A Component Framework for Context-Awareness

Rossano Pablo Pinto, Eleri Cardozo
School of Electrical and
Computer Engineering
State University of Campinas
Campinas, SP - Brazil
Email: {rossano,eleri}@dca.fee.unicamp.br

Eliane G. Guimarães
CenPRA - Renato Archer Research Center
Campinas, SP - Brazil
Email: eliane@dca.fee.unicamp.br

***Abstract*— The omnipresence of mobile networks and the multitude of resources connected to these networks motivate the development of applications able to mold themselves to the current context. Context-aware applications are autonomous to adapt their behavior according to the environment in which they execute. This behavior is achieved through an adaptation process that takes into account contextual information such as capabilities of the accessing terminal and network, user profile and preferences, current application's goals, and information about the surrounding environment (e.g., geographical location, nearby resources, and available services). Context-awareness is a key issue for the design and implementation of truly adaptive applications. However, due to the absence of de facto supporting models and platforms, 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 shortly introduces a metamodel for context-aware applications and a supporting platform for this class of applications. The platform offers features to discover and publish available services using both decentralized and centralized mechanisms.**

I. Introduction

In order to benefit from the current context, context-aware applications must (1) sense the environment into which they are executing, (2) infer their execution context, and (3) perform adaptive actions. These applications must be able to interact with the resources present in the environment in the same way they interact with their own resources. Information about the environment and application can be obtained via context-awareness monitoring mechanisms. Notably, software components, a well established technology for building distributed and highly modular applications, does not consider context-awareness. For instance, commercial component models such as Enterprise Java Beans (EJB) and COM+ (Component Object Model) support non-functional requirements in the business domain (transaction, persistence, and security), but requirements assigned to context-aware computing are not supported by these component models. As a result, context-awareness is incorporated into the application as a functional or non-functional requirement to be fulfilled by the developer, not by the underlying component infrastructure (platform). We argue that mechanisms for supporting context-awareness can be delegated to the supporting platform. In component-based systems, such mechanisms can be provided by a context-aware container.

This paper shortly introduces a metamodel for the development of context-aware applications built from software components [1]. The metamodel identifies the main elements of a context-aware application. ACORD-CS, a supporting software platform for context-aware applications in line with the proposed metamodel is presented in the sequence. In particular, the mechanism for component or service discovery and publishing is presented in detail.

This paper is structured as follows. Section II introduces the metamodel for context-aware applications. Section III presents ACORD-CS. Finally, Section IV presents some conclusions, related works, and final remarks.

II. A Metamodel for Context-aware Applications

The proposed metamodel [2] for context-aware applications is expressed in UML (Unified Modeling Language) [3] through a class diagram shown in Fig. 1.

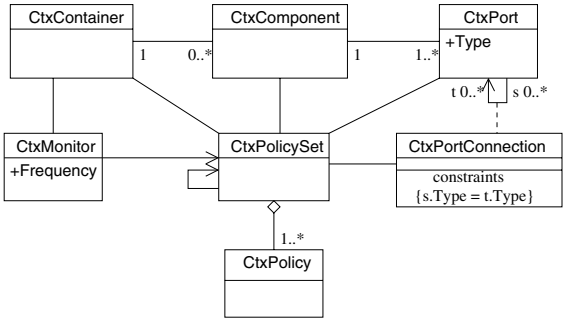

Fig. 1. Metamodel for context-aware applications.

The metamodel defines seven UML classes and their relationships that represent the main entities of a component-based and 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 (context-awareness mechanisms in this case) and the run-time environment, while components supply the functional requirements of the application.

978-1-4244-2202-9/08/$25.00

CtxPort represents component's endpoints for interactions (ports). Only ports of the same *Type* and complementary roles can interact (i.e. be connected). A port can support different interaction semantics such as those prescribed in RM-ODP (Reference Model for Open Distributed Processing) [4]. For instance, the CM-tel [5], [6] component model supports synchronous (request-reply), asynchronous (notification) or stream (flow-based) ports.

CtxPortConnection represents the connection between two complementary ports of the same type, 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, and can be used to monitor and update the connection parameters.

CtxPolicySet performs a central role in this metamodel. It represents a set of adaptation policies that trigger adaptation actions based on acquired contextual information. The relationship between a *CtxPolicySet* and other entity indicates that an adaptation can take place within that entity - 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 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 *CtxPolicySet* is composed of one or more policies represented by the *CtxPolicy* element. Each *CtxPolicy* owns one or more rules, i.e., conditions that triggers one or more actions 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 system elements such as components, ports, connections, and containers. This element usually preprocesses the information by performing aggregation, filtering, and scaling. The information acquired by this element is exposed by means of its attributes. Each attribute of this element represents a monitored information, which is continuously updated according to a certain implied or specified *frequency*.

This metamodel is defined through a UML Profile [3], a native UML extension mechanism. UML Profiles are composed of stereotypes and tagged-values. Each stereotype is used to mark a particular category of metamodel elements, i.e., classes, attributes, connections, and so on. For each element of the metamodel depicted in Fig. 1, a stereotype with the same name is defined. Each stereotype semantic can be augmented by the definition of tags and corresponding values (tagged-values). Stereotypes and tagged-values are used for several purposes, including hints for code generation.

The UML Profile representing the metamodel includes a high-level language for the definition of adaptation policies. This language is encoded in a special syntax for the values of the tags *Rule*, *Condition*, and *Action* [1].

III. A Platform for Supporting of Context-aware Applications

A context-aware application depends on information about the current context in order to perform adaptive actions. This information can be obtained from a centralized context-server or from active entities (peers) that propagate their presence by means of a known multicast protocol. In order to favor adaptation, this infrastructure, when available, may also provide utility services, such as search and registration of components and resource reservation facilities.

ACORD-CS [7] is a platform for the sharing of context information and components. ACORD-CS implements all the elements of the metamodel previously described. Its architecture defines a context-aware component model that features a container for supporting components. Every component hosted by the container can be announced to the environment. This announcement allows each receiving entity to build its context knowledge.

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. The elements of ACORD-CS are described in the sequence.

Context-aware Containers

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

Each ACORD-CS container announces it's services (components) through Multicast DNS (mDNS) and DNS-Service Discovery (DNS-SD) [8]. This mechanism allows ad-hoc operating mode, but also makes it easy for a centralized entity (context server) to acquire and infer high-level context information. For contextual information that is not possible to acquire through multicasting (e.g., mDNS propagates information only on the local link), a distributed service based on DNS-SD or a centralized service employing a context server can be employed.

Figure 2 shows the structure of the ACORD-CS container, which represents the *CtxContainer* element of the proposed metamodel.

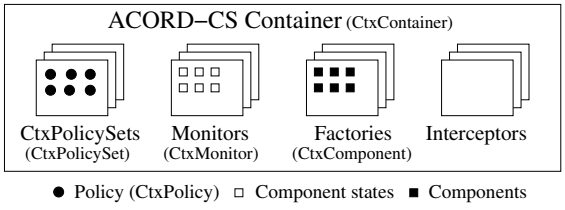

Fig. 2. Structure of the ACORD-CS container.

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 truly context-aware, ubiquitous, and autonomic applications and, at the same time, provides a way of software evolution and maintenance. This ability also allows the incorporation of new functionality at run-time.

Monitors feed policy sets with contextual information they obtain by interacting with the environment (including nearby containers) 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 and connection adjustments). When a policy condition matches, 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. An example of resource management action is to make network bandwidth reservations for the interaction of two connected media streaming ports.

1) Component Installation / Service announcement: Every time a component is activated by a container, the services provided by the component are automatically announced to the environment. Figure 3 shows the steps for component installation.

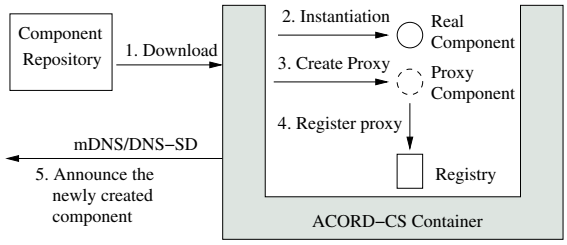

Fig. 3. Component installation and service announcement.

The very first step for component installation is the downloading of the component from a known source (component server or any other entity) (1). This implies the run-time support for unknown component types. Current version of ACORD-CS implements this feature with filesystem jar files installation and Java reflexion API. After the downloading, the component is instantiated (2). This process also creates a proxy for the real component (3) that is registered into a main registry present at every ACORD-CS container (4). After the registration, a user (another component) can find the just registered component and interact with it. A client of the newly instantiated component actually interacts with the proxy. This allows the interception of every interaction between the client and the component, which offers a way to monitor this and change everything (from method call logging to redirections and blocking) that is needed to comply with policies matching current context. According to a policy, the information about a newly created component can also be advertised to the network (5).

There is also a shutdown procedure that announces the ceasing of component or service offer.

2) Service / Component discovery: Figure 4 shows the three steps necessary to acquire context information without the help of a context-server.

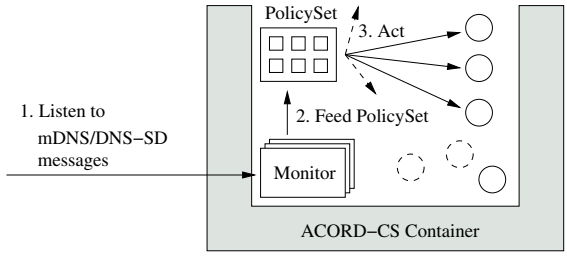

Fig. 4. Component and service discovery.

A special context monitor (1) listens for mDNS/DNS-SD messages and updates a list of available components directly accessible from the local network link.

The message carries the following information (this message was received by the network interface eth0 and uses IPv4):

```
= eth0 IPv4 Component X \
  _robot_remote_control._tcp local
  hostname = [movel.local]
  address = [10.10.1.42]
  port = [8888]
  txt = ["protocol=http,RMI;model=X12;\
    parameterY=12345"]
```

"Component X" is the identification of the announced component or service and _robot_remote_control._tcp is the service type and transport protocol it offers. "hostname" and "port" identify the location of the service and "txt" is used to provide any other information necessary to make use of the component. This example uses "txt" to indicate protocol and robot model. Some filtering and pre-processing is done in order to comply with the PolicySet data format. Relevant context information is inserted into the PolicySet (2), which, in turn, acts accordingly (3). An action can be anything possible from the component technology point of view (i.e. component downloading, connection, deletion, and so on).

3) Component replacement: At some point of execution, a context-aware container may be forced to replace a component in order to keep some quality of service (QoS) aspects such as security, speed, and so on. The state of the component to be replaced can be dumped to an XML file for further recovering by a new component. The ACORD-CS container offers two methods for that purpose: dumpComponentState, to generate the XML state file; and recoverComponent to update the state of the new component. The format of the XML state file is similar to the following:

```
<objectdump id="ctx.CContainer@18a7efd">
```

```
  <timestamp>1168630673952</timestanp>
  <class>ctx.CContainer</class>
  <attribute
     setby="setFileSystemHome"
     type="class java.lang.String"
     value="/framework/example/"/>
  <attribute
     setby="setContextServer"
     type="interface acords.server.\
     ContextServerInterface"
     value="acords.server.ContextServer\
     Interface_RMIWrap_LocalAccess@8b63d33e"
     remote="ContextServerInterface_RMI\
     Wrap_Stub[UnicastRef [liveRef: \
     [endpoint:[10.10.1.2:60044](remote),\
     objID:[-5d472694:110129768b2:-7fff,\
       8971166976842126142]]]]"/>
...
</objectdump>
```

The previous dump file represents an instance of a class called ctx.CContainer (which is an ACORD-CS container). Tag objectdump owns an attribute that identifies the instance of the dumped component (there must be several of them). It also provides timestamp and type (class) information. There is an attribute tag for each class attribute (the excerpt code shows only two of the several existent attributes). It identifies the method that must be called to set the information, the type of the information and the corresponding value. It also provides a remote reference to an object, when it is the case.

4) Component swapping: In order to reduce main memory execution footprint, it is possible to swap out unused components. The swap operations are executed by the methods swapOutComponent and swapInComponent. When a component is swapped out, it is created an XML file very similar to the one used for component replacement. Proxies of the components are not swapped out. It function as a mechanism to allow for the automatic swap in of components. When a client makes use of a proxy that represents a swapped out component, the container automatically swaps in the necessary component. To keep track of this, components can be in one of the following states: swapped, active, stopped and destroyed.

5) On-the-fly event handling installation: With the feature of new component and monitor installation at run-time, the need for new event handlers are natural. Every ACORD-CS element features the possibility of event handling definition at run-time. The methods to make it possible are installEventHandler and destroyEventHandler.

6) On-the-fly stub generation: Every ACORD-CS container is able to generate stubs for components it provide. These stubs can be requested by a client that wants to interact with the component. The current implementation offers generation for RMI (Java Remote Method Invocation) and CORBA (IIOP) interaction, but the container allows the installation of new ones. New stub generators are installed with the method installExporter.

Context Servers

Context servers are responsible for gathering, compiling and distributing contextual 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 via queries or subscription for further notification. For instance, a monitor can subscribe to receive updated contextual information, such as new resources that are incorporated into the environment. Context servers can also employ policies to control the way context information is propagated to the subscribed parties.

This server can be used by entities that don't support mDNS/DNS-SD messages, but also allows for high-level context inference that can be used even by the entities that support mDNS/DNS-SD messages.

Component Servers

Component servers store factories for managing context-aware components and other software artifacts needed by the components such as stubs for remote interaction; files containing images, scripts, and configuration parameters; and libraries. Component servers provide functionalities for searching and storing software artifacts. Both context and component servers can be grouped into federations of collaborating servers.

Supporting servers

Supporting servers help the context-aware containers to support non-functional requirements. Examples of supporting servers are security/authentication servers, transaction servers, and quality of service (QoS) servers. In special, a QoS server offers a significant service. 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. Also, some signal and operational ports could require to operate within some bounded time constraints. In this case, resource reservation in the network could assure the adequate response time. The QoS server [9] accepts requests for the reservation of network resources, such as bandwidth and traffic priority. This server has the same functionality as a Differentiated Service (DiffServ) Bandwidth Broker.

The ACORD-CS Platform

The ACORD-CS platform [1] is an implementation of the architectural framework described above. The platform consists of a set of utility classes, a code generation engine, and a set of servers. The utility classes provide utilities such as a rule-based execution engine, base classes for the model elements, and stubs for interacting with the servers.

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. Transformations generate 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). ACORD-CS employs the Linux utility *xsltproc* (from package libxslt) as an XSLT processor.

The code generation engine is dependent of technology. The current version of ACORD-CS generates Java components interacting through RMI. Jess [10] was chosen as a rule execution engine. As such, the rules are generated according to the Jess syntax.

For containers not capable of decentralized operation, ACORD-CS provides a context server that offers a publish/subscribe service for contextual information processing. Publishers invoke the context server to submit an XML document containing contextual information. A subscriber registers to the service by supplying its notification interfaces and, optionally, an XPath expression for selecting the contents or the published contextual information. In this case, subscribers are notified only with the documents that satisfy the supplied XPath expression. In addition to the push style of operation, the context server supports the pull style where information consumers poll the service for stored contextual information (also selected according to a supplied XPath expression). Push subscribers are notified as soon as the information is published. If the information specifies a lifetime, it is kept on a database during this lifetime.

A QoS server implementing a Differentiated Service (DiffServ) Bandwidth Broker [9] is also provided by ACORD-CS as a supporting server. The current version of ACORD-CS supplies a component server capable of receiving XPath expressions to search for component factories, stubs, and other software artifacts. The actual software artifacts are stored on any HTTP server accessible from the domain.

A brief scenary

An example of the usage of the proposal can be seen in reference [1], which discribes a WebLab (laboratorial infrastructure manipulated through the internet) for mobile robots experiments. One of the experiments in the WebLab is a vision-based robot navigation where a mobile robot must follow a colored stripe on the floor using solely its on-board camera. The WebLab allows several users to perform the experiment at the same time. Depending on the number of users, network bandwidth, processing power of the user's terminal, and robot speed, some adaptation is required in order to adjust the experiment parameters to the context of the users' terminal and their access networks. These adaptations include modification of the camera's frame rate parameter and robot speed. All the necessary adaptation is achieved autonomously through the context-awareness mechanisms described in this paper.

IV. Conclusion

The process of developing context-aware applications is rather difficult and error prone. The reason is the complexity of the tasks related to context-awareness that are commonly left to the application developer. These tasks include the gathering, processing, and distribution of contextual information, the logic of the adaptation mechanism, and quality of service processing. Well designed context-aware software components and metamodels that address context-awareness help to relief the developer from these tasks. In our proposal, 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. The ability to announce and discover components using mDNS/DNS-SD on a local network offers a de facto solution to resource discovery and auto configuration capabilities. The servers offered by ACORD-CS enable the discovery and use of contextual information and software components, not present at startup time, by the application containers, and allow the use of different resources encountered during the application execution. Moreover, the quality of service offered to audio and video flows are guaranteed by network resource reservation featured in the ACORD-CS QoS server.

Related work

Gravity [11] offers a mechanism to connect OSGi [12] components (services). Gravity authors mention that it lacks support to transfer state between components in a replacement. State transfer is possible in ACORD-CS.

PCOM [13] presents a distributed component model for pervasive computing. It provides mechanisms to add, remove and replace components while mobile terminals appear and disappear on an ad-hoc wireless network. PCOM uses BASE [14], a middleware for pervasive environments, to adapt communication layers (dynamically reselects stacks of communication protocols according to context).

DYVA [15] is a virtual dynamic reconfiguration machine. The virtual aspect of DYVA refers to its independence from a particular application or a particular component model. DYVA proposes a metamodel for the adaptation of component based software via the instrumentation of an already developed application. The instrumentation technique inserts interception code in key points of the application, such as constructors, methods, and so on. DYVA requires an XML description of the application in order to be able to insert the appropriate code into the existing application.

ACORD-CS shares some similarities with DYVA in respect to the construction of an abstract representation of the running application with a causal relation between the application and its abstract representation (in ACORD-CS this abstract representation is provided by shadow objects in Jess). Another similarity is that both DYVA and ACORD-CS are targeted to component-based software systems. The key difference is that while DYVA focuses on instrumenting an existing application, ACORD-CS focuses on the development of new applications.

The Chisel [16] framework implements what the author calls "completely unanticipated dynamic adaptation of software". The author argues that if the adaptation can not be foreseen until the application starts to run, then a completely unanticipated dynamic adaptation of software must be achieved. If adaptation is not envisioned in design, implementation,

compilation, loading, and execution phases, Chisel makes adaptation still possible through the use of interpreted scripts that can change during run-time.

The ACORD-CS framework is similar to Chisel in the sense that it offers completely unanticipated dynamic adaptation of software through a Jess inference engine. All the adaptation policies are written in the Jess language, which is an interpreted script that can be changed during run-time. It is possible to write rules that can learn about its execution and make proper adaptation through the modification of itself.

Several research proposals in context-aware applications focus on the modeling of contextual information. Strang [17] classifies contextual information into six modeling approaches, according to the data structure used to store and process the information. The categories are key-value (linear data structures), markup scheme (hierarchical data structures, e.g. CC/PP [18]), graphical language (e.g. UML or ORM [19]), object oriented (allows the extension by inheritance, [20]), based on logic, and based on ontologies [21]. Strang concludes that the most appropriate approaches are the object oriented and ontology-based ones. The metamodel and framework proposed in Section II and III favor the object oriented approach for contextual information modeling (*CtxMonitor*).

ContextUML 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 our metamodel, but without the limitation of Web services as the only sources of contextual information. Conditions and actions in ContextUML are specified in OCL (Object Constraint Language) [22]. We believe that OCL is too limited for defining adaptation strategies. As such, we employ tags (*Rule*, *Condition* and *Action*) with syntaxes easily mapped to rule-based engines.

While ContextUML ties the metamodel to the Web Services technology, we prefer to leave the metamodel free from details assigned to a particular technology. Placing technology dependencies on the platform, instead of on the metamodel, has two main advantages. Firstly, the same metamodel can be bound to different technologies. Secondly, the platform can generate contextual information processing functions from specification.

REFERENCES

[1] R. P. Pinto, E. Cardozo, P. R. Coelho, and E. G. Guimarães, "A Domain-independent Middleware Framework for Context-aware Applications," in *ARM 2007*. Newport Beach, CA, USA: ACM, 2007.

[2] R. P. Pinto, E. Cardozo, and E. G. Guimarães, "A Component Model for Context Aware Applications," in *V Component Based Development Workshop (WDBC'05)*. Juiz de Fora, MG: SBC (In Portuguese), November 2005.

[3] *Unified Modeling Language*, http://www.uml.org/, last access in Sep 27, 2005.

[4] ISO/IEC, "Reference Model of Open Distributed Processing, Part 3, Architecture," Tech. Rep., May 1995, iSO/IEC 10746-3.

[5] E. G. Guimarães, A. T. Maffeis, J. L. Pereira, B. G. Russo, E. Cardozo, M. Bergerman, and M. F. Magalhães, "REAL: A Virtual Laboratory for Mobile Robot Experiments," *IEEE Transactions on Education*, vol. 46, no. 1, pp. 37–42, Feb. 2003.

[6] E. G. Guimarães, A. T. Maffeis, R. P. Pinto, C. A. Miglinski, E. Cardozo, M. Bergerman, and M. F. Magalhães, "REAL: A Virtual Laboratory Built from Software Components," *Proceedings of the IEEE*, vol. 91, no. 3, pp. 440–448, March 2003.

[7] R. P. Pinto, E. Cardozo, A. Z. Lima, P. R. Coelho, E. G. Guimarães, R. F. Sassi, and L. F. Faina, "An Architecture for the Support of Context Aware Applications Based on Software Components," in *XXXII Conferencia Latinoamericana de Informtica - CLEI 2006 jointly organized with WCC 2006*, M. Marin and G. A. Leiva, Eds. Santiago, Chile: CLEI/IFIP, 2006, www.clei2006.org.

[8] D.-S. org, "DNS-Service Discovery Home Page," http://www.dns-sd.org - Last access in Sat Mar 15.

[9] R. P. Pinto, E. G. Guimarães, E. Cardozo, and M. F. Magalhães, "Incorporation of Quality of Service into Telematic Applications," in *21 Brazilian Symposium of Computer Networks (SBRC'03)*. Natal, RN: SBC, May 2003, (In Portuguese).

[10] E. Friedman-Hill, *Jess in Action: Rule-Based Systems in Java*. Manning, 2003.

[11] H. Cervantes and R. S. Hall, "Autonomous Adaptation to Dynamic Availability Using a Service-Oriented Component Model," in *ICSE'04: Proceedings of the 26th International Conference on Software Engineering*. IEEE Computer Society, 2004.

[12] *About the OSGi Service Platform - Technical Whitepaper*, OSGi Alliance, Bishop Ranch 6, 2400 Camino Ramon, Suite 375, San Ramon, CA 94583 USA, July, 12 2004, http://www.osgi.org/documents/osgi_technology/osgi-sp-overview.pdf - ltimo acesso em 07/03/2006.

[13] C. Becker, M. Handte, G. Schiele, and K. Rothermel, "PCOM - A Component System for Pervasive Computing," in *Second IEEE International Conference on Pervasive Computing and Communications*. Orlando, USA: IEEE Computer Society, 2004, pp. 67–76.

[14] C. Becker, G. Schiele, H. Gubbels, and K. Rothermel, "BASE - A Micro-broker-based Middleware For Pervasive Computing," in *First IEEE International Conference on Pervasive Computing and Communications - PerCom*. Forth Worth, USA: IEEE Computer Society, March 2003, pp. 443–451.

[15] A.Ketfi and N.Belkhatir, "A Metamodel-based approach for the dynamic reconfiguration of component-based software," *ICSR*, 2004, http://www-adele.imag.fr/Les.Publications/intConferences/ICSR2004.pdf - Last access in Oct 10, 2006.

[16] J. Keeney, "Completely Unanticipated Dynamic Adaptation of Software," Ph.D. dissertation, Department of Computer Science, Trinity College Dublin, May 2005, https://www.cs.tcd.ie/publications/tech-reports/reports.05/TCD-CS-2005-43.pdf - Last access in Dec 12, 2006.

[17] T. Strang and C. Linnhoff-Popien, "A Context Modeling Survey," Sep 2004, http://citeseer.ist.psu.edu/strang04context.html - Last access in Nov. 29, 2006.

[18] J. Indulska, R. Robinson, A. Rakotonirainy, and K. Henricksen, "Experiences in Using CC/PP in Context-Aware Systems," in *MDM 2003*, ser. LNCS 2574. School of Information Technology and Electrical Engineering - The University of Qeensland, 2003, pp. 247–261.

[19] K. H. Jadwiga, "Modelling Context Information with ORM," http://citeseer.ist.psu.edu/734803.html - Last access in Nov 11, 2006.

[20] K. Cheverst, K. Mitchell, and N. Davies, "Design of an Object Model for a Context Sensitive Tourist Guide," in *Computers and Graphics*, 1999, pp. 883–891.

[21] T. Gu, X. Wang, H. Pung, and D. Zhang, "Ontology Based Context Modeling and Reasoning using OWL." in *Proceedings of the 2004 Communication Networks and Distributed Systems Modeling and Simulation Conference - CNDS2004*, San Diego, CA, USA, January 2004.

[22] J. Warmer and A. Kleppe, *The Object Constraint Language*, 2nd ed. Addison Wesley, 2003.