision and distribution of components on different machines. A domain can contain one or more composite, of which each one has components implemented in one or more processes running on one or more machines [14].

The concept of domain such as presented in SCA [15] is used in MMSA. This concept allows taking into account constraints on the execution environment, in order to provide a good service to the machine running the component *(e.g., a component displaying video needs to know the physical characteristics of the host about which it will run to adapt the resolution or speed)*.

Figure 15 show a domain with three machines and eight components. At the top, on the left, we find three components that run in one process, on the right one finds two components that run in two different processes but on the same machine. Below, we find three components in two processes running on the same machine.

A domain is composed of several machines, each machine is responsible for execution of several processes and each process can contain one or more components (Figure 14).

The concept of domain brought much for MMSA, especially for the choice of connectors and adaptation services that allow the consideration of environmental constraints when designing the application architecture.

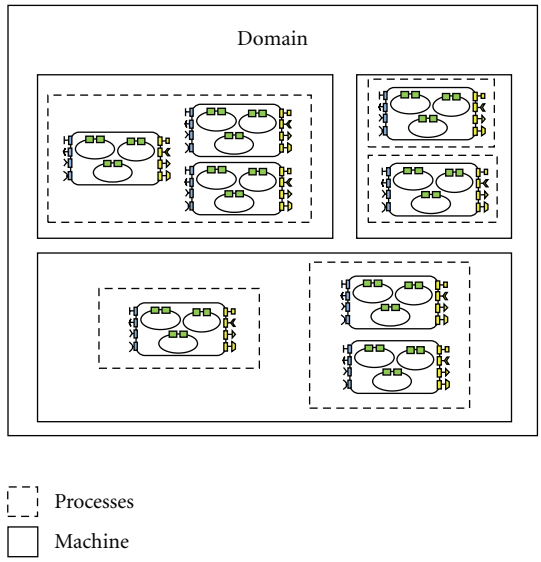

Figure 15: Example of the domain notion.

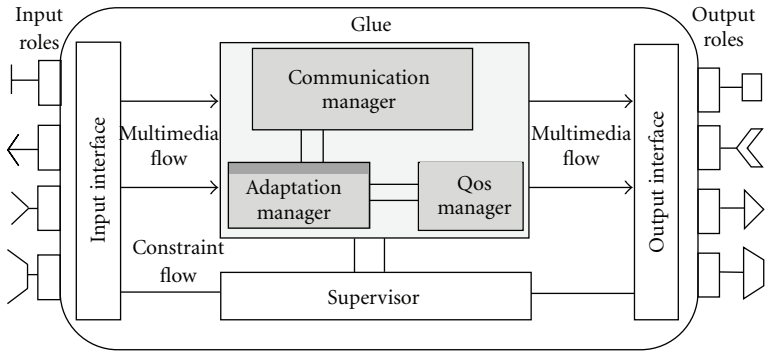

Figure 16: Model of multimedia connector.

6.2. MMSA Connector. Compared with those of the languages of description of architectures [16, 17], the connectors that we propose can be simple or composite and can ensure services. These connectors do not only ensure the communications links but also the adaptation of the data exchanged *(functional part of connectors)* between components.

The connector constitutes the entity of communication and adaptation in our approach (Figure 16), that is, it is able to transfer the multimedia data between the various components while ensuring the adaptation of the latter.

Allowing heterogeneous components to interact with each other is a significant task. The adaptation is considered as a nonfunctional concern of component. This task must be ensured by another element. The connector provides the nonfunctional concerns (communication, adaptation, security, etc.) which the component needs. The role of an adaptation connector is to receive the data, to adapt them according to the QoS manager directives, and to forward them the following component or to connector.

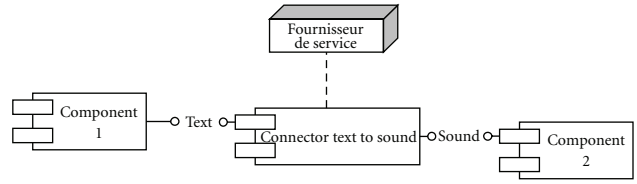

Figure 17: Relation connecteur-fournisseur.

6.2.1. Adaptation Service. The connector ensures the communication between two components (even heterogeneous) from the services provided by component or service providers that provide services according to the quality required by the QoS manager (Figure 17).

The adaptation services take part in realization of adaptation of exchanged data by components. The representation of a component by a set of services enables the use of its services to the adaptation task (Figure 18), the mechanism

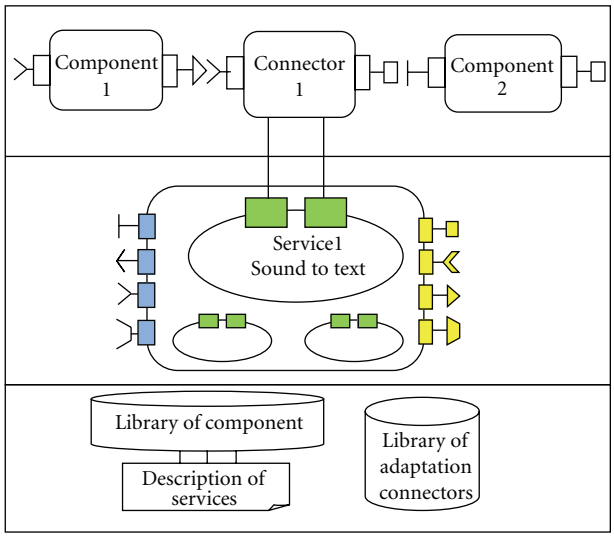

Figure 18: Use services of components in the adaptation task.

of such a use is the same of Web Service use, by considering components as service providers.

Two mechanisms can be exploited here: the composition of services defined by Kmelia [18] and the concept of library components defined in Fractal [19].

(i) The composition defines a hierarchical relationship (inclusion) which allows defining new services from existing services. The availability of mechanisms for service composition facilitates the definition of new abstractions of services without necessarily passing by the introduction of new components.For this composition of services, the concept of inclusion "Include" is defined by the use case diagram that can be used.

(ii) Fractal proposes the concepts of component library for developing components-based applications such as the Dream library [20], which is a library of components dedicated to the construction of message-oriented middleware dynamically configurable more or less complex. The same concept can be used by MMSA to propose libraries of adaptation connectors and adaptation services provided by the components (cf. Figure 18).

6.2.2. Shared Component. A shared component is a component that is included in several composites. Paradoxically, the shared components are useful to preserve encapsulation [19]. This concept allows the MMSA to share the same adaptation connectors to solve the problems of heterogeneity between components.

Unlike the components that are instantiated on demand and can have several instances, a service is single. But it has the advantage that it is connected to other components and services through standards of connection. These standards ensure decoupling, that is, the reduction of the dependences, these standards is XML documents as in web services.

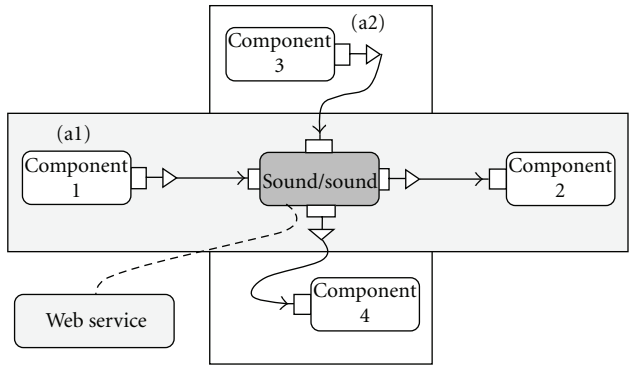

Figure 19: Example of a shared connector.

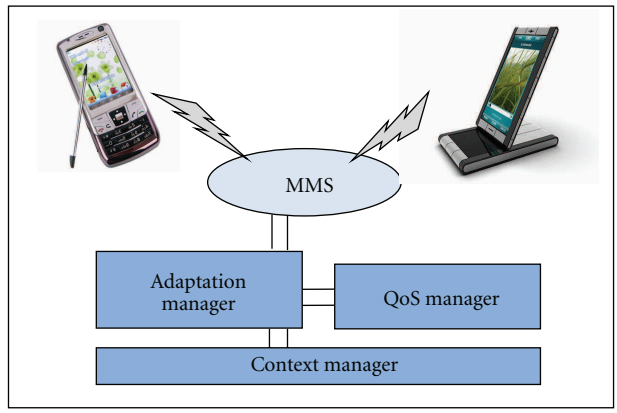

Figure 20: Design adapted MMS.

As shown in Figure 19, the notion of shared connector is very interesting. Especially, if the adaptation service is a Web Service. The aim is to benefit from the mechanism of instantiation of components and interoperability of Web Services; this allows better use of adaptive connectors.

6.3. Configuration. An MMSA configuration is described in an associated file of composition, whose name ends by configuration. This file uses a format based on XML called *Configuration Description Language* (CDL) to describe the components of this configuration. For the three components of Figure 19(a1), the basic structure of its configuration CDL is shown in Algorithm 1.

The configuration, expressed in CDL defines how this element interacts with the outside world. MMSA component could be implemented using almost any technology. Whatever the technology used, each component is based on a set of abstractions, including services, service parameters, the flow of data, and attachments. A composite descriptor does not contain the descriptors of its subcomponents, but it refers. The configuration of components is a central mechanism that relies on different types of links between ports *(or interfaces for models without ports)*.

```
<Configuration name="Example"...>
   <component name="Component1">... </component>
   <component name="Component2">... </component>
   <connector name ="Connector1">... </connector>
   <webservice name = "WebService1"> . . . <webservice>
   <liaison name= "Liaison1", Comp1="Connector1", Comp2="WebService1"> . . . </ liaison >
   <attachment name="Attachment 1", Comp1="Component1", Comp2="Connector1"> . . . </attachment>
   <attachment name="Attachment 2", Comp1="Connector1", Comp2="Component2"> . . . </attachment>
</Configuration>
```

Algorithm 1

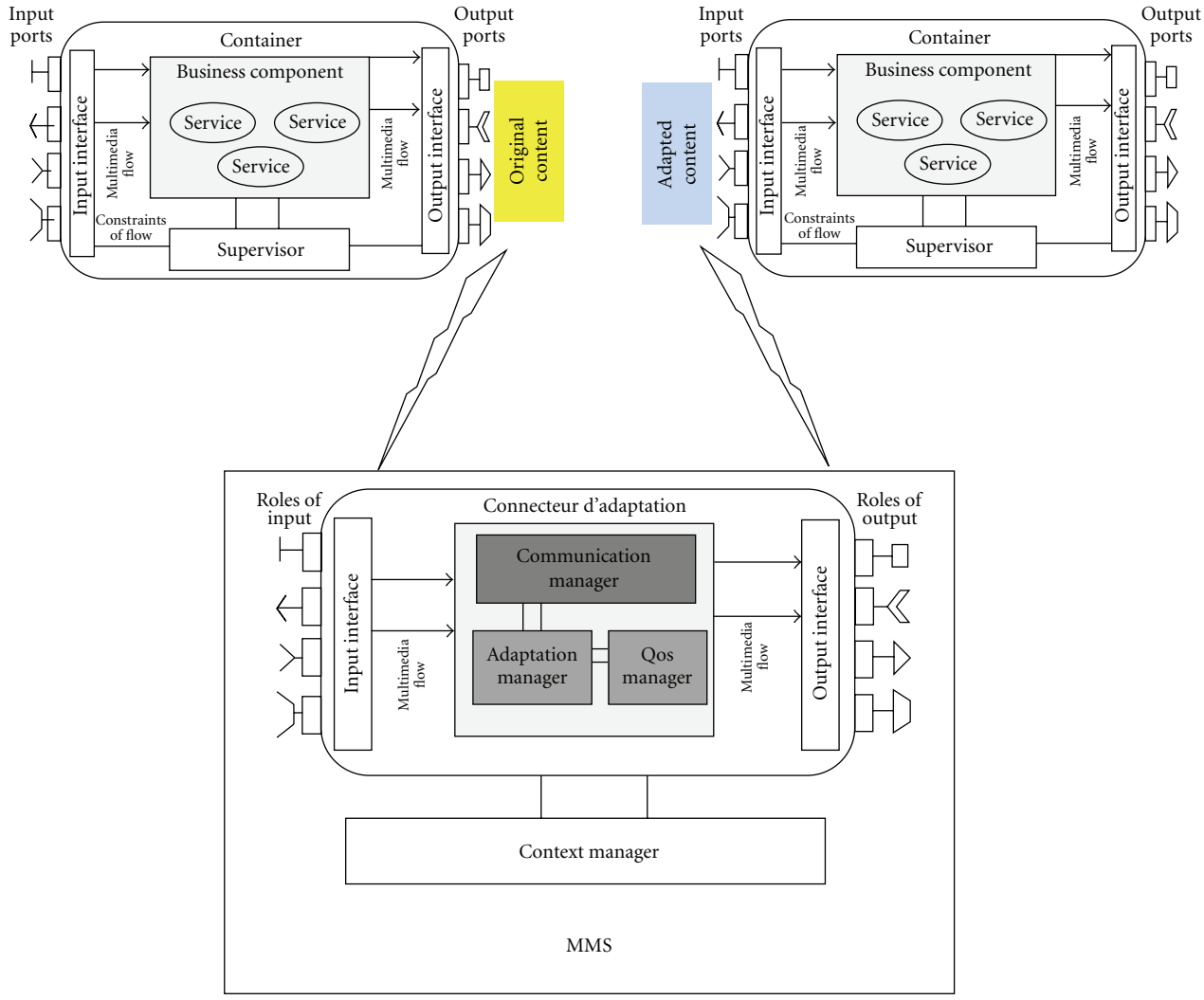

Figure 21: MMSA architecture for multimedia messaging service.

7. Case Study: Modeling and Implementation of Adaptation System for Wireless Networks Phone

Multimedia Messaging Service (MMS) is a standard in mobile messaging. Like SMS *(Short Messaging Service)*, MMS is a way to send a message from one mobile to another. The difference is that MMS can include not only the text but also, sound, images, and video. It is also possible to send MMS messages from a mobile phone to an email address.

While mobile phone users can create and send their own MMS messages, perhaps the biggest use of MMS is likely to be companies sending MMS messages to subscribers, enquirers, or customers. For example, a company could send

visitors an MMS map to help them finding their office. Other possible applications include weather reports, news and sport bulletins, and so forth.

To clarify our proposition, we use the example of a telephone network, and especially the MMS. This service is responsible for managing all multimedia messages sent from one device to another. However, the message received by the receiver is not compatible with the formats that it accepts, an "incompatible message" is displayed, so the receiver cannot read this message.

There are some interesting challenges with MMS that do not exist with SMS: Content adaptation *(multimedia content created by one brand of MMS phone may not be entirely compatible with the capabilities of the recipients' MMS phone)*, distribution lists, bulk messaging, handset configuration, and so forth.

MMS is considered as a connector. To make it adequate with adaptation needs, the MMS must be enriched with other components in order to be adapted to the needs of multimedia device.

The best way is to have independent services providing the adaptation and to return the adapted message to the MMS to assure the sending of the message to the receiver (see Figure 20).

The architecture of an MMS application on the MMSA approach is as shown in Figure 21.

In this architecture, MMS is considered as a communication component that cooperates with an adaptation sub-component and a QoS subcomponent to build the adaptation connector between two devices.

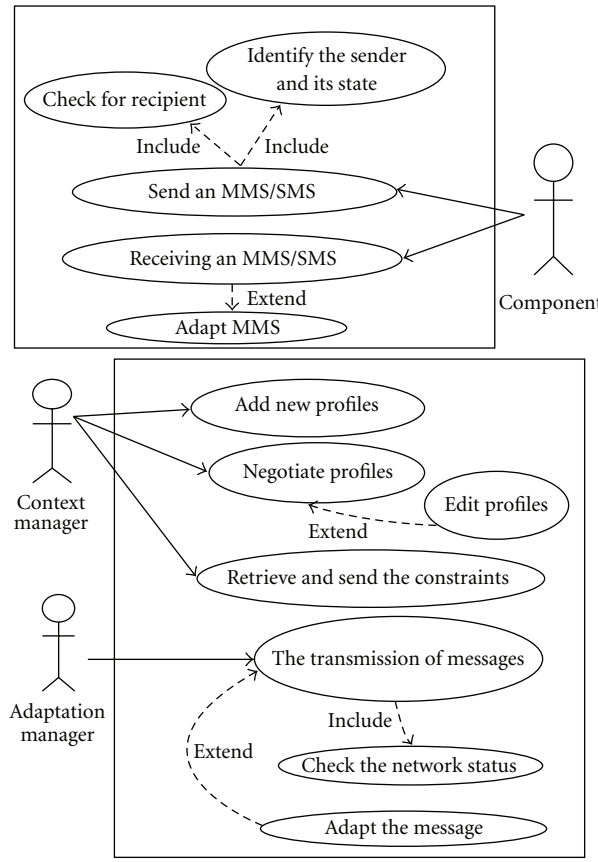

Figure 22: Use case diagram.

7.1. Modeling System by UML. To describe the context in which the MMS will be used, we use UML models providing the context independent model (CIM). The architectural elements identified of our system are as follows (Figure 22).

(i) *Sender and receiver components*: it sends or receives messages adapted to its characteristics.

(ii) *Adaptation connector*: it provides communication between components and adapting messages.

(iii) *Context manager*: it is responsible to make any necessary updates in the profiles database *(adding new profiles, modifying existing profiles)*.

UML sequence diagrams models the exchanged flows within our system in a visual manner, enabling us both to document and validate our logic. Sequence diagram focuses on identifying the behavior of the system. In addition to sender device and receiver device, sequence diagram (Figure 23) contains the MMS server, the adaptation manager, and the context manager.

7.2. Implementation. Every day, billions of images are transferred over networks, from a camera to a computer or from a mobile phone to another. Therefore, we will propose an application that ensures the adaptation of image flow exchanged between the different devices. Adaptation applies the conversion on digital image, which turns an image into another image, according to the context, in order to modify or completely change some properties of an image.

The adaptation platform is an instance of the MMSA architecture (Figure 21), it is created in Java; the platform (Figure 24) contains all the functions involved in the process.

The platform provides the following services:

Resize. In case of the size of the screen is a different recipient than the sender, there is a call to the resizing, which exists in the class image *(package adaptation)* in our program. This function, queries the database for the screen size of the recipient to apply on the selected image. Figure 25 illustrates this.

Grayscale. If the image is sent in color and the recipient device does not support colors, the image must be changed according to the characteristics of the recipient. Figure 26 illustrates the application of this function.

Transcoding. If the recipient does not support image format, for example, if the image is of BMP and the mobile recipient accepts JPEG (Figure 27), it is necessary to implement the algorithm format conversion to JPEG.

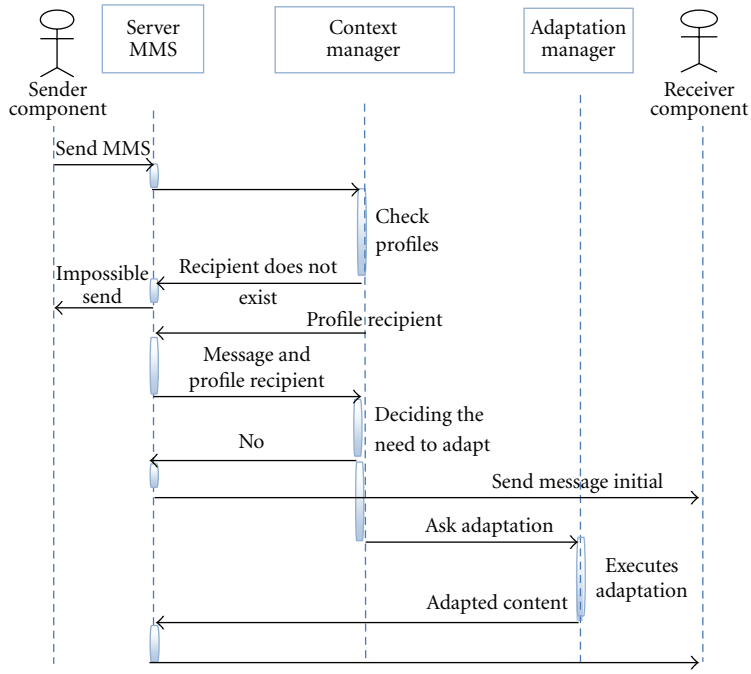

Figure 23: Sequence diagram.

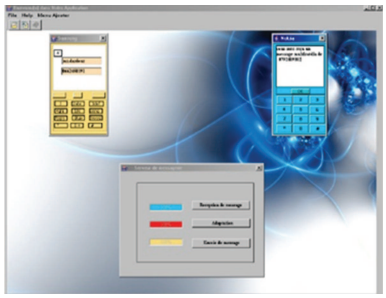

Figure 24: Adaptation Platform.

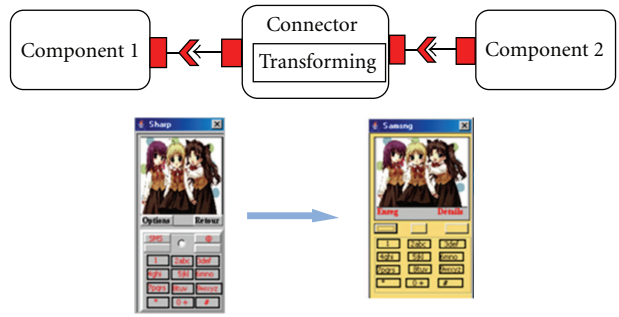

Figure 25: Image resizing.

8. Discussion

Software components are reusable software entities which goals are cost reduction in development, maintenance, and in software evolution. Many propositions claim the development mode based on the assembly of software components. Despite the common vocabulary *(component, port, interface, service, configuration, connector)*, these propositions are varied regarding their origins, their objectives, their concepts, and also their mechanisms.

Although they have much in common, the goals sought by the ADL are not always the same. For example, some are particularly interested in the semantics of components and connectors while others are mostly concentrated on defining the interconnections between components and connectors. Each ADL has its strengths. The choice of a language rather than another is guided by needs and expectations of the system designer.

Approaches like [16, 18, 21, 22] allow the separation of the functional concerns. They were proposed in order to capitalize the functional needs in modular entities. Several ideas were proposed within this perspective. We mainly distinguish two categories of approach for software architectures: those inspired on Component-Based Software Engineering (CBSE) and Service-Oriented Architecture (SOA). In the first case [16, 21, 23], the focus is on the static structure of the system; the software elements are components assembled by connectors in configurations. In the second case [18, 22, 24, 25], the focus is on the functional structure of the system; software elements are functionalities *(services)* linked by relations of collaboration or combination. The model proposed in this paper could be described as hybrid as it includes components and proposes services by these components.

Modern applications are more and more developed according to ADL-based development processes [26]. The

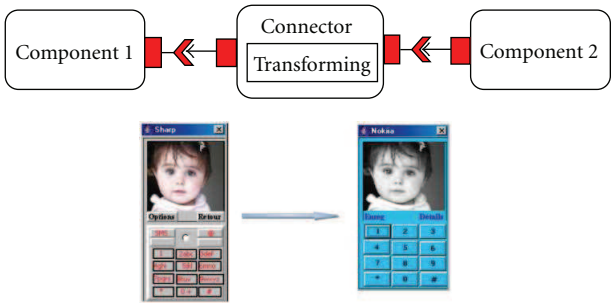

Figure 26: Image of color (grayscale).

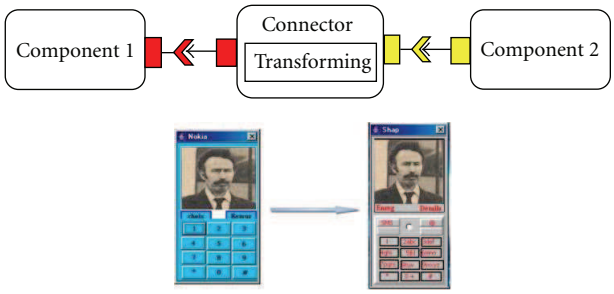

Figure 27: Adaptation of format (JPEG to BMP).

ADLs allow analysis and verification of properties early in the development cycle that the future system will have to satisfy, in particular the homogeneity and compatibility properties of components handling various media. Indeed, the current applications *(multimedia, embedded systems, communication systems, etc.)* consider the media notion as an important characteristic of their behavior [27, 28]. Most of existing ADLs such as SPT-UML [29], MARTE [30], and AADL [31] do not take into account the adaptation and the properties related to multimedia flow during the software construction phase. Some of them, treat the problem of heterogeneity by modification of the configuration parameters (addition, withdrawal, or replacement of components) [32] or by a meta-model which verifies the adequacy of service regarding its context and research of the adaptation strategy [33].

A simple component language [34] proposes a comparison of the principal characteristics of the components languages: component, interface, port, service and connector. The main objective of this work is to take into consideration the unforeseen connection of the developed components in an independent way. As a solution, it proposes the production of reusable and configurable connectors through the association of a particular service to the provided ports which will be used in the absence of the requested service at port level. A drawback of this work is the absence of the integration mechanisms of the new communication services which ensures the evolution of architecture towards new needs; it also lacks techniques for checking the quality of architectures and the provided services.

Component Connector Configuration (C3) [1] is an approach based on software architectures. It makes it possible to describe a view of logical architecture in order to automatically generate physical architecture for all the application instances. The idea is based on the refinement and the traceability of the architectural elements. The software architecture is described in accordance with the first three levels of modeling defined by the OMG [35, 36]. Consequently, to describe logical architecture, three types of connectors are defined: the connection connector (CC), the composition/decomposition connector (CDC), and expansion/compression connector (ECC). The connectors proposed do not ensure the connection of the heterogeneous components and do not take into account the semantics of configurations and that of the links between components.

MMSA tries to propose a generic solution to the problem of incompatibility of components, in order to ensure the interoperability of components in real time. It proposes a presentation of architecture starting from a set of components, connectors and services. A component provides a set of service through its interface "provided" and asked a set of service through its interface "required". It holds a manifest that describes all information needed for their composition. Then, a connector is responsible to ensure the communication between the components connected. A connector can be used to provide multiple connections; it is composed of three main components: a communication component, a QoS component, and an adaptation component. The adaptation component can be a web service, if there is no component that can perform the requested adaptation. A service is a significant task *(a set of actions with an interface that clearly describes the parameters and the function realized by this service)* and is provided by a component or by another provider such as Web service.

9. Conclusion

We proposed a generic meta-model for the description of software architectures. This meta-model integrates multimedia and QoS concepts. This enables to present separately data flow parameters and media which present a very important aspect of component configurations and assemblies. The contribution of this work is situated in a context of abstraction level-based description integrating functional and nonfunctional concerns of the components. This ensures a quality of the components assembly by inserting the adaptation connectors, as well as management of adaptation service quality. The main advantages of MMSA are the consideration of the multimedia aspect and the separation between the functional and nonfunctional concerns of the components.

Our proposition can be used as a support to develop the management applications of the numerical resources (DAM: Digital Asset Management), for example. Such applications handle a wide variety of media, and communicate with the users through various platforms (Cellphones, PDA, PC, portables, etc.). MMSA can bring an effective solution to the development of DAM. It offers the possibility to take into consideration the factors generating the incompatibilities between components in the DAM architecture. It gives a solution at the architectural level by injecting the adaptation

connectors at the execution level by the management of QoS and the reconfiguration of these connectors.

As we have seen, the world is very wide in terms of the diversity of image. We tried to implement the essentials in the field of image treatment to better understand the problem of heterogeneous components and data flows.

As a perspective, we propose to develop a modeling tool for our approach and to investigate other nonfunctional concerns. The development of the service quality aspect must be also taken into account.

References

[1] A. Amirat and M. Oussalah, "First-class connectors to support systematic construction of hierarchical software architecture," *Journal of Object Technology*, vol. 8, no. 7, pp. 107–130, 2009.

[2] R. J. Allen, *A formal approach to software architecture*, Ph.D. thesis, School of CompScien, Carnegie Mellon University, 1997.

[3] N. Medvidovic, D. S. Rosenblum, and R. N. Taylor, "Language and environment for architecture-based software development and evolution," in *Proceedings of the International Conference on Software Engineering (ICSE '99)*, pp. 44–53, Los Angeles, Calif, USA, May 1999.

[4] D. Garlan, R.-T. Monroe, and D. Wile, "Acme: architectural description component-based systems," in *Foundations of Component-Based Systems*, pp. 47–68, Cambridge University Press, Cambridge, UK, 2000.

[5] E. Dashofy, A. V. D. Hoek, and R. N. Taylor, "A comprehensive approach for the development of XML-based software architecture description languages," *ACM Transactions on Software Engineering and Methodology*, vol. 14, no. 2, pp. 199–245, 2005.

[6] R. Allen, S. Vestal, D. Cornhill, and B. Lewis, "Using an architecture description language for quantitative analysis of real-time systems," in *Proceedings of the 3rd International Workshop on Software and Performance (WOSP '02)*, pp. 203–210, ACM, Rome, Italy, 2002.

[7] C. Szyperski, *Component Software: Beyond Object-Oriented Programming*, Addison-Wesley, Reading, Mass, USA, 2002.

[8] E. Bruneton, T. Coupaye, M. Leclercq, V. Quéma, and J.-B. Stefani, "An open component model and its support in java," in *Component-Based Software Engineering*, vol. 3054 of *Lecture Notes in Computer Science*, pp. 7–22, Springer, Berlin, Germany, 2000.

[9] R. Monson-Haefel, *Enterprise JavaBeans*, O'Reilly & Associates, Sebastopol, Calif, USA, 1999.

[10] J. Cheesman and J. Daniels, *UML Components: A Simple Process for Specifying Component-Based Software*, Addison-Wesley, Reading, Mass, USA, 2000.

[11] J. Aldrich, C. Chambers, and D. Notkin, "ArchJava: connecting software architecture to implementation," in *Proceedings of the 24th International Conference on Software Engineering (ICSE '02)*, pp. 187–197, ACM, Orlando, Fla, USA, May 2002.

[12] O. Nierstrasz and L. Dami, "Component-oriented software technology," in *Object-Oriented Software Composition*, pp. 3–28, Prentice-Hall, Englewood Cliffs, NJ, USA, 1995.

[13] J. C. Seco and L. Caires, "A basic model of typed components," in *Proceedings of the 14th European Conference on Object-Oriented Programming*, vol. 1850 of *Lecture Notes in Computer Science*, pp. 108–129, 2000.

[14] D. Chappell, Introducing SCA, 2007, http://www.davidchappell.com/.

[15] L. Seinturier, P. Merle, D. Fournier, N. Dolet, V. Schiavoni, and J.-B. Stefani, "Reconfigurable SCA applications with the FraSCAti platform," in *Proceedings of the IEEE International Conference on Services Computing*, Bangalore, India, September 2009.

[16] R. Allen and D. Garlan, "A formal basis for architectural connection," *ACM Transactions on Software Engineering and Methodology*, vol. 6, no. 3, pp. 213–249, 1997.

[17] N. R. Mehta, N. Medvidovic, and S. Phadke, "Towards a taxonomy of software connectors," in *Proceedings of the International Conference on Software Engineering (ICSE '00)*, pp. 178–187, ACM, June 2000.

[18] C. Attiogbé, P. André, and M. Messabihi, "Correction d'assemblages de composants impliquant des interfaces paramétrées," in *Proceedings of the 3sd Francophone Conference in Software Architectures*, Hermès, Lavoisier, 2009.

[19] T. Coupaye and J.-B. Stefani, "Fractal component-based software engineering," in *Proceedings of the Conference on Object-Oriented Technology (ECOOP '06)*, M. Südholt and C. Consel, Eds., vol. 4379 of *Lecture Notes in Computer Scienc*, pp. 117–129, Springer, 2006.

[20] V. Quéma, *An approach to building configurable software infrastructures dramatically*, Ph.D. thesis, National Institute polytechnic of Grenoble, 2005.

[21] K. Bergner, A. Rausch, M. Sihling, A. Vilbig, and M. Broy, "A formal model for component ware," in *Foundations of Component-Based Systems*, pp. 189–210, Cambridge University Press, New York, NY, USA, 2000.

[22] E. M. Maximilien and M. P. Singh, "Self-adjusting trust and selection for web services," in *Proceedings of the 2nd International Conference on Autonomic Computing (ICAC '05)*, pp. 385–386, June 2005.

[23] C. Szyperski, *Component Software: Beyond Object-Oriented Programming*, Addison-Wesley, Reading, Mass, USA, 1997.

[24] M.-P. Papazoglou, "Service-oriented computing: concepts, characteristics and directions," in *Proceedings of the 4th International Conference on Web Information Systems Engineering (WISE '03)*, pp. 3–12, IEEE Computer Society, Roma, Italy, 2003.

[25] B. El Asri, A. Kenzi, M. Nassar, and A. Kriouile, "Towards an MVSOA architecture for the implementation of multiview components," in *Proceedings of the 3sd Francophone Conference in Software Architectures*, pp. 1–17, 2009.

[26] P. Avgeriou and U. Zdun, "Modeling architectural patterns using architectural primitives," in *Proceedings of the 20th Annual ACM Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA '05)*, pp. 133–146, ACM, October 2005.

[27] A. Avižienis, J.-C. Laprie, B. Randell, and C. Landwehr, "Basic concepts and taxonomy of dependable and secure computing," *IEEE Transactions on Dependable and Secure Computing*, vol. 1, no. 1, pp. 11–33, 2004.

[28] S. Balsamo, M. Bernado, and M. Simeoni, "Performance evaluation at the architecture leveL," in *Formal Methods for Software Architectures*, vol. 2804 of *Lecture Notes in Computer Science*, pp. 207–258, Springer, Berlin, Germany, 2003.

[29] S. Graf and I. Ober, "How useful is the UML realtime profile SPT without semantics?" in *Proceedings of the Specification, Implementation and Validation of Object-oriented Embedded Systems Associated with Real-Time and Embedded Technology and Applications Symposium*, Toronto, Canada, 2004.

[30] Object Management Group, "UMLTM Profile for Modeling and Analysis of Real-Time and Embedded systems (MARTE)," http://www.omg.org/cgi-bin/doc?realtime/2005-02-06.

[31] Society of Automotive Engineers, "Architecture Analysis & Design Language (AADL)," SAE Standards no. AS5506, November 2008.

[32] C. Marcel, R. Michel, M. Christian, L. Calin, and M. Costin, "Dynamic adaptation of services," in *Proceedings of the Déploiement et (Re) Configuration de Logiciels (DECOR '04)*, Grenoble, France, 2004.

[33] M. Cremene, M. Riveill, and C. Martel, "Autonomic adaptation solution based on service-context adequacy determination," *Electronic Notes in Theoretical Computer Science*, vol. 189, pp. 35–50, 2007.

[34] L. Fabresse, C. Dony, and M. Huchard, "Foundations of a simple and unified component-oriented language," *Computer Languages, Systems & Structures*, vol. 34, no. 2-3, pp. 130–149, 2008.

[35] OMG, "Unified Modeling Superstructure," 2007, http://www.omg.org/spec/UML/2.1.2/Superstructure/PDF.

[36] OMG, "Unified Modeling Language: Infrastructure," 2007, http://www.omg.org/spec/UML/2.1.2/Infrastructure/PDF.