An Architecture for the Development of Context-aware Services based on MDA and Ontologies

Samyr Vale, Slimane Hammoudi

Abstract - Most ubiquitous or pervasive applications focus on the development of legacy or ad hoc architectures to capture and adapt context. There is no consensus or standards for context definition, representation and reasoning. These architectures join business logic with context-aware statements into the same artifact and do not provide context interoperability and reuse. In this work we present an interoperable architecture for the development of context-aware services based on Model Driven Engineering and ontologies. Founded on the concerns separation in individual and independent models we provide interoperability and reuse to context-aware development. We also present our well defined context metamodel based on the OMG's ODM (Ontology Definition Metamodel) and supported by OMG's MDA (Model Driven Architecture).

Index Terms - Ubiquitous, Context-awareness, MDA, Ontologies.

I. INTRODUCTION

The IT community is waking up to the need for the development of ubiquitous or pervasive applications due to the emergence of advanced mobile devices with Internet access. In order, distributed and Internet supported applications have reached the top of the podium of the most developed applications nowadays. Internet based applications increase business activities by the everywhere access terminals.

Web Services have been spread over Internet due to their interoperability and low-coupled characteristics. They have also been presented as the best middleware architecture in use.

Ubiquitous applications development is the pledge to the next tendency in software development. Ubiquitous (also named pervasive applications) was idealized by Weiser [1]. By the use of sensors he intended to help users in their daily activities in a transparent way. Context-awareness tailors pervasive applications by the use of user's current situation.

Manuscript received January 7, 2009. This work was supported in part by FAPEMA (Foundation for the Support of the Research in Maranhão).

Samyr Vale is with LERIA-UFR Sciences, Université d'Angers, 2 Bd Lavoisier, 49000, Angers - France (samyr.vale@gmail.com)

S. Hammoudi is with ESEO, 4 rue Merlet de la Boulaye, 49009, Angers - France.

Context information can be supplied by a specific device, ontology, data base or an XML archive and it can refers to device characteristics, or a user profile and preferences, or a location and time. The most used definition of context is "any relevant information about the user-application interaction, including the user and the application themselves"[2]. Context-aware applications use context information to provide relevant computational solutions and information to users without their interaction. The development of context-aware applications is an arduous task. Context is dynamic and services must be adaptable to contextual changes. In others words, applications must change their behavior according to changes in each element in the computational environment specified as context. Behavior changes can be realized by a new process start, a new service invocation, a new message format adaptation or a new interface creation.

As a rule, context definition and adaptation are developed on the execution level. This solution can be applied to very strict domains where developers know context source, message format, and in cases where change in input context performs as a switch-case statement. Nevertheless, context is dynamic and context-aware applications have to adapt themselves for each context change. Problems of capture, interpretation, representation and reasoning are current in context-aware applications. There exist many researches into context-aware software development. The majority are about how to capture and provide context information and others about context adaptation. In most cases, contextual logic and business logic are joined into the same application artifact and context-aware activities are limited to context data request. Due to this, context identification, representation, reasoning and reuse are arduous works. Most context-aware applications are based on legacy or ad hoc architectures and are no-standard based applications.

Our work focuses on context-aware mobile distributed applications. We promote the development of context-aware statements, context information definition and business logic in independent and individual models and in different abstraction levels. We have investigated the application of model driven approach for context-aware applications development [3][4][5].

In this paper we propose the use of the Model Driven Engineering approach to develop distributed context-aware applications by the separation of concerns in different models. Business logic, architectural details and context information can be modeled in an independent way providing adaptability, interoperability, reuse and transformation possibility through different languages and to different platforms.

OMG's (Object Management Group) Model Driven Architecture (MDA) [6] standard has been adopted for Model Driven Development by the development of PIM (Platform Independent Models) and PSM (Platform Specific Models) models.

Some works have proposed the use of the MDA approach to context-aware applications. But until now there is neither a pattern for the context metadata representation nor a consensus about the best formalism to model context-aware applications. We also use ontology concepts to represent our context metamodel. Ontologies and models together are now supported by ODM (Ontology Definition Metamodel) [7]. ODM is the OMG (Object Management Group) newest proposition for providing representation, management and interoperability based on MDA (Model Driven Architecture) to business semantic. Ontologies have been widely used as the best solution for context reasoning [8].

We use Web Services as the target platform technology because of their interoperability and Internet support characteristics. There are also some works using Web Services for the development of Context-aware applications, but most focus on the use of message parameters to transmit context information by the SOAP protocol. These solutions do not show how applications can adapt new activities by context changing and how these activities are implemented.

The remainder of this paper is organized as follows: Section 2 provides the general description of the ARCAMODE architecture. Section 3 presents the context metamodel and model into the Context View description. Section 4 presents the Component Composition View. Section 5 depicts the Service View of ARCAMODE. Section 6 presents the Adaptation View followed by our conclusions.

II. ARCAMODE (AN ARCHITECTURE FOR CONTEXT-AWARE SERVICES DEVELOPMENT BASED ON MDA AND ONTOLOGIES)

The Model Driven Architecture (MDA) [6] predicts that everything can be modeled. The different abstraction levels provided by MDA are relevant for systems design because different systems requirements are treated by different levels.

For each level there are some representation formalism standards. MOF (Meta-object Facility) is the OMG's formalism used for meta-metamodel representation, the top of the abstraction layer also named M3 layer. For M2 or metamodel layer, UML is the OMG most used solution for the development of general purpose metamodels. The M1 abstraction layer defines models of the real world. M0, the bottom layer, is reserved for instances of software application or code. Different representation languages can be created and applied in these abstraction levels to represent concepts.

There are some transformation languages (e.g. OMG's QVT, ATL, YATL, BOTL) also used to transform models from more abstract level until the application code.

As presented in Figure 1 our ODM and MDA based architecture owns different models which represent the architectural views. The architecture named as ARCAMODE provides a Service, Context, Adaptation, Composition and Business Views.

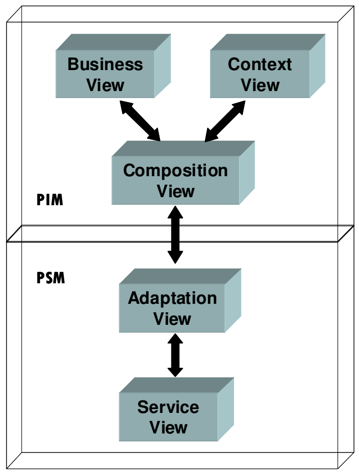

Figure 1. ARCAMODE Views

The Business, Context and Composition Views are expressed by PIM models, *i.e.*, these models abstract platforms details. The Context View is responsible for context information definition, capture, representation and interpretation. This view is also responsible for defining context-aware elements, *i.e.,* software elements that realize activities based on context information (components and operations).

The Composition View establishes the link between business logic and contextual logic, *i.e.,* how business statements and contextual activities will be executed harmoniously to provide relevant results for user needs.

The Adaptation and Service Views are expressed in PSM models, *i.e.,* they specify target platforms details, middleware technologies, connectors, protocols and others platform specific requirements. The details of each model of the ARCAMODE architecture are described in the next sections.

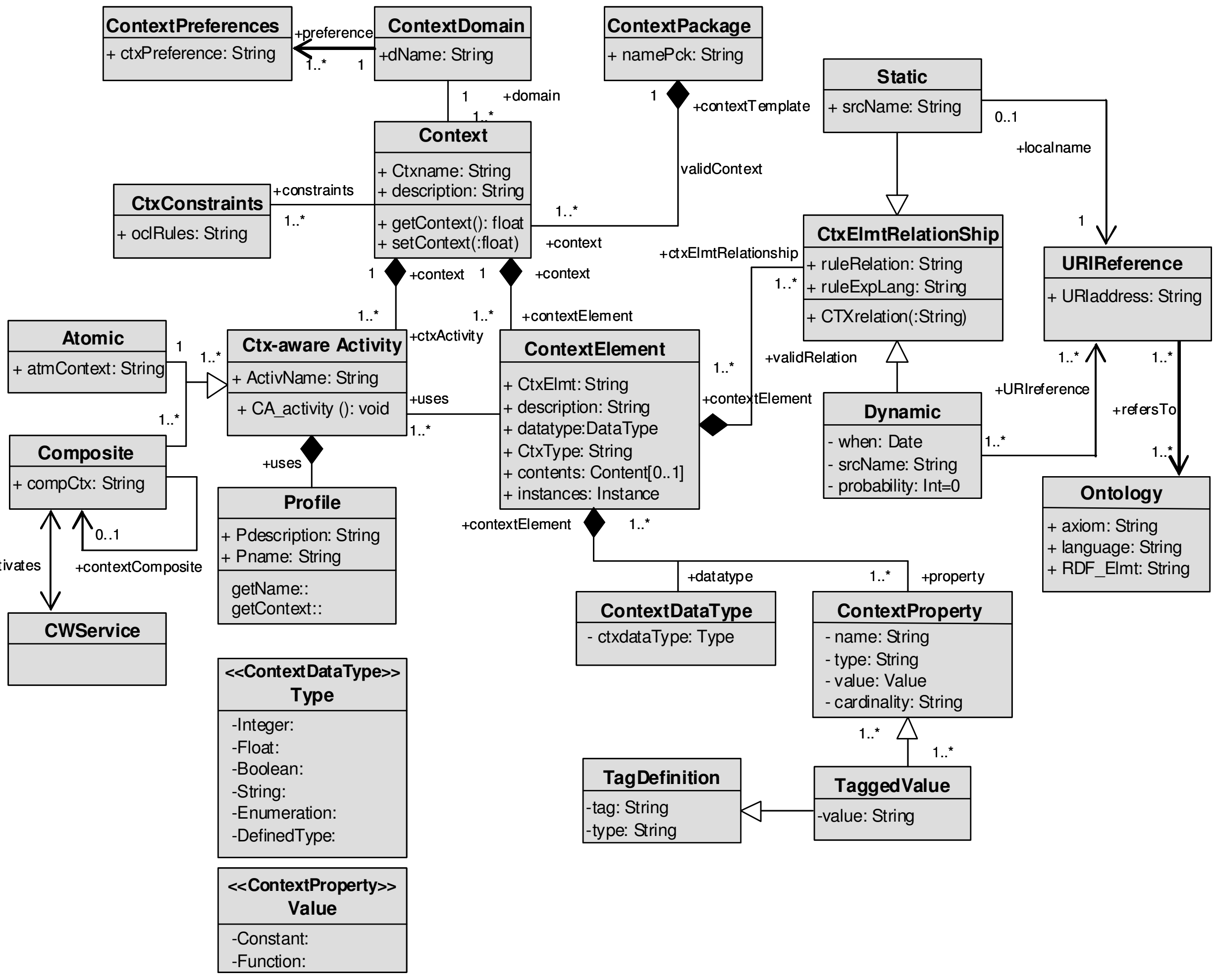

Figure 2. The Context Metamodel

III. THE CONTEXT VIEW

Context information and context-aware activities are defined and represented in the ARCAMODE's Context View.

To represent context information we use ontology concepts. "*An ontology defines the common terms and concepts (meaning) used to describe and represent an area of knowledge*" [7].

Ontologies have been used to represent complex knowledge and to improve semantic and consistence in web applications. There exist some works using ontologies to represent context information due to the variety of their meanings and sources.

Figure 2 presents our context metamodel represented by the UML formalism. The UML is the model language used by the ODM to represent ontology models based on the Model Driven Architecture (MDA). The context metamodel represents the context domain and definitions.

We use W3C`s RDF (Resource Description Framework) to represent context data at model level [9].

RDF Schema (RDFS) is MOF supported and permits definitions of vocabularies to represent context at metamodel level. A class in RDFSchema is a defined term and its collection members, defined using *rdf:type*. Classes in *RDFSchema* have no pre-defined definition or constraints on their membership. Each class in our Context Metamodel represents an RDFSchema. A property in *RDFSchema* is a defined term and its domain and range which indicate which resource class this property applies (domain) and to which resource class this property leads (range).

The RDFS *ContextElement* is an RDF Resource that represents context elements. Resource is a semantic concept used to represent context at meta-metamodel level [7]. The *ContextElement* is any element that represents context. We employ Dey's context definition as aforementioned [2].

In Figure 3 we represent a context model based on the context metamodel depicted in Figure 2. We have defined in our Context Model the principal context elements used in most of context-aware mobile applications. We have defined three primary context elements: person, computational artifact and environment. Usually mobile applications refer to a user that owns a device with some specific characteristics like display, keyboard, battery, memory, GPS functions, mini-browser and voice. This user is situated in an environment with some specific characteristics also specified in the model, like location, space, time and weather. This specific device works over a network with specific details like protocol, message and QoS.

The context model conforms to the Context Metamodel and this is based on the ODM metamodel and conforms to the MOF meta-metamodel.

The RDFS *Preferences* class represents URI references or external context sources like an external ontology model.

The RDFS *ContextProperty* class represents context element attributes and associations. The advantage of comparing with UML attributes and associations is the independence provided by Properties. Property can be defined without associations as the opposite of UML traditional classes.

The RDFS *ContextDataType* class groups context data. Context data type can be one of the defined context elements in our context model or another type defined by the engineer. Any context element can be represented at context model level as soon as it conforms to the context metamodel.

Applications cannot adapt themselves to any context information due to their variability of meaning. So specification of context scope is necessary to provide context reasoning for applications.

Since properties can represent many domains, several associations could be needed. Relations could be *Static* (properties that do not change on the time) or *Dynamic* (properties with no persistent values).

Properties can be updated or modified based on the newest context references defined in the ontology represented in a URI reference.

A specific Context owns its context-aware tasks, represented by the Ctx-*awareActivity* class. A context-aware activity uses a specific context element and stores the user contextual activities in a *Profile*. These activities are realized by Web services and they can be *Static* or *Composite* with the coordination of multiple services to provide the required context-aware activity.

A specific context can represent a specific domain and can be defined in an individual and independent *ContextPackage*.

IV. THE COMPONENT COMPOSITION VIEW

In this section we present the Contextual Component Composition View.

Providing a well-detailed definition of context is not sufficient to adapt context and to change applications behavior by the information provided. Business logic must implement activities to be executed in context changes. For each context provided applications must adapt themselves providing the solution required. This is the principle of context-awareness.

Many solutions presented in the literature focus on context data and middleware, but how to manipulate and process this information is overlooked in many cases.

We separate business logic and contextual logic into individual components. Business Process Components are responsible for business logic implementation and Contextual Process Components are responsible for providing changes in application behavior by the context provided.

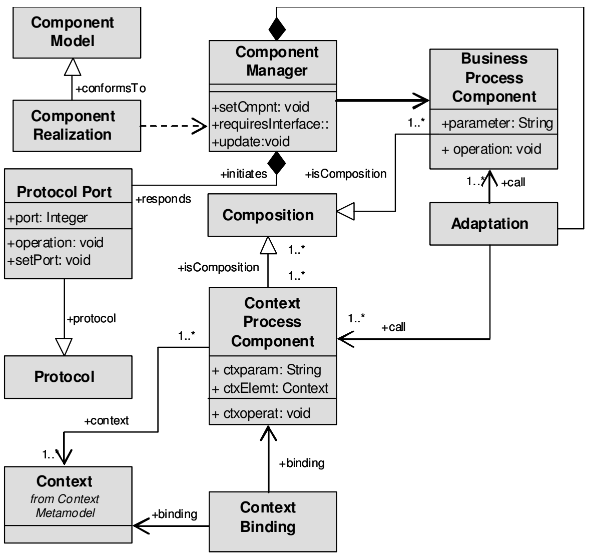

Figure 4. Component Composition Model.

Figure 4 presents the component composition model of the ARCAMODE Architecture. It implements new functionalities for application and in complex context-aware systems each Contextual Process Component can treat some context domain. These components will be integrated in a middleware infrastructure. Composition Process components implement roles in a process. An EJB (Enterprise Java Bean) or a CORBA Component can be a process component implementation.

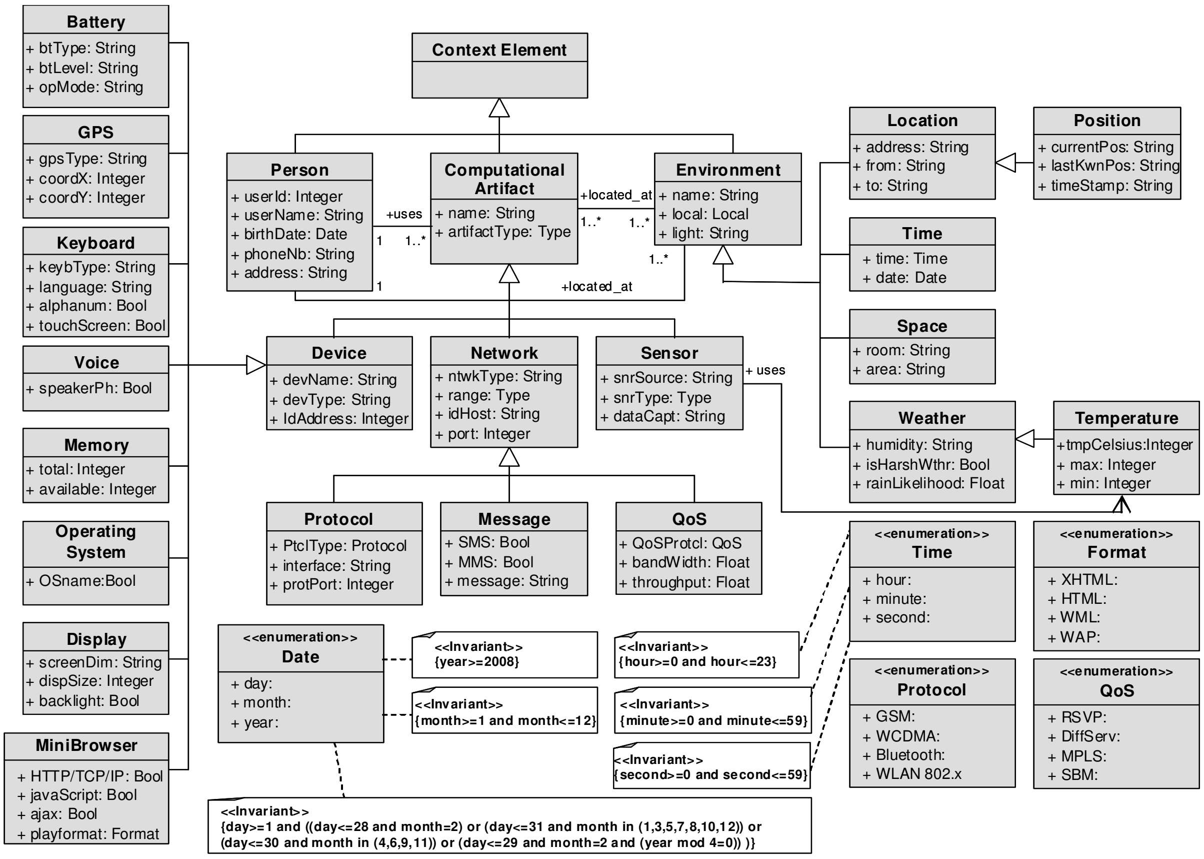

Figure 3. Context Model

Process components represent active processing units. They can interact by the use of ports represented by the Protocol Port element and by the use of a communication protocol (represented by Protocol class) they can be composed and make communication.

Composition specifies how components work together and how they can be connected. In our Composition Component View we present a particular element to bind Contextual and Business components. This is the role implemented by the Component Manager element.

The Component Manager is also responsible for adapting external components and communication protocol definition. Context Process Component implements contextual logic, *i.e.,* activities realized through a specific context. It binds context information provided by the Context Element. The Context Element is defined in the Context Metamodel.

V. THE SERVICE VIEW

This section depicts the architectural Service View. The Service View is based on the WSDL [10] metamodel and is represented by the UML formalism in Figure 5.

Services are the base of ARCAMODE architecture due to their interoperability benefits. *Context Service* represents Contextual Services, i.e., services which provide context information between clients and components.

Services can also be composed of services represented by the Context Service Composition class.

Context Service owns an interface represented by *Context Service Interface* class. *Context Service Interface* aggregates operations (represented by the Operation class) and operations aggregate messages (represented by the Message class).

The *Message* class is responsible for transmitting context data as message part represented by the *Part* class. Message refers to the Context Metadata Element which supplies Context data format and semantic details.

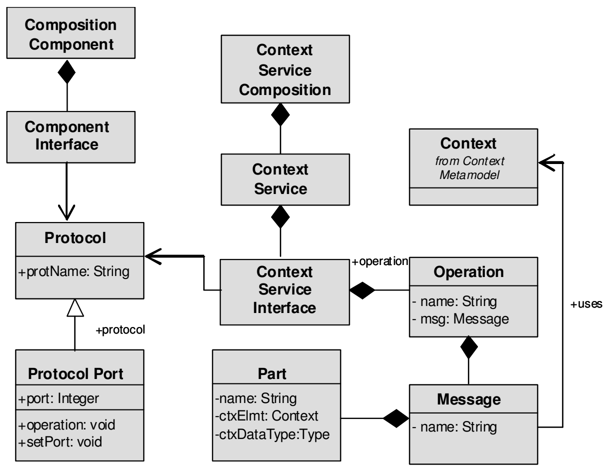

Figure 5. Service Model.

OMG (Object Management Group) and W3C (World Wide Web Consortium) have presented and have been working in MDA and Web Services patterns, respectively. This approach leads to a loosely coupled development of distributed components based on Model Driven Engineering.

VI. ADAPTATION VIEW

The Adaptation View provides a composition component formed by one or more business components, one or more context components and a connector component. As shown in Figure 6 this composition component is named CC Manager (Composite Component Manager).

Composition is an abstraction of process component and describes how instances of process components are configured and connected to implement the composition process component.

The CC Manager is composed of components and other software artifacts which will make communication among themselves or with other business components. The components in CC Manager are logically independent but own specific interfaces to interact. CC Manager can have the same interface as Business Components and Context Components or not. It can implement interfaces for new external components or services.

The connector component is responsible for putting context and business components in communication.

Figure 6 presents the component interaction provided by the CC Manager. The Connector component provides a common protocol for components and services. Its main role is the adaptation of the different architectural elements by the use of the provided protocol.

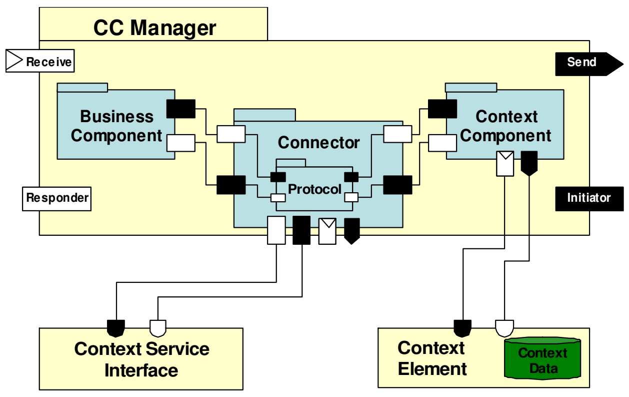

Figure 6. Component Manager.

Communication among elements is realized by ports whose concept is inherent from UML2.0 components. Connector realizes middleware activities, bindings and implements interfaces. The protocol provides SOAP [11] communication to support Web Services.

The Context component is responsible for contextual activities implementation of the context acquired. It is also responsible for managing context information and to implement interface between itself and the context element.

Each component model provides an interface and protocol. Context component and Business component protocols are not represented in Figure 6 once they are inherent for each component.

Business component can be modeled in an independent way and business logic developers do not have to know contextual details. Interfaces and protocols are the bridge among all architectural elements. Communication is realized by the Connector component.

The connector component can provide multiple protocols and realize parser activities. It can use SOAP protocol for web services communication and other protocol for communication between components (e.g. CORBA components). Figure 7 presents the Context Protocol Component activities.

Context Protocol is also a Composition Component of the set of components responsible for all the necessary details to make components communication.

The CC Protocol (Composition Component Protocol) establishes and switches messages received from one component to its specific destination (other component or service).

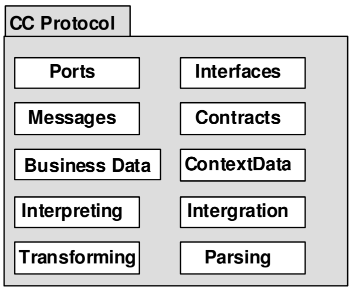

Figure 7. Component Composition Protocol.

Ports Component provides a well defined interaction point between the others components. It identifies which are the components ports and services ports and sets which component port intends to make conversation. Ports element also specifies temporal details of connections.

Interfaces element specifies interfaces, constraints and mapping methods. It contains operation and controls information flows.

Messages are the elements transmitted when a component initiates communication. Messages flow between ports and own parameters and values.

Contracts specify messages format and description of actions realized by the component. The *Interpreting* element is responsible for messages parameters interpretation.

By transformation techniques messages are adapted according to contract specification. Messages must be conformed to a contract to be communicable.

The *Integration* element manages business and context data, their semantic and representation formats. This element is also responsible for avoiding data redundancy.

The *Parsing* element provides more flexibility in communication. It is used when business component protocol and context component protocol are compatible. So it receives input and produces output messages.

VII. CONCLUSION

In this paper, we have presented ARCAMODE (Context-aware Service Architecture based on MDA and Ontologies). Our architecture proposes a well defined context metamodel and model and five architectural views. It provides many advantages to context-aware software development:

- context definition and representation in a standard formalism,
- concerns separation of context, context-aware logic, and business logic in different and individual models and in different abstraction levels,
- reasoning of context elements by RDF schemas which supports ontologies development,
- representation of context and context-aware elements in UML based on the ODM (Ontology Definition Metamodel) and supported by MDA (Model Driven Architecture),
- models transformation possibilities by traditional transformation languages,
- context reuse and interoperability,
- web services as target platform and interoperable middleware.

REFERENCES

[1] Weiser, M. The Computer for the 21s century, Scientific American, 1991. 94-104.

[2] A.K. Dey, D. Salber, and G. D. Abowd. A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications, Human-Computer Interaction Journal, pp. 97–166, 2001.

[3] Vale S., Hammoudi S., Towards Context Independence in Distributed Context-aware Applications by the Model Driven Approach. ACM SIPE'08, July 7, 2008, Sorrento, Italy.

[4] Vale S., Hammoudi S., Context-aware Model Driven Development by Parameterized Transformation. MDISIS held with CAISE'08.pp. 121-133, June 2008, Montpellier-France.

[5] Vale S., Hammoudi S., Model Driven Development of Context-aware Service Oriented Architecture. Pergrid'08 held with 11th IEEE Conference on Computational Science and Engineering, 2008, Sao Paulo-Brazil.

[6] OMG. Model Driven Architecture (MDA), document number ormsc/2001-07-01, 2001.

[7] OMG. Ontology Definition Metamodel (ODM). OMG document OMG/ RFP ad/2006-05-01, 2006.

[8] Gu Tao. Pung H.K., Zhang D.Q., A Service-Oriented Middleware for Building context-aware Services, Journal of Network and Computer Applications, Elsevier, 2005. pp. 1-18.

[9] W3C. Resource Description Framework (RDF), 2004-02-10

[10] W3C. Web Services Description Language (WSDL)1.1, 2001.

[11] W3C. Simple Object Access Protocol (SOAP) 1.1, 2001.