Interoperability between Mobile Distributed Components using the UniFrame Approach

Purvi V. Shah
Barrett R. Bryant
Carol C. Burt
Department of Computer and
Information Sciences
University of Alabama at Birmingham
shahp@cis.uab.edu
bryant@cis.uab.edu
cburt@cis.uab.edu

Rajeev R. Raje
Andrew M. Olson
Department of Computer and
Information Science
Indiana University Purdue University
rraje@cs.iupui.edu
aolson@cs.iupui.edu

Mikhail Auguston
Department of Computer Science
New Mexico State University
mikau@cs.nmsu.edu

ABSTRACT

Frequently, software development for large-scale Distributed Component Systems (DCS) requires a combination of distributed and heterogeneous components (i.e., software components that adhere to different middleware models). One solution for the integration of heterogeneous components is the UniFrame Approach (UA). The UA provides a comprehensive framework that unifies existing and emerging distributed component models under a common meta-model, called the Unified Meta-Model (UMM). The UMM enables the discovery, interoperability, and collaboration of heterogeneous components and envisions a plug and play component environment where Quality of Service contracts are a part of the component description. Middleware bridges are generated by component integration toolkits, which are parts of the UA that allow an easy integration of heterogeneous components to build a DCS. This paper presents the Unified Component Interoperability framework that addresses the challenges of generating middleware bridges for the collaboration aspect of UniFrame research. A special emphasis is given to issues involved in the interoperability of heterogeneous components to build a mobile DCS.

Keywords

UniFrame Approach, distributed heterogeneous software components, mobile distributed component systems.

1. INTRODUCTION

A number of middleware technologies have evolved over the last decade to address specific business problems. In an effort to facilitate the design of business systems in a platform independent matter, the Object Management Group (OMG) is currently progressing the Model Driven Architecture (MDA^{TM}) [4]. MDA^{TM} maintains a clean separation of Platform Independent Models (PIMs) which represent the domain from Platform Specific Models which expose details related to a middleware technology. In this way a single PIM can be mapped to multiple implementation technologies (such as OMG CORBA®, Sun Java^{TM} 2 Enterprise Edition and Microsoft® COM+).

While all of the component-based technologies have established a concise means for describing functional contracts (the interface or services offered by a component in the architecture), none of these technologies have embraced an architecture or a vocabulary for specifying non-functional Quality of Service (QoS) contracts. Non-functional contracts are necessary to analyze the ability of a service to meet QoS constraints when used in a composition.

The UniFrame Approach (UA) [5] (which includes identification and progression of requisite standard activities) envisions a plug and play component environment where QoS contracts are part of component description. Middleware Bridges and QoS instrumentation are generated by component integration toolkits, which are parts of UA, thereby allowing an easy integration of to form a Distributed Component System.

The recent advances in wireless networking technologies and the trend towards mobile computing devices, (e.g., laptop computers, third generation mobile phones, personal digital assistants, watches), are enabling new classes of mobile DCS applications that present challenging problems to a DCS developer. Traditional object-oriented middleware has been adapted to mobile settings, mainly to make the mobile devices inter-operable with existing fixed networks (e.g., nomadic setting). On the other hand, the mobile middleware specifically targeting the needs of mobile computing has also been devised.

This paper presents the Unified Component Interoperability (UCI) framework that addresses the challenges of generating

middleware bridges for the collaboration aspect of UA. A special emphasis is given to issues involved in the interoperability of heterogeneous components to build a mobile DCS. This framework, hence, includes support for integrating heterogeneous components adhering to mobile middleware and traditional middleware models. This would prove to be beneficial to the COTS mobile DCS developers by shielding them from the implementation nuances of different component types viz. Mobile[1] and non-Mobile Components. The mobile DCS developer would be able to develop a mobile DCS with components implemented under diverse component models.

The rest of the paper is organized as follows: Section 2 provides a brief background about the UA; Section 3 explains the objectives and the basics concepts of the UCI framework; Section 4 describes the experiment setup; Section 5 presents a case study; and finally Section 6 states conclusions that one can draw from this work.

2. UNIFRAME APPROACH (UA)

The UA attempts to unify the existing and emerging distributed component models under a common meta-model for the purpose of enabling discovery, interoperability, and collaboration of components via generative programming techniques. This work on the framework addresses the challenges of generating middleware bridges for the collaboration aspect of the UniFrame (Unified Component Meta Model Framework) research project. The core parts of the UA are: Unified Meta-Model [5] and the Formal Specification and the Automated System Generation.

2.1 The Unified Meta-Model (UMM)

The UMM defines the framework that provides an organization for the distributed network of heterogeneous components, such as the Internet, within which a software engineering team builds a DCS. Its core concepts are: components, service and service guarantees and infrastructure. The innovative aspects of the UMM lie in the structure of these parts and their inter-relations. It provides ways to bridge gaps that currently exist in standards.

2.1.1 Component

In UniFrame, components are autonomous entities, whose implementations are non-uniform. Each component adheres to a distributed-component model but there is no notion of a unified implementation framework. Each component has a state, an identity, a behavior, a well-defined interface and a private implementation. A component has three aspects; computational, cooperative and auxiliary. The computational aspect reflects the tasks a component is able to carry out. Each UMM component precisely describes its services to others so that they may subscribe to them if they desire. Components actively cooperate with one another, serving them or delegating tasks to others. They can form consortia to solve specific problems through this cooperative aspect. A component's auxiliary aspect consists of additional characteristics, such as security or fault tolerance.

2.1.2 Service and Service Guarantees

A service offered by a component could be an intensive computational effort or an access to underlying resources. In a DCS, it is natural to expect several choices for obtaining a specific service. Thus, each component must be able to specify the QoS offered. The QoS is an indication given by a component, on behalf of its owner, about its confidence to carry out the required services. The QoS offered by each component depends upon the computation it performs, the algorithm used, its expected computational effort, required resources, the motivation of the DCS developer, and the dynamics of supply and demand. A catalog of various QoS parameters that may be used is described in [5].

2.1.3 Infrastructure

In a heterogeneous network, there will be clusters of components that are capable of communicating with components in other clusters. To bridge gaps between alien clusters, UMM employs three special types of components, a Headhunter, an Adapter and an Internet Component Broker (ICB). The headhunter and ICB [5] are responsible for allowing a seamless integration of different component models and sustaining cooperation among heterogeneous components. The tasks of headhunters are to detect the presence of new components in the search space, register their functionalities, and attempt matchmaking between service producers and consumers. It attempts at discovering components and registering them. Details of the headhunter architectures are given in [5] and [6]. Headhunters may cooperate with each other in order to serve a large number of components. The ICB acts as a translator between heterogeneous components. Adapter components register with ICB and indicate their specializations (which component models they can bridge efficiently). During a request from a seeker, the headhunter component not only searches for a provider, but also supplies the necessary details of an ICB.

2.2 Formal Specification and Automated System Generation

A software engineering team can manually carry out the entire process of creating a DCS in UniFrame, including searching for components via the Web or by the phone. Nevertheless, one of UA's goals is to formalize the process so that a computer can automate as much of it as practical. It assumes that the DCS developer receives an informal description of the problem. The UA determines from this informal specification corresponding UMM Application Domain Platform Independent Formal Specifications. These latter formal specifications state the kind of components in the domain that are to make up the DCS, how these relate to one another within the DCS, and any other

---

[1] In this paper, the term "Mobile components" will be referred to as components, which are used for application development on the wireless devices, like the PDAs.

constraints that the DCS must satisfy. They form the basis for issuing component requests to the headhunters in the network and for assembling the components they return into an implementation of the needed DCS. Different component providers will provide on the Internet a variety of possibly heterogeneous components oriented towards a specific problem domain. Once all the components necessary for implementing a specified DCS are available then the task is to assemble them into a solution. This requires the Headhunters to deliver an address of the component and its formal UMM platform specific specifications. UniFrame takes a pragmatic approach, based on Generative Programming [3], to component-based programming. This research targets on this aspect of UA and thus focusing on the assembly of mobile DCS from heterogeneous components.

Further details about the UA can found in [1], [5] and [6].

3. UNIFIED COMPONENT INTEROPERABILITY (UCI) FRAMEWORK

The objectives of the UCI framework are:

- To identify a set of rules for static and dynamic assembly of heterogeneous mobile and non-mobile components according to the UA.

- To functionally bridge the gap between the heterogeneous components in order to coordinate how they are connected and interoperate amongst them.

There are essentially two parts in this framework: UMM formal specifications associated with the components and a heterogeneous component integrator.

The UMM formal specifications as indicated in section 2.2 are based on mobile DCS developer's informal specifications. These specifications define both the functionality and QoS contracts of a component.

The component integrator itself consists of: a translator, an internal representation and the Middleware Bridge Generation Engine (MBGE). The translator takes the platform specific UMM specifications of the heterogeneous components and translates them into the unified internal representation.

The translator maps the functional attributes of the platform specific UMM specifications of the heterogeneous components to that of the platform independent UMM specifications. Based on this mapping of the two heterogeneous components the MBGE then generates the required bridge code between two heterogeneous components.

A major problem, concerning the genericity of the components, is the application dependent information that may be needed for generation of the middleware bridges. This is tackled by separation of UMM specifications into platform independent UMM specifications that represent the domain from platform specific UMM specifications that expose details related to a middleware technology.

4. EXPERIMENT SETUP

This framework should be useful for the development of middleware bridges to attain interoperability between heterogeneous components in a mobile DCS. Mobile DCS are systems whose functionality is distributed between PDAs, front ends and host computer back ends. These systems allow multiple
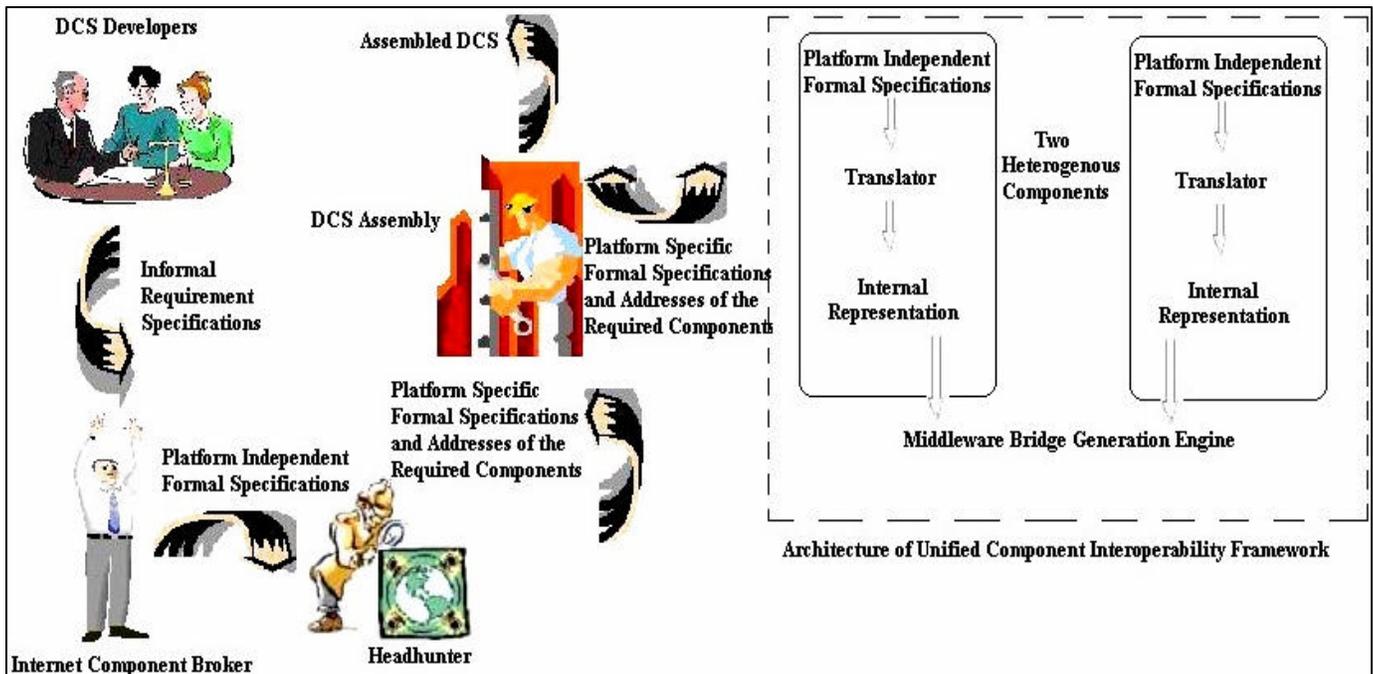

Figure 1. Architecture of Unified Component Interoperability Framework

users, at different locations to collaborate.

Our experiment setup consists of a nomadic Mobile Computing Environment (MCE) built of a network consisting of PDAs, PC's, workstations and servers. We have a Xircom® Wireless Ethernet Access Point, APWE1100 attached to the existing wired network by Ethernet cable.

Currently, we have two PDAs connected to the Ethernet by the wireless access point. In order to have a heterogeneous MCE we are using the following two PDAs,

1) Sharp Zaurus SL-5500 Linux[TM] PDA: This is a high-end PDA. There is Personal Profile implementation of Java[TM] 2 Micro Edition (J2ME) [7] for the Zaurus. The Linksys® Wireless Network CompactFlash[TM] card, WCF11 connects this PDA to the Ethernet via the access point.

2) Palm[TM] m500 PDA: This PDA features Palm Operating System®. It is a low-end PDA that supports Mobile Information Device Profile (MIDP) of J2ME. The Xircom® Wireless LAN Module for Palm[TM] handheld (PWE1130) connects this PDA to the Ethernet via the access point.

Sample applications are to be implemented on these PDAs using the Java[TM] 2 platform. The Java[TM] 2 platform is an excellent choice for development of software components in this MCE scenario.

Since, we are focusing on developing the Mobile Components for small devices such as PDAs, J2ME is used. J2ME uses subsets of Java[TM] 2 Standard Edition components, such as smaller Java[TM] Virtual Machines (JVM's) and leaner Application Programming Interfaces (APIs).

J2ME is divided into configurations and profiles. Configurations are specifications that detail a JVM and a base set of APIs that can be used with a certain class of device. A profile builds on a configuration but adds more specific APIs to make a complete environment for building applications. J2ME has two main branches,

1) Connected Limited Device Configuration (CLDC): This configuration is for small wireless devices with intermittent network connections, like pagers, mobile phones, and low-end PDAs. The MIDP, which is based on CLDC, is a J2ME application environment. Palm[TM] PDAs are MIDP-compliant devices.

2) Connected Device Configuration (CDC): This configuration is for larger devices (in terms of memory and processing power) with robust network connections. The Sharp Zaurus PDA fits into this configuration. The Foundation Profile extends CDC and serves as the basis for several other profiles.

The PDAs we are using in our MCE each fit into one of the above-mentioned J2ME configurations. Hence, we are able to explore the interoperation of different Mobile component technologies.

The criteria for the sample DCS is that they should be representative for mobile DCS and they should cover the main properties of open distributed systems. In this way we ensure that the solutions are solutions to actual problems and thus that our approach has the required capabilities. We have chosen a Bank Account Management System (BAMS) as our sample application.

5. A SCENARIO

This section presents a short example to illustrate the basic process involved in constructing a middleware bridge to achieve interoperability between two heterogeneous components to build a mobile DCS using the UCI framework within UniFrame.

Let us assume that a private bank is trying to build a software system to automate its day-to-day operations. The bank has to utilize a Client-Server DCS model with client software residing on a PC as well as a PDA. The bank has also chosen to assemble the system using COTS software components instead of building the system from scratch. The in-house software development team in the bank has come out with the following simple design for the system:

- The system consists of two categories of components:

BAMS Server Component Description in UMM (Platform Independent Model)	BAMS Client Component Description in UMM (Platform Independent Model)

0) Component Name: AccountServer
1) Informal Description: Provides an account management service. Supports 3 functions,
    a) Deposit
    b) Withdraw
    c) Balance
2) Computational Attributes:
    a) Functional Attributes: a.1) Acts as an account server
3) Cooperation Attributes:
    a) Pre-processing Collaborators: a.1) AccountClient
    b) Post-processing Collaborators: a.2) AccountDatabase
	0) Component Name: AccountClient
	1) Informal Description: Requests Bank Account services from an appropriate server and interacts with the user interface services. Supports 3 functions,
	    a) Deposit
	    b) Withdraw
	    c) checkBalance
	2) Computational Attributes:
	    a) Functional Attributes: a.1) Accepts interface services queries and returns the answers
	3) Cooperation Attributes:
	    a) Pre-processing Collaborators: a.1) AccountServer
	    b) Post-processing Collaborators: a.2) AccountDatabase

Figure 2. Platform Independent Formal Specification, of a desired BAMS

`AccountServer` and `AccountClient`.

- The components should offer the following functionality: Deposit, Withdraw and Balance check.

The DCS developer can state an informal specification for the above DCS as: Create a BAMS that has provides the following functionality: Deposit, Balance and Withdraw.

Also, it meets with the following QoS requirements:

- Dependability > 0.97

- Turn-around Time < 100

From the informal specification and the available knowledge associated with Bank Account Management System, a platform independent formal specification, of a desired DCS will be formulated for a headhunter in UniFrame as shown in Figure 2.

In response, the headhunter discovers the choices shown in Figure 3.

Thus, the final system should be assembled from `javaRMIAccountClient` for PDA (falling in the CDC configuration of J2ME) users, `javaCorbaAccountClient` for PC users and `javaCorbaAccountServer` with middleware code for the 'javaRmi-CorbaBridge' would be required for interoperability between the server component and the PDA client component. This bridge would redirect the service request coming from the `javaRMIAccountClient` to the `javaCorbaAccountServer` that ultimately provides the service.

The translator shall take the platform specific UMM specifications of the heterogeneous components (Java-Rmi Client Component and the Java-Corba Server Component) and translates them into the unified internal representation. It maps the

BAMS Server Component Description in UMM (Platform Specific CORBA Model)	BAMS Client Component Description in UMM (Platform Specific CORBA Model)	BAMS Client Component Description in UMM (Platform Specific Java-RMI Model)

0) Component Name: javaCorbaAccountServer
1) Informal Description: Provides an account management service. Supports 3 functions,
a) corbaDeposit
b) corbaWithdraw
c) corbaBalance
2) Computational Attributes:
a) Functional Attributes:
a.1) Acts as an account server
a.2) Algorithm: Simple addition/subtraction
a.3) Complexity: 0(1)
a.4) Syntactic Contract:
a.4.I) void corbaDeposit(float ip);
a.4.II) void corbaWithdraw(float ip) throws overDrawException;
a.4.III) float corbaBalance();
a.5) Technology: Java-CORBA
b) Inherent Attributes:
b.1) id: pc123d1.cis.uab.edu/javaCorbaAccountServer
3) Cooperation Attributes:
a) Pre-processing Collaborators:
a.1) AccountClient
b) Post-processing Collaborators:
b.1) AccountDatabase
4) Auxiliary Attributes: a) SecurityAttribute: -
5) QoS Metrics:
a) Availability: 95%
b) Turn-around Time: MTAT=90
c) HardwareResources: Workstation/ PC, Ethernet connection
d) SoftwareResources: J2EE version 1.4 software environment
e) Dependability: 0.99
	0) Component Name: javaCorbaAccountClient
	1) Informal Description: Requests Bank Account services from an appropriate server and interacts with the user interface services. Supports 3 functions,
	a) javaCorbaDeposit
	b) javaCorbaWithdraw
	c) javaCorbaCheckBalance
	2) Computational Attributes:
	a) Functional Attributes:
	a.1) Accepts interface services queries and returns the answers
	a.2) Algorithm: -
	a.3) Complexity: 0(1)
	a.4) Syntactic Contract:
	a.4.I) void javaCorbaDeposit(float ip);
	a.4.II) void javaCorbaWithdraw(float ip);
	a.4.III) float javaCorbaCheckBalance();
	a.5) Technology: Java-CORBA
	b) Inherent Attributes:
	b.1) id: pc123d1.cis.uab.edu/javaCorbaAccountClient
	3) Cooperation Attributes:
	a) Pre-processing Collaborators:
	a.1) AccountServer
	b) Post-processing Collaborators:
	b.2) AccountService
	4) Auxiliary Attributes: a) SecurityAttribute: -
	5) QoS Metrics:
	a) Availability: 80%
	b) Turn-around Time: MTAT=90
	c) HardwareResources: Workstation/ PC, Ethernet connection
	d) SoftwareResources: J2EE version 1.4 software environment
	e) Dependability: 0.98
		0) Component Name: javaRmiAccountClient
		1) Informal Description: Requests Bank Account services from an appropriate server and interacts with the user interface services. Supports 3 functions,
		a) javaRmiDeposit
		b) javaRmiWithdraw
		c) javaRmiCheckBalance
		2) Computational Attributes:
		a) Functional Attributes:
		a.1) Accepts interface services queries and returns the answers
		a.2) Algorithm: Java™ Foundation Classes (JFC)
		a.3) Complexity: 0(1)
		a.4) Syntactic Contract:
		a.4.I) void javaRmiDeposit(float ip);
		a.4.II) void javaRmiWithdraw(float ip);
		a.4.III) float javaRmiCheckBalance();
		a.5) Technology: Java-RMI
		b) Inherent Attributes:
		b.1) id: pc123d1.cis.uab.edu/javaRmiAccountClient
		3) Cooperation Attributes:
		a) Pre-processing Collaborators:
		a.1) AccountServer
		b) Post-processing Collaborators:
		b.2) AccountService
		4) Auxiliary Attributes: a) SecurityAttribute: -
		5) QoS Metrics:
		a) Availability: 80%
		b) Turn-around Time: MTAT=90
		c) HardwareResources: Workstation/ PC, Ethernet connection
		d) SoftwareResources: J2ME software environment
		e) Dependability: 0.98

Figure 3. Platform Dependent Formal Specification of Components discovered by the Headhunter

functional attributes of the platform specific UMM specifications of these heterogeneous components to that of the platform independent UMM specifications. For example the translator in this case maps the functional attributes of the platform independent formal specifications of a client component to its platform specific formal specifications as shown in Table 1.

Table 1. BAMS Client Component Description in UMM

Platform Independent Formal Specifications	Platform Specific Java-RMI Formal Specifications
Deposit	JavaRmiDeposit
Withdraw	JavaRmiWithdraw
checkBalance	JavaRmiCheckBalance

Based on the mapping shown in Table 1 of the two heterogeneous components the MBGE then generates the Java-RMI proxy server for the above Java-RMI client component. Similarly it generates a Java-Corba Proxy Client for the Java-Corba server component. Finally, it merges the code and generates the required bridge code between two heterogeneous components.

The bridge code provides a common message-forwarding interface between two component models therefore taking care of request service mapping, data type mapping, parameter passing, etc. Thus, a major problem, concerning the genericity of the components, that is the application dependent information that may be needed for generation of the middleware bridges is tackled by the separation of the UMM specifications into platform independent UMM specifications that represent the domain from platform specific UMM specifications that expose details related to a middleware technology.

One restriction to the generation of middleware bridge using this framework is that the bridge is to be coded in a language that can be mapped to both the component models. However, in this example Java^{TM} language can be mapped to both RMI and CORBA® component models.

Currently we have established connectivity between the components on PDAs (falling in the CDC configuration of J2ME) implemented using the J2ME-RMI component model (traditional object-oriented middleware that has bee adapted to mobile settings) with the components adhering to other component models such as CORBA®.

Efforts are underway to establish connectivity between the components on PDAs (falling in the CLDC configuration of J2ME) implemented using J2ME and mobile middleware with the components adhering to traditional component models such as CORBA® and Java^{TM}-RMI.

6. CONCLUSION

This paper has presented a Unified Component Interoperability framework for the interoperability of heterogeneous components

that is a part of UniFrame research. Our approach of building interoperability among all the component models should be the same as long as the UMM formal platform dependent as well as platform independent specifications of the heterogeneous components and the associated domain knowledge are specified. This framework would allow a mobile DCS developer to use heterogeneous components to build a mobile DCS. Although a simple case study is provided in this paper the principles of the proposed approach are general enough to be applied to any large mobile DCS.

7. ACKNOWLEDGMENTS

8. REFERENCES

[1] C. C. Burt, R. R. Raje, M. Auguston, B. R. Bryant, A. M. Olson, "Quality of Service Issues Related to Transforming Platform Independent Models to Platform Specific Models", Proceedings of the 6th IEEE International Enterprise Distributed Object Computing Conference, pp 212-223, 2002.

[2] L. Capra, W. Emmerich, C. Mascolo. "Middleware for Mobile Computing (A Survey)". UCL Research Note RN/30/01, 2001.

[3] K. Czarnecki, U. W. Eisenecker, "Generative Programming: Methods, Tools, and Applications", Addison Wesley, 2000.

[4] Object Management Group. 2001. Model Driven Architecture: A Technical Perspective. Technical Report. Document # ormsc/ 2001-07-01, Object Management Group, 2001.

[5] R. R. Raje, B. R. Bryant, A. M. Olson, M. Auguston, C. C. Burt, "A quality-of-service-based framework for creating distributed heterogeneous software components", Concurrency and Computation: Practice and Experience, pp 1009-1034, 2002.

[6] N. N. Siram, R. R. Raje, B. R. Bryant, A. M. Olson, M. Auguston, C. C. Burt, "An Architecture for the UniFrame Resource Discovery Service", Proceedings of the 3rd International Workshop on Software Engineering and Middleware, 2002.

[7] Sun Microsystems, "Java^{TM} 2 Micro Edition Technology", August 2002, "http://java.sun.com/j2me/".