A Domain-independent Middleware Framework for Context-aware Applications

Rossano P. Pinto
Eleri Cardozo
Paulo R.S.L. Coelho
Faculty of Electrical and Computer Engineering
State University of Campinas
Campinas, Brazil 13083-970
rossano@dca.fee.unicamp.br

Eliane G. Guimarães
CenPRA - Renato Archer Research Center
Campinas, Brazil 13089-970
eliane.guimaraes@cenpra.gov.br

ABSTRACT

This paper presents a middleware framework for context-aware applications. Instead of trying to incorporate into the middleware a set of domain-dependent adaptive policies, our approach allows the application itself to define such policies according to a chosen adaptive behavior. The middleware defines a domain-independent metamodel for context-aware applications. For applications specified according to the metamodel, the framework is able to generate code from the applications' design. We demonstrate the use of the middleware with a context-aware application in the field of mobile robotics.

Categories and Subject Descriptors

D.2.11 [**Software Engineering**]: Software Architectures; D.2.13 [**Software Engineering**]: Reusable Software

General Terms

Experimentation, Design

Keywords

Model Driven Engineering, UML Profile, Context-Awareness

1. INTRODUCTION

A context-aware application is able to adapt its behavior when the environment surrounding the application changes. The adaptation process can take place on the application's logic, on the infrastructure supporting the application (middleware), or on both. Adaptive middlewares are a natural choice for supporting context-aware applications. By leaving some adaptive behavior to the middleware, the application is free from implementing such behaviors. Examples of adaptive functions that can be left to the middleware include adaptations on communication [1], on dissemination of contextual information [2], and on service discovery [3].

As adaptive behaviors are highly application-dependent, adaptive middlewares supporting context-aware applications must be highly customizable in terms of the adaptation policies the middleware employs; how contextual information is gathered, transformed, and fused; and how the application is notified about the changes in the environment.

This paper presents ACORD-CS, a domain-independent middleware framework for context-aware applications. ACORD-CS defines a metamodel for context-aware applications, a set of middleware services, and a code generation engine. Instead of imposing a restrict set of adaptation behaviors, ACORD-CS lets the application to define such behaviors as well as to reuse some existing domain-specific behaviors.

There are several proposals for incorporating context-awareness to the software applications: conceptual models [4], frameworks [5], and middlewares [6]. The adaptation process is usually based on policies, but utility functions [7], aspect oriented programming (AOP) [8], and design patterns [9], among other approaches, were proposed as well. The contribution of ACORD-CS is to integrate on a middleware framework a set of strategies reported on the literature as the most promising for developing context-aware applications. Metamodeling, component-based development, code generation from design, and policy-based adaptation are the strategies integrated by the ACORD-CS middleware framework.

The paper is structured as follows. Section 2 describes the ACORD-CS metamodel for context-aware applications. Section 3 presents the ACORD-CS middleware framework. Section 4 presents a context-aware application built over ACORD-CS. Section 5 presents the related work addressing the same issues presented in this paper. Finally, Section 6 closes the paper with concluding remarks.

2. A METAMODEL FOR CONTEXT-AWARE APPLICATIONS

The proposed metamodel for context-aware applications is expressed in UML (Unified Modeling Language) [10] through a class diagram shown in Figure 1. The metamodel defines seven UML classes and their relationships that represent the main entities of a context-aware application. *CtxComponent* and *CtxContainer* are concepts that represent components

ARM'07, November 26, 2007 - Newport Beach, CA

and containers as defined by the existing component models. As such, components are units of reuse, 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.

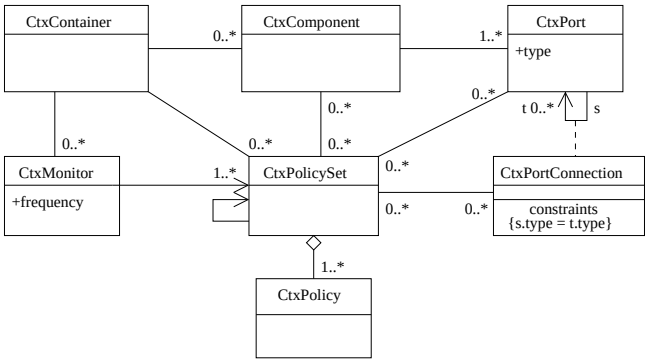

Figure 1: Metamodel for context-aware applications.

CtxPort represents endpoints for interactions (ports) among components. A port can support different interaction semantics, such as those prescribed by RM-ODP (Reference Model for Open Distributed Processing) [11]. For instance, the CM-tel component model supports synchronous (request-reply), asynchronous (notification) or stream (flow-based) ports [12].

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 an important role in this metamodel. It represents a set of adaptation policies that trigger adaptation actions based on acquired contextual information. The relationship between *CtxPolicySet* and the remaining elements 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 properties), a port (e.g., by setting a port output format), and a port connection (e.g., by changing its QoS parameters). The model does not enforce any mechanism by which adaptation is performed. Behavioral (parametric) adaptation is usually accomplished via updating of component and port configuration properties or by calling methods defined 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 policy set is composed of one or more adaptation policies represented by the *CtxPolicy* element.

A *CtxMonitor* is responsible for acquiring contextual information from any source and for feeding policy sets with this information. Contextual information may come from many sources, for instance, sensor readings, service invocation, or user inputs. The attribute *frequency* indicates the frequency in which the contextual information is acquired. *CtxMonitor* usually preprocesses the information by performing aggregation, filtering, and scaling.

This metamodel is defined through a UML Profile [10], a native UML extension mechanism. UML Profiles are composed of stereotypes marking a particular UML element, such as classes, associations, and attributes. This mark can be used for several purposes, including hints for automatic code generation. UML profiles can be enhanced by the adding of *tagged-values* to stereotypes. Each tag is associated with a single value chosen from a set of permitted values.

For each element of the model depicted in Figure 1 a stereotype with the same name is defined. The metamodel defines a tag related to rules expressing adaptation policies. The tag *Rule* is associated with the stereotype *CtxPolicy* and defines a rule associated with the policy. The UML Profile does not define how rules are modeled since there is no rule syntax universally accepted.

3. THE ACORD-CS MIDDLEWARE

In order to support context-aware applications complying to the metamodel stated in Figure 1, a middleware was defined in line with this metamodel. The middleware consists of a context-aware container and a set of services for supporting the gathering, storing, and distribution of contextual information.

Context-aware Container

A context-aware container offers monitoring and adaptation services to the components. Figure 2 shows an ACORD-CS container, which represents the *CtxContainer* of the proposed metamodel.

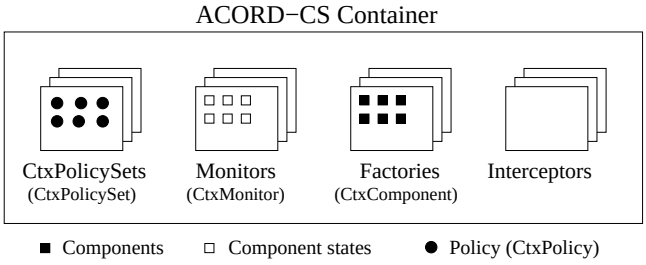

Figure 2: ACORD-CS container.

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 services. 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 when a component property changes). In the case of one of the rules composing the policy has its condition satisfied, the corresponding action is executed. Actions produce behavioral and structural adaptations over the application. Finally, interceptors are hooks that allow pre- and post-processing in method calls for purposes of logging, security, and resource management.

Facilities for Application Building

ACORD-CS offers two facilities for application building, a UML design tool and a code generation engine. The UML design tool is BOUML [13] extended to incorporate the elements defined by the ACORD-CS metamodel. A tab was added to the BOUML's class dialog that allows to choose the proper ACORD-CS model elements according to the class' stereotype.

The code generation engine is based on XSL (XML Stylesheet Language) transformations. An XSL transformation receives as input the UML diagram of the application formatted according to XMI (XML Model Interchange) and an XSL document specifying the transformation. XMI files are generated by BOUML from the class diagrams. Transformations produce software artifacts such as Java classes, shell scripts, or text files (e.g., a file containing a set of rules specified in a rule-based language).

The code generation engine is dependent of technology. The current version of ACORD-CS adopts a lightweight, Java-based component model able to execute on small processors. The lightweight component model does not need extensive run time systems as demanded by the commercial component models such as Enterprise Java Beans. This model captures only the key elements of component models such as a factory design pattern to control the life-cycle of components, a mechanism to compose components, and persistence of the component state. Components inherit from *CtxComponent*. This base class provides methods for composing components (connecting their ports) and for saving and restoring the state of components (important for component replacement). *CtxPortConnection* is realized through a component connector. Currently, a connector based on RMI (Java Remote Method Invocation) is supplied by ACORD-CS.

Policy sets are composed of policies that are evaluated in response to events monitored by the application. A policy is modeled as a set of rules (rulesets). If a rule condition matches, the corresponding actions are executed. Actions produce behavioral and structural adaptation of the application. ACORD-CS has a general strategy to handle policies as rulesets. Classes stereotyped with *CtxPolicy* employ UML tagged-values to express the rules. Tags carry the rule name and tag values hold the rules written on a chosen syntax. Rule conditions and actions must refer to attributes present on the remaining model elements. These attributes are updated as the rules fire. ACORD-CS defines a simple interface to rule engines. This interface has four methods:

- *defrule(rule, ruleset)*: adds or updates a rule in a ruleset;
- *set(variable, value)*: asserts a value for a variable in the rule engine. The variable must be an attribute of a model element;
- *fire(ruleset)*: fires the rules in a ruleset;
- *get(variable)*: gets the current value of a variable.

ACORD-CS provides this interface for the Java Expert System Shell (JESS) and for a fuzzy engine implemented by E. Sazonov [14]. We found fuzzy rules more intuitive for expressing adaptation policies. Policies are expressed in near natural language such as *"if ControlRate is small and TrajectoryAngle is narrow then RobotSpeed is slow"*. Moreover, fuzzy engines are faster than expert system engines and more tolerant to inaccuracies on the input data [15]. This former property is important as contextual information can be conflicting and inaccurate. ACORD-CS checks if the linguistic variables that appear on the rules (e.g., *ControlRate*) are attributes of the model elements.

Context Providers

Context providers are responsible for gathering, compiling, storing, and distributing contextual information. Since resources and services available in a domain are considered contextual information, the context providers are responsible for maintaining such information. Context-aware containers offer access interfaces to context providers in order to allow monitors and other elements to interact with them. Interaction can be a query or a subscription for further asynchronous notification. For instance, a monitor can subscribe to receive updated contextual information, such as new resources that are incorporated into the environment. Context providers may employ policies to control the way context information is propagated to the subscribed parties.

ACORD-CS supplies a context provider that offers a publish/subscribe service for contextual information processing. Publishers invoke the context provider to submit an XML document containing contextual information. A subscriber registers to the service supplying its notification interfaces and, optionally, an XPath expression for selecting the contents or publisher of contextual information.

Component Providers

Component providers act as a repository of factories for managing context-aware components. Component providers offer functionalities for searching and storing component factories. ACORD-CS supplies a component provider similar to the context provider. The component provider stores component description searched via XPath expressions. The component factories themselves are stored on HTTP servers accessible from the application's environment.

Supporting Services

Supporting services help the context-aware containers to support non-functional requirements. Examples of supporting services include security, transaction, and QoS services. As an example of integration of supporting services, a *CtxPortConnection* element may negotiate to a bandwidth broker certain class of service to the media ports the element connects.

4. A CONTEXT-AWARE APPLICATION

WebLabs allow laboratorial infrastructures be manipulated through the internet. When an experiment is performed in real time (e.g., the control of a mobile robot), several contextual variables must be taken into account, for instance, the speed and bandwidth of the network the user is connected to, the processing power of the user's machine, the number of users performing the experiment concurrently, and so on.

A mobile robotic WebLab developed by the authors is reported in [16]. Figure 3 shows the WebLab console with the on-board and panoramic camera view (left), navigation map (top right), and teleoperation controls (bottom right).

One of the experiments in the WebLab is vision-based robot navigation where a mobile robot must follow a colored stripe on the floor using solely its on-board camera.

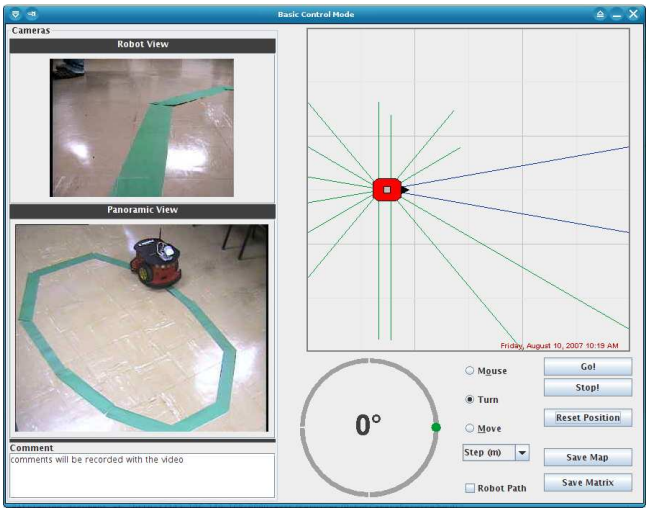

Figure 3: WebLab console.

In this experiment a control software running on the user's computer performs the following cycle:

1. invoke the robot's vision service to acquire an image from the on-board camera;
2. call an image processing algorithm to identify the stripe. If no stripe is identified do: (i) invoke the robot's locomotion service to stop the robot; (ii) sleep some amount of time and go to step 1 (hoping some external entity acted on the robot);
3. invoke an heuristic to compute a turn movement that will position the stripe in the center of the image;
4. invoke the robot's locomotion service to set a constant translational speed and to perform the turn movement;
5. go to step 1.

A very sensitive control parameter is the speed the robot moves. The maximum speed that the robot can follow the stripe depends on the complexity of the path (angles between segments) and the frequency of the control cycles. This frequency depends on two factors:

1. the speed of the network the user is connected to;
2. the processing power of the user's computer.

The first factor impacts on the delay between the acquisition of the control input (image) and the execution of a control action (robot movement). The second factor impacts on the amount of time necessary to compute a control action (as image processing is involved).

The navigation-based experiment as reported in [16] requires the user to set the robot's speed. If the user chooses a high speed, the robot looses the stripe and stops waiting for user interference. On the contrary, with a low speed the robot takes a long time to complete the trajectory. In order to adjust the robot's speed autonomously and to limit the consumption of network resources by the experiment, a context-aware application was deployed on the WebLab's side. Figure 4 shows the main components of the application modeled according to the UML profile of Figure 1. A single policy rules is shown as an anchor note. Darker model elements belong to the previous (not context-aware)

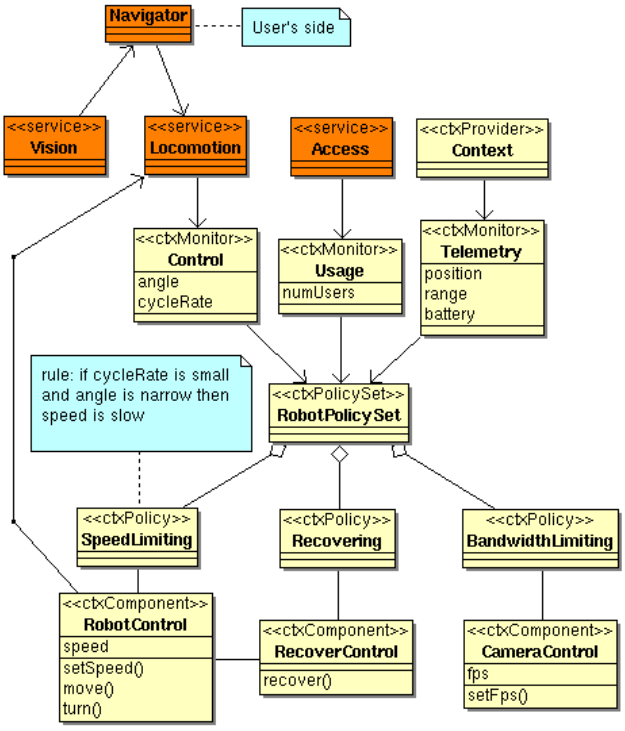

Figure 4: The components of the context-aware application modeled according to the UML Profile.

implementation [16]. Except for the *Navigator* element, the remaining elements are deployed at the WebLab's side.

This context-aware application employs monitors and policy sets according to ACORD-CS metamodel. The monitors acquire data from a context provider and from two supporting services, the robot's locomotion service controlling the robot and an access service enforcing usage upon reservation and authentication. Three monitors were implemented:

- Control monitor: intercepts invocations to the locomotion service and extracts the parameters of the control action performed (robot's orientation and speed);
- Telemetry monitor: subscribes to the context provider to receive the information about the robot's telemetry (position, sonar range, and battery voltage);
- Usage monitor: polls the access service for information about the number of users performing the experiment.

Three adaptation policies were implemented:

- Speed limiting policy: limits the robot's speed preventing it from loosing the stripe;
- Recovering policy: recovers, without user interference, when the robot looses the stripe;
- Bandwidth limiting policy: limits the bandwidth demanded by the panoramic camera in order to save bandwidth for the robot control.

Speed limiting policies are implemented with 12 fuzzy rules of the form *"if cycleRate is small and angle is narrow*

then speed is slow". The rule inputs are the rate (frequency) of the control actions and the angle of the trajectory (given the turn movement computed by the control). Control cycle rate and trajectory angle are supplied by the control monitor. These rules are fired as soon as a control action is performed by the user's algorithm.

Recovering policy consists of a single rule that call a recovery function when the control sets the robot's speed to zero (step 2(ii) of the control cycle). This rule acts on a component that performs recovering actions such as to move the robot backwards, to turn the robot some angle, and to limit the maximum speed. After the recovering action was taken, the policy waits an amount of time. If the control continues to set speed equal to zero, the recovering action is repeated for 3 times before user interference is requested. Otherwise, if the speed is set to a value greater than zero, the recovering action succeeds as the control was able to identify the stripe again.

Bandwidth limiting policies are implemented with 9 fuzzy rules of the form *"if cycleRate is small and numUsers is low then fps is medium"*. These rules fire periodically and adjust the panoramic camera's framerate (fps) to avoid excessive bandwidth consumption that can degrade the control. The input of the rules are the control cycle rate and the number of users accessing the experiment.

Results

The experiment was conducted on three different access networks: a residential internet access of 2 Mbit/s, a campus network, and a local area network (LAN) directly connected to the WebLab. The user's computer was a Dell D520 notebook with 2GB of RAM memory. Table 1 summarizes the results for different access networks. Speed is the average speed in mm/s. Time is the total time to complete the trajectory in seconds. Recov is the number of recovering actions performed during the experiment. Fps is the average frames per second set on the panoramic camera.

Access	Cycles/s	Speed	Time	Recov	Fps
Residential	0.90	69	124	4	4
Campus	1.46	127	99	12	12
LAN	1.72	143	84	8	12

Table 1: Summarized results.

The results show the effectiveness of the adaptation policies. As the speed of the network increases the adaptation policies improves the performance of the experiment (time to complete the trajectory) and the quality of the video supplied by the network camera. Without this adaptation capability, the experiment would produce results dependent of the user's context (accessing network and computer) and robot's context (complexity of the trajectory). It is nearly impossible to estimate a correct robot speed for a given point in the trajectory without taken contextual information into account.

5. RELATED WORK

The literature reports many proposals in context-aware systems that share similar concepts to those addressed in this paper. WildCAT [5] is a framework for context-aware applications written in Java. WildCAT models contextual information as a tree of resources that resembles the Unix filesystem. Contextual information is selected via logical predicates (expressions) that selects elements on the tree. ACORD-CS employs XML for representing contextual information and XPath to select pieces of information. Both WildCAT and ACORD-CS employ notification based on the publish-subscriber pattern.

Khan et al [17] uses a Model Driven Architecture (MDA)-based development approach to build self-adaptive software systems. This work is related to ACORD-CS in the sense that both employs model transformations for code generation. The tools and modeling technology are also similar. The UML application model is converted to XMI (XML Metadata Interchange), an XML dialect to express UML models. Then, an XSLT (XML Stylesheet Language Transformation) is applied in order to transform the model expressed in XMI to Java code.

Chauvel and Barais [18] present a policy-based model for self-adapting component architectures. The work addresses both architectural (structural) and behavioral adaptations as ACORD-CS does, but no implementation of this model was reported.

Kephart and Das [7] propose a self-management strategy based on policies. The authors classify policies into three categories: action policies, goal policies, and utility function policies. Action and goal policies dictate, respectively, the action and the goal that the system should take if the policy is applied. Utility policies choose the next system state based on a utility (goodness) function. The authors claim that utility function policies are best suitable for achieving self-management in autonomic systems. ACORD-CS employs action policies as we believe that complex context-aware applications are not best modeled in terms of states and state transitions.

ContextUML [4] proposes a metamodel for context-aware Web services. The metamodel specifies sources of contextual information (context sources), conditions and actions (context triggering), and context-aware Web services (context binding). Context sources are similar to context monitors in the ACORD-CS metamodel, but without the limitation of Web services as the only source of contextual information. Conditions and actions in ContextUML are specified in OCL (Object Constraint Language) [19]. We believe that OCL is not suitable for defining adaptation strategies since OCL is a notation for enhancing the expressiveness of UML model, not to model adaptation actions over the implemented model. Although OCL extensions such as Action Semantics [20] allows to model adaptation actions, we believe rules have a syntax more appropriated to model adaptation actions. While ContextUML ties the metamodel to the Web services technology, we prefer to leave the metamodel free from details assigned to a particular technology, an approach in line with MDA.

6. CONCLUSIONS

The process of writing context-aware applications is difficult and error prone. This paper presented a strategy for reducing the complexity of such applications centered on three elements. The first element is a UML Profile for context-aware applications. UML Profiles contribute for a more systematic design of context-aware applications. The second element is the modeling of context-awareness through policies. Policies contribute for the separation of the adaptation process from the functional logic of the application

(separation of concerns). Adaptation is a key attribute of context-aware applications. The third element is ACORD-CS, a middleware framework for supporting context-aware applications. By adhering to a UML Profile, it is possible to generate a portion of code for the components of the application adhering to the UML Profile. Such code includes Java classes, policy rules, shell scripts, and configuration files.

ACORD-CS is a domain independent middleware framework that can be customized for a particular domain through the addition of domain-specific adaptive policies. For instance, the policies employed to limit the speed of the robot according to the rate of control actions and the complexity of the trajectory can be incorporated into an adaptive middleware for mobile robotics applications.

Finally, an application in the field of mobile robotics illustrated the functionalities of ACORD-CS.

Acknowledgments

This research is supported by the Brazilian National Education and Research Network (RNP) under the Giga Project 2463. Paulo R.S.L. Coelho is a Ph.D. student supported by CAPES. The authors acknowledge the reviewers for the useful comments, critics, and suggestions.

7. REFERENCES

[1] Raja Afandi, Jianqing Zhang, and Carl A. Gunter. AMPol-Q: Adaptive Middleware Policy to Support QoS. In *4th International Conference on Service Oriented Computing (ICSOC'06)*, Chicago, USA, 2006.

[2] Markus Huebscher and Julie McCann. Adaptive Middleware For Context-Aware Applications. *Personal and Ubiquitous Computing Journal September*, 2006. Springer.

[3] Carlos A. Flores-Cortés, Gordon S. Blair, and Paul Grace. An Adaptive Middleware to Overcome Service Discovery Heterogeneity in Mobile Ad Hoc Environments. *IEEE DS Online Exclusive Content Middleware*, 2007.

[4] Quan Z. Sheng and Boualem Beatallah. ContextUML: A UML-Based Modeling Language for Model-Driven Development of Context-Aware Web Services. *Proceedings of the International Conference on Mobile Business - ICMB*, 2005.

[5] Pierre-Charles David and Thomas Ledoux. WildCAT: a Generic Framework for Context-aware Applications. In *International Workshop on Middleware for Pervasive and Ad-Hoc Computing*, Grenoble, France, 2005.

[6] Carl-Fredrik Sørensen, Maomao Wu, Thirunavukkarasu Sivaharan, Gordon S. Blair, Paul Okanda, Adrian Friday, and Hector Duran-Limon. A Context-aware Middleware for Applications in Mobile Ad Hoc Environments. In *Proceedings of the 2nd Workshop on Middleware for Pervasive and Ad-hoc Computing*, Toronto, Canada, 2004.

[7] Jeffrey O. Kephart and Rajarshi Das. Achieving Self-Management via Utility Functions. *IEEE Internet Computing*, 11(1), 2007.

[8] Francisco Dantas, Thais Batista, Nelio Cacho, and Alessandro Garcia. Towards Aspect-Oriented Programming for Context-Aware Systems: A Comparative Study. In *Proceedings of the 1st International Workshop on Software Engineering for Pervasive Computing Applications, Systems, and Environments*, Mineapolis, USA, 2007.

[9] Gustavo Rossi, Silvia Godillo, and Fernando Lyardet. Design Patterns for Context-Aware Adaptations. In *Proceedings of the 2005 Symposium on Applications and the Internet Workshops*, Trento, Italy, 2005.

[10] *Unified Modeling Language*. `http://www.uml.org/`.

[11] ISO/IEC 10746-2 / ITU-T Rec. X.902. *ODP Reference Model Part 2, Foundations*. International Organization for Standardization and International Electrotechnical Committee, June 1995.

[12] Eiane G. Guimarães, Antonio T. Maffeis, Rossano P. Pinto, Carlos A. Miglinski, Eleri Cardozo, Marcel Bergerman, and Mauricio F. Magalhães. REAL: A Virtual Laboratory Built from Software Components. *Proceedings of the IEEE*, 91(3):440–448, March 2003.

[13] Bouml home page. `http://bouml.free.fr`.

[14] Edward Sazanov. *Fuzzy Engine for Java*. `http://people.clarkson.edu/~esazonov/`.

[15] Witold Pedrycz and Fernando Gomide. *Fuzzy Systems Engineering: Toward Human-Centric Computing*. Wiley-IEEE Press, 2007.

[16] Paulo R. Coelho, Rodrigo F. Sassi, Eleri Cardozo, Eliane G. Guimarães, Luis F. Faina, Rossano P. Pinto, and Alex Z. Lima. A Web Lab for Mobile Robotics Education. In *IEEE International Conference on Robotics and Automation - ICRA'07*, Rome, Italy, 2007.

[17] Mohammad U. Khan, Roland Reichle, and Kurt Geihs. Applying Architectural Constraints in the Modeling of Self-adapting Component-based Applications. In *1st Workshop on Model-driven Software Adaptation (M-ADAPT)*, Berlin, Germany, 2007.

[18] Franck Chauvel and Olivier Barais. Modeling Adaptation Policies for Self-Adaptive Component Architectures. In *1st Workshop on Model-driven Software Adaptation (M-ADAPT)*, Berlin, Germany, 2007.

[19] Jos Warmer and Anneke Kleppe. *The Object Constraint Language*. Addison Wesley, second edition, 2003.

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