AN ARCHITECTURE FOR THE DEVELOPMENT OF CONTEXT-AWARE APPLICATIONS IN ROBOTICS

Rossano P. Pinto*, Eleri Cardozo*, Eliane G. Guimarães†, Rodrigo F. Sassi*, Alex Z. Lima*, Paulo R. S. L. Coelho*

* *Computer Engineering and Industrial Automation,*
School of Electrical and Computer Engineering,
State University of Campinas
Campinas - SP - Brazil, 13083-970

† *CenPRA - Renato Archer Research Center*
Campinas - SP - Brazil, 13089-970

Emails: rossano@dca.fee.unicamp.br, eleri@dca.fee.unicamp.br, eliane@dca.fee.unicamp.br, sassi@dca.fee.unicamp.br, trustlix@dca.fee.unicamp.br, pcoelho@dca.fee.unicamp.br

Abstract— The ubiquity offered by today's mobile networks and terminals motivates the development of context-aware applications. A context-aware application is autonomous to adapt its behavior according to the context in which it executes. The adaption process takes into account contextual information, such as capabilities of the accessing terminal, user profile and preferences, and information about the surrounding environment (e.g., geographical location, temperature, and light conditions). Context-awareness is a key issue for the design and implementation of truly adaptive applications. However, due to the absence of supporting models and architectures, context-awareness has been incorporated into existing applications in an ad-hoc way. In order to contribute to a more systematic design and implementation of context-aware applications, this paper introduces a metamodel and a supporting architecture for this class of applications. An application in the field of mobile robotics is presented in order to illustrate the benefits of incorporating context-awareness into the design of autonomous robotic systems.

Keywords— Context-awareness, metamodel, UML Profile, code generation

1 Introduction

Mobile computing is today ubiquitous and affordable thanks to advances in wireless networking and mobile terminals. This ubiquity in communications motivates the development of applications able to adapt autonomously to the context in which they execute. Such applications are called *Context-aware applications* and present a high degree of adaptability and autonomy not found on today's common distributed applications. In order to benefit from the current context of execution, such applications must (1) sense the environment into which they are executing, (2) infer their execution context, and (3) perform adaptation actions. In order to achieve this dynamic behavior, these applications must interact with the resources present in the environment in the same way they interact with their own resources.

The current software development techniques do not take context-awareness into account. Notably, software components, a well established technology for building distributed applications, are not enough for the development of context-aware applications. The reason is that the current component models and supporting platforms do not consider context-awareness. For instance, commercial component models, such as Enterprise Java Beans (EJB), support non-functional requirements in the business domain (transaction, persistence, and security), but requirements assigned to context-aware computing are not supported by those component models. As a result, context-awareness must be incorporated into the application as a functional or non-functional requirement to be fulfilled by the developer, not by the underlying component infrastructure (platform).

This paper introduces a metamodel for the development of context-aware applications built from software components. The metamodel identifies the main elements of a context-aware application. ACORD-CS, a supporting software platform for context-aware applications is also presented, as well as an application in the field of autonomous mobile robotics built according to the metamodel and using the ACORD-CS platform.

Related work

The following related work are worth of mention.

ContextUML (Sheng and Beatallah, 2005) is a UML profile aimed at designing context-aware web services, so it is tied to this technology. It offers abstractions like service (operation, input/output message), sources (web services) of context information, context binding and triggering, constraint and action. A context constraint is modeled as a predicate (boolean function) that consists of an operator and two or more operands: `rainLikelihood`$\geq$ `80%`. Conditions are expressed using UML's object constraint

language - OCL (Warmer and Kleppe, 2003). The action part is a transformation function `filter(M,R)`, where M is the output message and R is a transformation rule. In this case, if the output message is a list of tourist attractions, this function would select only indoor attractions. Our approach uses a tiny language able to make structural and behavioral adaptations that favors the generation of rule based scripts for inference engines like JESS and CLIPS. We argue that OCL and derivatives, like XION (Xio, 2002) and Action Semantics (Act, 2002), are not appropriate for adaptation purposes, as it is too overwhelming for the task.

Ayed (Ayed and Berbers, 2006) presented a UML Profile for the design of platform-independent context-aware applications. The UML Profile offers abstractions for context, context quality, events, periodic information, and associations. Restrictions and adaptation conditions are specified with OCL (Object Constraint Language) (Warmer and Kleppe, 2003). Ayed makes the distinction among structural, architectural and behavioral adaptation. Both structural and architectural adaptation is specified through class diagrams, while behavioral adaptation is expressed through sequence diagrams. Ayed, makes an interesting use of OCL and sequence diagrams to specify alternative execution paths. OCL is used to make a choice among the several execution paths. DYVA (A.Ketfi and N.Belkhatir, 2004) proposes a metamodel for the adaptation of component based software using instrumentation techniques over an already developed application. They insert adaptation code into strategic points of the application, as constructors, factories, and general methods of interest. To do that, the application must be described by an XML document. We argue that, if the source code of the application is not available, or it is too bad written, it is too difficult to create this XML document. Our approach targets the development of new applications instead of instrumenting an already developed one.

This paper is structured as follows. Section 2 introduces the metamodel for context-aware applications. Section 3 presents ACORD-CS. Section 4 illustrates the development of a context-aware application using the ACORD-CS platform. Finally, Section 5 presents some conclusions and final remarks.

2 A Metamodel for Context-aware Applications

The proposed metamodel (Pinto et al., 2005) for context-aware applications is expressed in UML (Unified Modeling Language) (OMG, 2004) through a class diagram shown in Fig. 1.

The metamodel defines seven UML classes

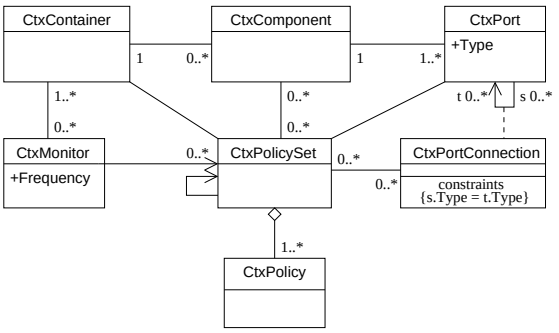

Figure 1: Conceptual model for the proposed UML profile for the development of context-aware applications.

and their relationships that represent the main entities of a context-aware application. *CtxComponent* and *CtxContainer* are concepts that represent components and containers as defined by the existing component models. As such, components are units of deployment and composition, while containers provide the resources necessary for the execution of components. More specifically, containers supply the non-functional requirements and the run-time environment, while components supply the functional requirements of the application.

CtxPort represents component's endpoints for interactions (ports). A port can support different interaction semantics, such as those prescribed in RM-ODP (Reference Model for Open Distributed Processing) (ISO/IEC 10746-2 / ITU-T Rec. X.902, 1995). For instance, the CM-tel component model supports synchronous (request-reply), asynchronous (notification) or stream (flow-based) ports (Guimarães, Maffeis, Pereira, Russo, Cardozo, Bergerman and Magalhães, 2003; Guimarães et al., 2003).

CtxPortConnection represents the connection between two complementary ports, for instance, a producer and a consumer of video flows. This element holds time-varying properties of the connection, such as quality of service (QoS) parameters.

CtxPolicySet performs a central role in this metamodel. It represents a set of adaptation policies that trigger adaptation actions based on acquired context information. The relationship between a *CtxPolicySet* and any other element of the model indicates that an adaptation can take place within a container (e.g., by instantiating a new component), a component (e.g., by changing one of its configuration parameters), a port (e.g., by setting a port output format), and a port connection (e.g., by setting the proper QoS parameters). The model does not enforce any mechanism by which adaptation is performed. Behavioral (parametric) adaptation is usually accomplished via the updating of component and port configuration properties or by calling methods de-

fined by the components for performing adaptation functions. Structural adaptation is accomplished via component creation, destruction, and replacement, or via rearrangements in the port connections. A *CtxPolicySet* is composed of one or more policies represented by the *CtxPolicy* element as shown by Fig. 2.

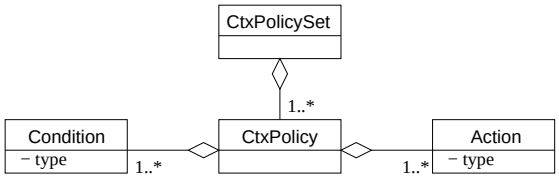

Figure 2: A simple model for adaptation policies.

Each *CtxPolicy* owns one or more *Condition*s that triggers one or more *Action*s when the conditions hold. This model favors the use of rule-based languages to describe adaptation policies.

A *CtxMonitor* is responsible for acquiring contextual information from any source and feed *CtxPolicySet* with this information. *CtxMonitors* can acquire contextual information from sensor readings, service invocation (e.g., a weather forecasting service), user input (e.g., preferences), and so on. This element usually preprocesses the information by performing aggregation, filtering, and scaling.

3 An Architecture for Context-aware Applications

A context-aware application depends on information about the current context in order to perform adaptation actions. This information generally comes from an infrastructure that gathers information about the environment and distributes it to interested parties. In order to favor adaptation, this infrastructure may also provide utility services, such as search and registration of components and resource reservation facilities.

ACORD-CS is an architecture for the sharing of context information and components. ACORD-CS implements all the elements of the metamodel previously described. The architecture defines a context-aware component model and a container for supporting components. ACORD-CS also supplies an inference engine to support adaptation policies, and a set of servers. Three servers were implemented in ACORD-CS: context, component, and quality of service (QoS) servers, as shown by Fig. 3. The elements of ACORD-CS are described in the sequence.

Context-aware Container

A context-aware container is a run-time environment that offers monitoring and adaptation services to the components. Fig. 4 shows an

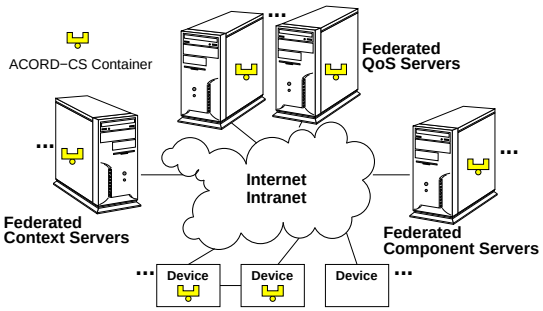

Figure 3: ACORD-CS architecture.

ACORD-CS container, which represents the *CtxContainer* of the proposed metamodel.

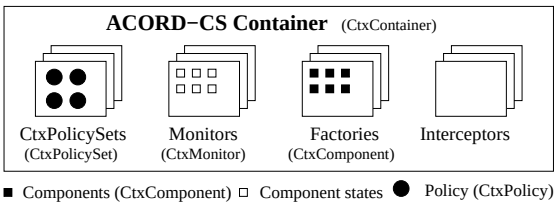

Figure 4: ACORD-CS container.

The container provides a run-time environment for component factories, policy sets, and monitors. Component factories provide methods for creating, destroying, and searching for components. Component factories are installed and removed dynamically. As such, the type of components a container supports may vary over time. The ability to dynamically install new types of components is an essential requirement for building true context-aware and ubiquitous applications. Monitors feed policy sets with contextual information they obtain by interacting with the environment and servers. Policy sets are composed of policies that are evaluated every time the system state changes (e.g., by the creation or destruction of components, or by component parameter adjustment). The element *CtxPolicySet* represents an inference engine in ACORD-CS. In the case of a policy condition match, the corresponding action is executed. Actions produce behavioral and structural adaptation over the application. Finally, interceptors are hooks that allow pre and post processing in method calls for purposes of logging, security, and resource management.

Context Server

Context servers are responsible for gathering, compiling and distributing context information. A context server keeps also information about the resources and services available in a domain. Context-aware containers provide access interfaces to this server in order to allow monitors and other elements to interact with it. Interaction can

be a query or a subscription for further notification. For instance, a monitor can subscribe to receive updated context information, such as new resources that are incorporated to the environment. Context servers can also employ policies to control the way context information is propagated to the subscribed parties.

Component Server

Component servers act as a repository of factories for managing context-aware components. It provides functionalities for searching and storing component factories. Both context and component servers can be grouped into federations of collaborating servers.

QoS Server

Since components can feature stream ports, such as video and audio ports, a mechanism for resource reservation to assure quality of service for media flows are necessary. The QoS server (Pinto et al., 2003) accepts requests for the reservation of network resources, such as bandwidth and traffic priority. This server has the same functionality as the Differentiated Service (DiffServ) Bandwidth Broker.

4 Application in Mobile Robotics

A context-aware application in the field of mobile robotics was developed in compliance with ACORD-CS. The application consists of an environment in which mobile robots are allowed to navigate. The environment has as resources an Axis 213-PTZ panoramic camera and a 802.11b/g wireless network access point. A Pioneer P3-DX mobile robot from ActivMedia[1] equipped with an on-board processor, sonars, a Canon VC-C4 PTZ camera and 802.11b wireless network interface, was employed in the experiment. All the servers (context and component) and the robot runs GNU/Linux as operating system. Fig. 5 illustrates the environment.

The robot can be remotely controlled by an operator, using the operator's console, which allows to control the movements of the robot and the cameras, and to receive and save the images captured by the on-board and panoramic cameras. Fig. 6 shows the operator's console.

The environment provides a Context Server, which points to services, such as panoramic cameras, navigation beacons (bluetooth antennas, bar code marks), and communication devices (WiFi access points).

An ACORD-CS container (RobotContainer) runs on the robot's on-board processor. In this container is instantiated a monitor that senses the

[1] http://www.mobilerobots.com

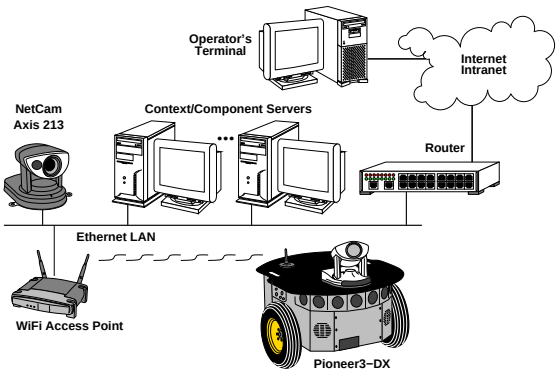

Figure 5: Infrastructure for the application.

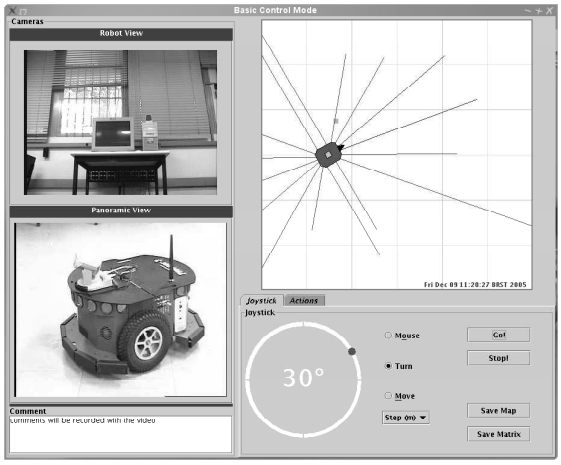

Figure 6: Operator's console.

wireless access points and their signal strengths. Three components are initially instantiated in the container: a video producer that captures video from the on-board camera, a tele-operation server component with methods like *move*, *turn*, etc., and a navigator component that implements the potential field algorithm for autonomous navigation.

An ACORD-CS container (ClientContainer) is also installed on the operator's terminal. A monitor in this container accesses the Context Server to locate the available robots and panoramic cameras present in the environment. Four components are installed on this container: two video consumer components that present video from the on-board and panoramic cameras, a tele-operation client component that allows the operator to move the robot manually, and a navigator component identical to that installed on the robot's container. Fig. 7 shows the initial configuration. The CtxPolicySet acquires the mission from the operator (target point and navigation parameters) and stores on the navigators components (*Mission Assignment* in Fig. 7).

The CtxPolicySet installed on the robot's container performs behavioral and structural adaptation functions based on the signal strength measured in its wireless interface (Linux utility *iwlist*

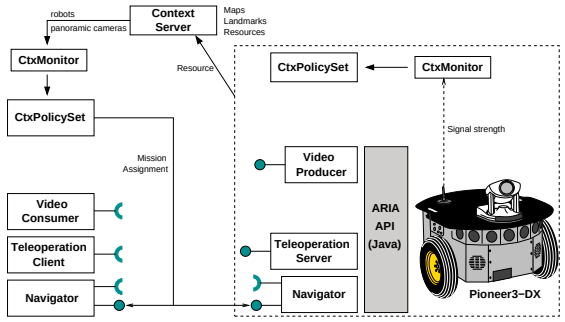

Figure 7: Initial Configuration.

and *iwconfig* are employed to sense the signal strength). The actual signal strength defines three contexts:

Full connectivity context - In this context, the robot is controlled by the components running on operator's terminal (tele-operation or navigator). Fig. 8 shows the structure of the application in this context.

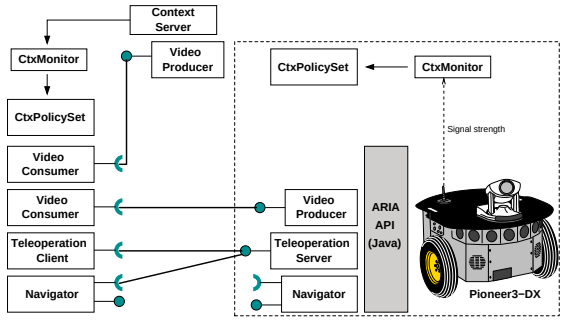

Figure 8: Full connectivity context.

Deficient connectivity context - In this context, the robot is still controlled by the components running on operator's terminal (teleoperation or navigator), but the video framerate of the on-board camera decreases (a behavioral adaptation). In this context, a structural adaptation also takes place with the download of a video consumer component from the component server. This component stores video frames on a file instead of presenting them. This component is not connected yet in this context. Fig. 9 shows the structure of the application in this context.

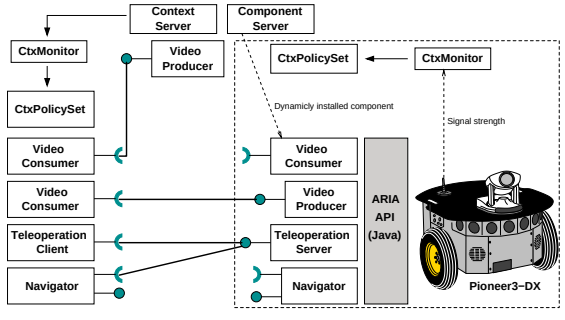

Figure 9: Deficient connectivity context.

Lack of connectivity context - In this context, the robot is disconnected from the operator's terminal. The navigator component on the robot's container assumes the navigation. The video consumer component downloaded previously is connected to the video producer in order to record the video captured by the on-board camera for a posteriori exhibition. Fig. 10 shows the structure of the application in this context (a strong structural adaptation took place).

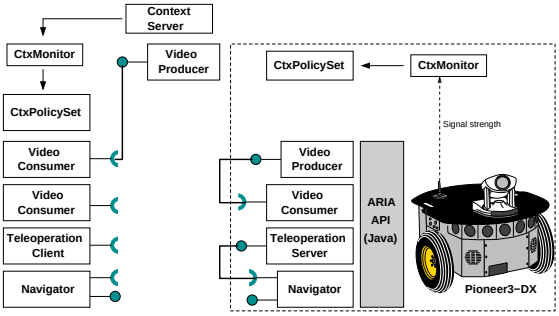

Figure 10: Lack of connectivity context.

Evaluation

All the implementation was written in the Java language. To deal with new component (and factory) types, these are instantiated using the native Java language reflection API. CtxPolicySet counts with the JESS (Friedman-Hill, 2003) inference engine, and all the policies are written in the JESS language. Considering this, the size and instantiation times are quite small, as shown by Tables 1 and 2 .

Node	Size (KB)
Robot	˜864
Client	˜824
Context/Component Server	˜820

Table 1: Installation size of components in each node.

Operations	Time
VideoConsumer XML search download and parse	˜142ms
VideoConsumer.jar (˜3.2KB) download and install	˜164ms
VideoConsumer instantiation	˜11ms
RobotClient XML search download and parse	˜60ms
RobotClient.jar (˜11KB) download and install	˜755ms

Table 2: Average time measurements over 20 runnings.

The VideoConsumer XML (Table 2) is the component description stored in the component

server. It is searched by the RobotContainer when it detects a *Deficient connectivity*, and by the ClientContainer when it detects a panoramic and an on-board camera. The information contained therein is used to download and instantiate the VideoConsumer at the RobotContainer and ClientContainer. The VideoConsumer is packaged in a jar file at the component server. Similarly, RobotClient XML is the description of the components (Tele-operation Client and Navigator) used to interact with the robot. When the client container detects a robot presence, it searches the robot description file, downloads the RobotClient jar file and instantiates the necessary components. The experiment showed that the download and instantiation times are small, proving its feasibility in an ever changing and highly dynamic environment.

5 Conclusion

The process of writing context-aware applications is rather difficult and error prone. Delegating the task of gathering, processing contextual information, and adapt, to the application programmer is too overwhelming. The use of context-aware software components and a metamodel that targets context-awareness, relief the developer from this responsibility. This is achieved by delegating some responsibilities to the container in the form of non-functional requirements, such as (re)connection of ports, system state monitoring (quality of port connections), and context gathering and processing. Also, the adaptation rules are separated from the main application logic (CtxPolicySet). The servers offered by ACORD-CS enable the discovery and use of context information and software components, not present at startup time, by the application containers, allowing the use of different resources encountered during the application execution, also the quality of service offered to audio and video flows are guaranteed by network resource reservation featured in the ACORD-CS QoS server. We are currently working on a better way to represent policies in a high-level and technology independent language, and on the validation of our metamodel using other component models other than ACORD-CS.

References

Act (2002). "OMG Unified Modeling Language Specification (Action Semantics)". http://www.omg.org/docs/ptc/02-01-09.pdf.

A.Ketfi and N.Belkhatir (2004). "A Metamodel-based approach for the dynamic reconfiguration of component-based software", *ICSR* . http://www-adele.imag.fr/Les.Publications/intConferences/-ICSR2004.pdf.

Ayed, D. and Berbers, Y. (2006). "UML Profile for the Design of a Platform-Independent Context-Aware Applications.", *MODDM'06.*, ACM, Melbourne, Australia.

Friedman-Hill, E. (2003). *"Jess in Action: Rule-Based Systems in Java"*, Manning.

Guimarães, E. G., Maffeis, A. T., Pereira, J. L., Russo, B. G., Cardozo, E., Bergerman, M. and Magalhães, M. F. (2003). "REAL: A Virtual Laboratory for Mobile Robot Experiments", *IEEE Transactions on Education* **46**(1): 37–42.

Guimarães, E. G., Maffeis, A. T., Pinto, R. P., Miglinski, C. A., Cardozo, E., Bergerman, M. and Magalhães, M. F. (2003). "REAL: A Virtual Laboratory Built from Software Components", *Proceedings of the IEEE* **91**(3): 440–448.

ISO/IEC 10746-2 / ITU-T Rec. X.902 (n.d.). *"ODP Reference Model Part 2, Foundations"*, International Organization for Standardization and International Electrotechnical Committee".

OMG (2004). *Unified Modeling Language*, http://www.uml.org/. Last access in 27/09/2005.

Pinto, R. P., Cardozo, E. and Guimarães, E. G. (2005). "Um Modelo de Componentes para Aplicações Sensíveis a Contexto", *V Workshop de Desenvolvimento Baseado em Componentes (WDBC'05)*, SBC, Juiz de Fora, MG.

Pinto, R. P., Guimarães, E. G., Cardozo, E. and Magalhães, M. F. (2003). "Incorporação de Qualidade de Serviço em Aplicações Telemáticas", *21 Simpósio Brasileiro de Redes de Computadores (SBRC'03)*, SBC, Natal, RN.

Sheng, Q. Z. and Beatallah, B. (2005). "ContextUML: A UML-Based Modeling Language for Model-Driven Development of Context-Aware Web Services", *ICMB* .

Warmer, J. and Kleppe, A. (2003). *"The Object Constraint Language"*, second edn, Addison Wesley.

Xio (2002). *"Xion Action Language"*, http://lglpc35.epfl.ch/objx/netsilon/xion/-index.html. Last access in 27/09/2005.