KSACI: A Handheld Device Infrastructure for Agents Communication

Submitted to the following Special Track:
"AGENTS FOR HAND-HELD, MOBILE OR EMBEDDED DEVICES"

Ryan L. Albuquerque [†]   Jomi F. Hübner [‡]   Gustavo E. de Paula [†]

Jaime S. Sichman [‡]   Geber L. Ramalho [†]

[†]   Centro de Informática, Universidade Federal de Pernambuco,
      Av. General Polidoro, 352-A, Cidade Universitária, Recife,
      PE, Brasil, ZIP 50.740-050.
      {rla2,gep,glr}@cin.ufpe.br
[‡]   Laboratório de Técnicas Inteligentes, Escola Politécnica da
      Universidade de São Paulo, Av. Prof. Luciano Gualberto, 158,
      CEP 05508-900, São Paulo, SP, Brasil
      http://www.lti.pcs.usp.br
      {jomi,jaime}@pcs.usp.br

Abstract. The recent development of software platforms for cell phones and handheld computers, such as Java 2 Micro Edition (J2ME), has broadened application perspectives in this area. In fact, the developers can now write their own software to run in handheld devices, what was impossible a short time ago since the software and the platforms were proprietary. Among the myriad of applications for these devices, some of them are very complex and need the intelligent behavior typically provided by agents yet available. However, since J2ME is a very recent platform, there are no appropriate J2ME-based environments or tools for agent development. This paper describes KSACI, a pioneer tool providing communication infrastructure among agents running in handheld devices. KSACI supports KQML, as the outer language, and XML, as the inner one. A preliminary version of KSACI was implemented, extending SACI (Simple Agent Communication Infrastructure), a Java open-source communication infrastructure for desktop agents.

1 Introduction

This new century is witnessing a new trend in computer research: the pervasive computing. According to this trend, computation will be increasingly embedded in small mobile and connected devices, providing to users relevant information and services anytime and anywhere [1]. One of the recent efforts for the actual

implementation of computer pervasiveness is the specification and implementation of Java 2 Micro Edition (J2ME, also called KJava) [2][3]. J2ME slims down the Java Standard Edition (J2SE) [4] by removing or rewriting key parts of the core runtime environment in order to fit it into small devices. As an outcome of a consortium involving Sun Microsystems Inc. and the major telecommunication devices manufactures (e.g. Motorola, Nokia, Sony, Samsung and Ericsson), J2ME broadens application perspectives in this area. In fact, it allows the developers to write their own software for handheld devices, which was unfeasible before since the software and platforms for these devices were proprietary.

The integration of novel handheld devices' capabilities, such as location, storage, processing and communication, has opened a myriad of applications from which users can daily benefit. These applications may range from simple e-mail systems to complex applications, such as intelligent Personal Digital Assistants, interactive multiplayer games, e-commerce location-sensitive transactions systems, and so on. Some of these applications require the intelligent behavior typically provided by agents, since they must exhibit capabilities such as autonomy, goal-driven reasoning, reactivity, adaptation, as well as communication, coordination and cooperation with other software entities.

Let us consider, for instance, the case of a shopping center, where each shop may have an agent that uses its "spatial tracking" capabilities to identify whether passing by users carrying a handheld device with an embedded agent are regular clients of the shop. If it is the case, the shop agent can suggest some novel products that may interest the user. If the user's agent accepts the shop agent suggestion, he/she may use his/her handheld device agent to negotiate with the shop agent prices and payment methods, before deciding to buy the good. In another scenario, a user may ask his/her agent to book two tickets for the cinema tonight. Based on the user preferences regarding movie genders, schedules and movie houses, the agent could remotely check the guest's agenda about the available time and contact some entertainment brokering agents in order to choose the configuration which best fits the user's requirements. After booking the appointment, this agent would authorize the payment and then notify the user and his/her guest. When the user arrives at the movie house's entrance, the handheld device agent is triggered to ask the movie house agent to open the roulette for the two paid tickets.

In order to embed agents in J2ME-based handheld devices, it would be extremely helpful to reuse APIs devoted to implement some building-blocks agent's components, such as deductive inference mechanisms, as well as communication languages and protocols.

Unfortunately, since J2ME is a recently released platform, there are no appropriate J2ME-based environments or tools for agent development (neither for reasoning nor for communication) yet available. The problem of providing inference mechanisms is perhaps not critical at the moment, since several applications can be developed using reactive agents alone or coupled with a PC/Workstation server, working as a cognitive agent partner. In this case, there are some J2SE-based tools to support agent's deductive inference, such as JEOPS [5][6], JESS [7] InterProlog [8] and JavaLog [9]. There are also several tools written in J2SE that can serve as an agent communication infrastructure. However, the adaptation of tools from J2SE to J2ME is not straightforward and has not been done yet. For these reasons, our project

has been targeted to solve firstly the problem of providing agents communication infrastructure for handheld devices.

This paper describes KSACI, a pioneer tool which extends SACI [10][11] (a Java open-source agent communication API) in order to enable handheld devices embedded agents to exchange information and knowledge with other embedded agents or with agents located in desktop computers (PCs and Workstations).

This paper describes KSACI, a pioneer tool which extends SACI (Simple Agent Communication Infrastructure) [10][11] (a Java open-source agent communication infrastructure) in order to enable handheld devices embedded agents to exchange information and knowledge with other embedded agents or with agents located in desktop computers (PCs and Workstations). This tool works with an agent communication protocol called KQML [12][13][14][15] and with a content optionally written in XML. KSACI is already fully implemented and its preliminary test results, on cell phone emulators, are encouraging.

The next section enumerates some problems and requirements for the development of J2ME-based applications. Then, the KSACI architecture and its relationship with other technologies (XML, KQML and SACI) are presented. Finally, some conclusions are presented and future research directions are pointed out.

2 Requirements

The following sections enumerate some of the main requirements for agent communication in a handheld environment.

2.1 Programming Language

A programming language that supports the development of device-targeted applications is necessary. This language should be slim enough to meet the severe restrictions of memory, storage space and processing power. It should also be a non-proprietary language, so that applications can be developed with high compatibility among different device platforms like PDAs, cell phones and two-way pagers. In addition, it should make available all object oriented software engineering modeling characteristics like reusability, modularity and portability.

2.2 Agent Communication Language

In a Multi-Agent System (MAS), the agents solve problems collectively through coordination [17], cooperation [16] and/or negotiation [17]. Since they do not necessarily run on the same platform or architectural model, they need to share a common ontology and a common communication protocol, so that they can communicate effectively exchanging information and knowledge. The use of an Agent Communication Language (ACL) is particularly required in a highly heterogeneous environment, which is the case of handheld devices. ACL can provide high-level communication interoperability among these devices.

ACL is composed of an *outer* and an *inner language*. An outer language is independent of platform, content, ontology and network transport protocol and is responsible for encapsulate the main attributes of messages in the communicative act. The language where the content of a message is written is commonly called an *inner language* (e.g. Prolog, Lisp, XML and Java). The inner language should be structured enough to facilitate parsing and expressive enough to represent the required knowledge.

2.4 Communication Infrastructure

A communication infrastructure stands for a model that should be followed by a set applications or modules of, to guarantee some important requirements to developer. The main requirements that should be provided by an infrastructure are reusability, modularity, efficiency and facilities

In a handheld device environment, the agents are distributed into a wireless network and occasionally will have to interact with each other. The implementation of multi-agent systems in such environment is sometimes difficult, since the project designer should to be aware, besides the agent's intelligence features, of distributed programming technologies like RMI [18], CORBA [19], DCOM [20], etc.

It will be interesting to rely on an infrastructure that hides these complexities, and that could use a communication protocol and could be compatible with most of the devices. The infrastructure should be robust and easy.

3 Java 2 Platform, Micro Edition (J2ME)

An appropriate solution for the problem of a programming language is the use of the just released Java 2 Platform: the Micro Edition. J2ME (also called *KJava*) is a language that slims down the Standard Edition by removing or rewriting key parts of the core runtime environment to be able to fit in small devices. It is based on a model of three layers of software built on top of the host operational system (see figure 1).
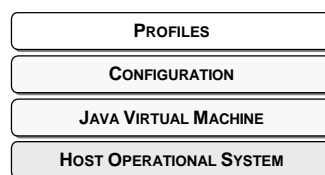
| PROFILES |
| CONFIGURATION |
| JAVA VIRTUAL MACHINE |
| HOST OPERATIONAL SYSTEM |

Figure 1: Three-Layered Model of J2ME

The *Java Virtual Machine Layer* is an implementation of a Java virtual machine customized for a particular device's host operating system and supports a particular J2ME ™ configuration. The *Configuration Layer* defines the "lowest common denominator" of the Java platform features and libraries that the developers can assume to be available on all devices. It defines the minimum set of Java virtual machine features and Java class libraries available to a particular "category" of

devices, such as medium devices (e.g. net TV, set-top boxes, network PCs) and small devices (e.g. pagers, cell-phones and smart-phones). *The Profile Layer* is the most visible layer to users and application providers. It defines the minimum set of Application Programming Interfaces (APIs) available on a particular "family" of devices. Profiles are implemented "upon" a particular configuration and typically include class libraries that are far more domain-specific than the class libraries provided in a configuration. Applications are written "for" a particular profile and are thus portable to any device that "supports" that profile. A device can support multiple profiles.

As discussed, to develop an application in J2ME, there are two main building blocks: the configuration and the profile. So far, there are two configurations already developed:

- **CDC** (*Connected Device Configuration*): targeted to shared, fixed and connected information devices like TV set boxes, Internet TVs, Internet-enabled screen-phones, high-end communicators, and automobile entertainment / navigation systems;
- **CLDC** (*Connected Limited Device Configuration*) [21][22]: targeted to personal, mobile and connected information devices such as cell phones, pagers and personal organizers.

The only profile developed so far was built on top of the CLDC: the **MIDP** (*Mobile Information Device Profile*) [23][24]. There are other profiles under development such as *PDA* (built on top of CLDC), *Foundation Profile* (built on top of CDC), *Personal Profile* (built on top of Foundation Profile) and *RMI Profile* (built on top of Foundation Profile) [3] (see Figure 2).

As described earlier, J2ME is a slimed version of J2SE, i.e. some J2SE functionalities are absent in the Micro Edition. Among these absences, the most important to the problem discussed in this paper are [3]:

- *No support for floating points.* Because the floating point operations are especially expensive without a dedicated coprocessor (thus, the virtual machine are not able to support any floating-point types or constants and nor any byte code involving floating point operations). This J2ME limitation has a strong impact on the way people code because the floating point is a so-used feature of all programming languages. Moreover, in an agent communication some message contents may be misunderstood if it refers to floating points values (e.g. in an e-commerce transaction, an agent could ask the price, commonly a float number, to an agent written in J2ME and it will not be able to understand).
- *No user-defined class loader.* For security reasons, an application cannot influence in any way how classes are loaded. Thus, the only class loader that is available to applications running on the KVM is the system class loader provided by the KVM itself. This means that an application written in J2ME cannot locate and load classes other than those from core runtime classes and the application own classes.
- *No reflection.* The absence of reflection eliminates some J2SE important features, such as object serialization. Consequently, the agents will not be able to use Remote Method Invocation (RMI) to transport their messages. All network transportation is done via HTTP sockets.

When developing a mobile application, one can run into problems on how to deal with the above J2ME limitations. The seemingly obvious technique is to take all missing J2SE classes and add to your J2ME application, but it is not a valid approach. Only the set of classes provided by J2ME Sun's Specification are available.
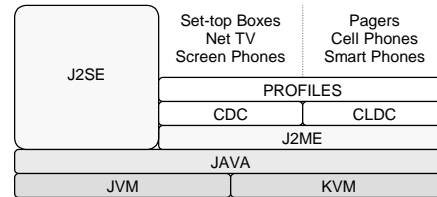

Figure 2: Java 2 Platforms

4 KSACI

This section describes how we have dealt with the J2ME restrictions previously discuss, in order to provide means for agents in handheld devices to communicate with other agents. In the first part we present a J2SE tool called SACI [10][11] that will serve as the agent communication infrastructure. After that, the KSACI architecture, which extends the SACI one, is presented, and a solution for the problem of the contents of messages is described.

4.1 SACI

SACI is a communication infrastructure developed by us, which attend to the requirements to run applications using the intelligent agents approach in a handheld device presented earlier in section 2.4. SACI works with the KQML ACL and compared with some analogous tools (e.g. Jackal, FIPA OS and JKQML), presents a good performance [10].

Besides the meeting requirements described above, SACI solves several problems regarding our required infrastructure. In fact, SACI has a society model based on client-server platform, where a society is composed of agents, one of them called facilitator or router. This agent is responsible to provide all the services of the society like white and yellow pages, registering and announcing skills. The other "common" agents function as clients of a SACI Society Server. The other important characteristic of SACI is that it provides support for both RMI and HTTP transport protocol, allowing its clients to be written in both J2SE and J2ME.

The internal behavior of SACI agents is simple. The agent has to **enter** into a society; then, **exchange** messages and **announce** skills; and at last, **exit** the society. Each society has one (and only one) facilitator (an agent with special features) to help the delivery of messages. The facilitator has a list of agent identification (and their respective network address) and a list of skills available in the society (and the respective set of agents that is able to perform those skills). catalize

As described earlier, a SACI agent should follow a predefined behavior. To **enter** or **exit** in a society, the agent should ask permission to the society facilitator. The SACI agent (including the facilitator) has an entity called *Mbox* (standing for Message Box) to queue the received messages for processing. An agent wanting entrance permission should send a message to the society facilitator that will capture the agent's name and network address, and generate a unique identification for that agent in the society. This identification (and localization) is stored in a list and will provide a service of *white-pages* (just like the one in a phone catalog). To an agent leave the society, it will have to follow the same process, but the facilitator will, in its turn, remove the agent's references from white-pages and yellow pages lists.

Once inside the society, the SACI agents may exchange messages with each other and/or announce skills to the facilitator. To **exchange messages**, the agent (A) should ask the facilitator for the other agent's (B) localization. Once received this localization (network address from white-page list), the agent pair (A and B) can communicate directly exchanging KQML messages (see Figure 3).
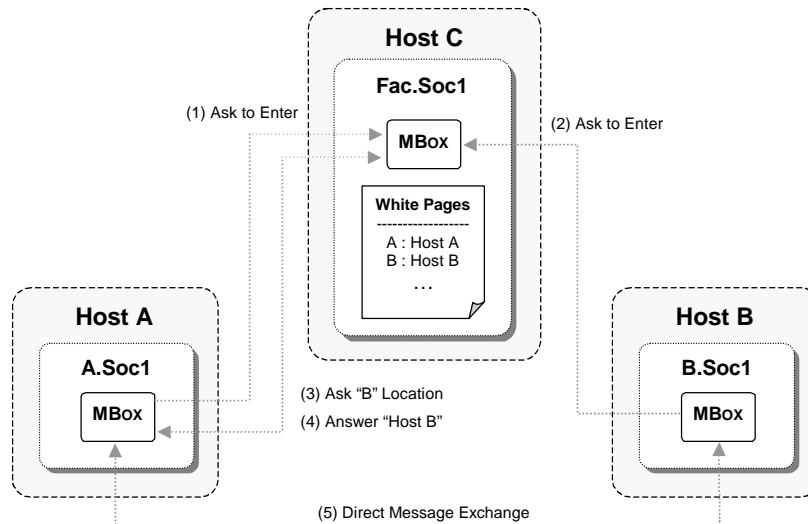

Figure 3- White Page Service

The **announce** of agent skills obey the same process of entering the society, but here the already set agent identification will be stored together with its specific skills in a list called *yellow-pages* that will provider the same service of a yellow-pages of a phone catalog where the subscribers are ordered by their skill (services). When an agent needs someone to do a specific job, it asks the facilitator for somebody who is able to perform it. After query the yellow-page list, the facilitator sends a list of agents back to the asker agent. By now on, the agent will ask the services directly to the provider agent.

4.2 SACI Extended Architecture

The SACI Extended Architecture (see Figure 4) is based on a client-server model where the server controls the main functions of the system (like the society ones), and the clients use the services provided by this server. The infrastructure for supporting MAS in handheld devices should have a server (SACI Server, written in J2SE) running on a desktop because of the restrictive limitations of these devices. In this architecture, there are three possible kinds of clients: those running on a desktop, those running on a handheld device and their respective proxy client. The Desktop SACI Client (written also in J2SE), shown in Figure 4, as the name suggests, is the module of the SACI framework responsible to define a specification for the clients running on a desktop. The other two modules are related. The Handheld KSACI Client is implemented in J2ME and has a respective proxy in the desktop side to manage the messages between device and desktop, since the protocol used in SACI environment is different from the one supported by J2ME.
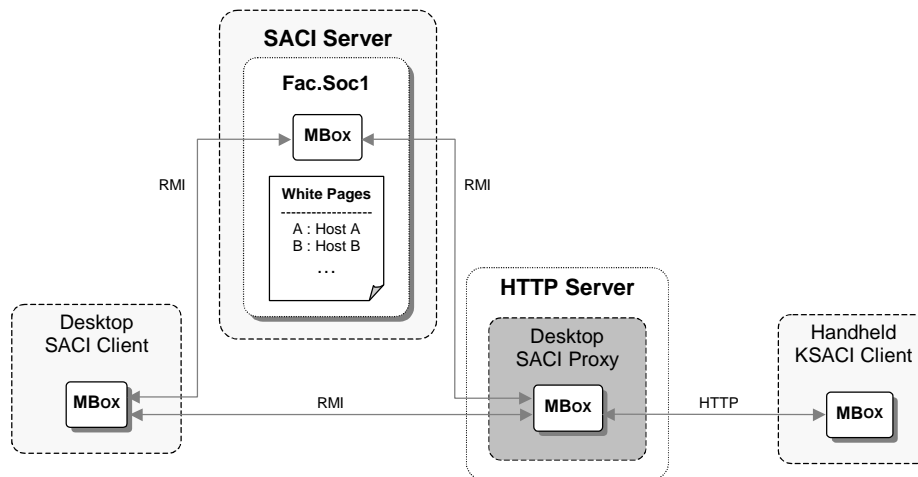

Figure 4 - KSACI architecture

The modules located in desktop communicate with each other using RMI. When a Handheld KSACI Client wants to communicate with these desktop modules, it has to send their messages through the Desktop SACI Proxy located inside a HTTP Server on desktop. This proxy is a special SACI Agent responsible for listening a TCP/IP port and redirecting the messages to the respective client or server on desktop via RMI. There will be a Proxy Client like this for each Handheld KSACI Client, since a proxy will not be able to decide which handheld client to send the messages. The Handheld KSACI Client will have to be continuously checking the proxy Message Box (MBox) for new messages from other modules (facilitators and agents), since the only protocol available for J2ME is HTTP and it is .

4.3 Transport Protocols

In the SACI Extended Architecture there are two kinds of transport protocols: RMI (Remote Method Invocation) and HTTP (Hipertext Transfer Protocol). The former one is used in communications between the three modules written in J2SE

(SACI Server, Desktop SACI Client and Desktop SACI Proxy). This protocol specifies that client Java applications can invoke methods on an object running in a remote server. The client uses a proxy object to represent the real, remote object. Because the proxy and the remote object implement the same interface, the application calls methods on the proxy as if it were directly calling methods on the remote object. RMI requires a Java Virtual Machine on both ends, and object serialization is used to pass parameters. This implies that a Desktop SACI Client can treat remote objects (located in another Desktop Client or in the SACI Server) as if it were present on the local VM.

The later protocol (HTTP) uses a reliable TCP/IP connection and the valid content types for it are non-structured text, hipertext and images. Handheld KSACI Clients are not then able to manage remote objects and it is necessary an alternative way to transport objects in the content of a KQML Message, since objects are very important in certain applications.

4.4 Agent Communication Language

The Outer Language chosen for our architecture was KQML because it was the ACL used by SACI and by several of MAS nowadays. KQML is a language and a protocol specification to support high-level communication among agents [12][13][14][15]. And as an Agent Communication Language (ACL), KQML enables agents to exchange information and knowledge.

As known, KQML does not specify the content of its messages. As SACI uses Java as its language, the content language of the KQML messages can be either java Strings or serialized java objects. The java string can represent any declarative language, such as KIF and Prolog. In some application, it is natural to use a serialized object as the content language, since it is not necessary to have a parser on both sides of the communication.

KSACI fully support string-based contents. As there is no Prolog or KIF-like interpreter already implemented in J2ME, the message contents tend to be unstructured or follow *ad hoc* syntaxes. To cope with this problem, we add to KSACI the capability of interpreting XML contents. We claim that, in the context handheld agents communication, XML [26][27] can play a major role as a content language because of three main reasons:

- its data is self-describing in an internationally standardized and non proprietary format;
- it is catalyzing the convergence of some terms, which may be the initial steps for a world-wide ontology in some domains;
- it is possible to implement a simple and efficient XML parser in J2ME.

In order to provide XML contents exchange in KSACI, we have used kXML [32], a XML parser for J2ME, on the KSACI side. On the SACI server side, we have used a J2SE XML parser named Castor [31].

5 Conclusions

KSACI is a pioneer effort to enable agent communication capabilities in handheld devices, using the technology state of the art. The KSACI Architecture provides an agent communication infrastructure for those who want to develop wireless applications for handheld devices. An agent located in one of these devices will be able to connect with a server and enter into an agent society, and communicate with other agents localized both in a PC or another handheld device. KSACI is already fully implemented and its preliminary test results, on cell phone emulators, are encouraging. Its main limitations are exclusively due to the current specifications of J2ME and related platforms and protocols.

Our future works include a systematic and quantitative evaluation of KSACI on real J2ME-based cell phones, which are not available yet. We also intend to port JEOPS [5] to J2ME, an object-oriented production system.

6 References

[1]   Andrew C. Huang, Benjamin C. Ling, Shankar Ponnekanti, Armando Fox. "Pervasive Computing: What Is It Good For?. In proceedings of the Workshop on Mobile Data Management (MobiDE) in conjunction with ACM MobiCom '99, Seattle, WA, September 1999 (forthcoming)

[2]   Java 2 Plataform, Micro Edition, http://java.sun.com/j2me

[3]   Eric Giguère, "*Java 2 Micro Edition: The Ultimate Guide to Programming Handheld and Embedded Devices*", Chapter 3. ISBN 0-471-39065-8.

[4]   Java 2 Plataform, Standard Edition, http://java.sun.com/j2se/1.3

[5]   JEOPS - The Java Embedded Object Production System, http://sourceforge.net/projects/jeops/

[6]   Filho, C. S. F., Ramalho, G. L., JEOPS – The Java Embedded Object Production System, in Proc. Lecture Note in Artificial Intelligence no. 1952.

[7]   JESS, http://herzberg.ca.sandia.gov/jess/

[8]   InterProlog, http://www.declarativa.com/InterProlog/default.htm

[9]   JavaLog, http://www.exa.unicen.edu.ar/~azunino/javalog.html

[10]   Jomi Fred Hübner and Jaime Simão Sichman, "*SACI: Uma Ferramenta para Implementação e Monitoração da Comunicação entre Agentes*", IBERAMIA, 2000

[11]   Jomi Fred Hübner and Jaime Simão Sichman, "*SACI Programming Guide*"

[12]   Yannis Labrou and Tim Finin, "*A Proposal for a new KQML Specification KQML*",. UMBC, Baltimore, 1997.

[13]   Yannis Labrou and Tim Finin, "*Agent Communication Language: the current landscape*", IEEE Intelligent systems, March/April, 1999.

[14]   Yannis Labrou and Tim Finin. "*A proposal for a new KQML specification*", UMBC, Baltimore, 1997.

[15]   Yannis Labrou, Tim Finin, and Yun Peng. "*Agent communication languages: the current landscape. IEEE Intelligent Systems*", 14(2):45–52, March/April 1999.

[16]   H. Haugeneder and D. Steiner. Co-operating agents: Concepts and applications. In N. R. Jennings and wooldridge, editors, Agent Technology Foundation, Application, and Markets, pages175-202. Springer-Verlag, 1998.

[17]   Weiss, G. Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence, The MIT Press, Cambridge, Massachussets, London, England, 1999.

[18] RMI (Remote Method Invocation), http://java.sun.com/products/jdk/rmi/

[19] CORBA (Common Object Request Broker Architecture), http://www.corba.org/

[20] DCOM (Distributed Component Object Model),
http://www.microsoft.com/com/tech/DCOM.asp

[21] Connected Limited Device Configuration (CLDC), http://java.sun.com/products/cldc/

[22] "*Applications for mobile Information Devices: White Paper*", Sun Microsystems,
Inc., 2000

[23] Wireless Technologies,
http://developer.java.sun.com/developer/technicalArticles/wireless/#midp

[24] Mobile Information Device Profile (MIDP), "*http://java.sun.com/products/midp*"

[25] "*Java ^{TM} 2 Platform Micro Edition (J2ME ^{TM}) Technology for Creating Mobile
Devices: White Paper*", Sun Microsystems, Inc., 2000

[26] The W3C XML Extensible Markup Language Working Group Homepage,
http://www.w2c.org/XML

[27] Benjamin N, Grosof, Yannis Labrou and Hoi Y. Chan. "*A Declarative Approach to
Business Rules in Contracts: Courteous Logic Programs in XML*". In Proc. 1st ACM
Conference on Eletronic Commerce (EC-99), Denver, Colorado, USA, 1997,
http://www.ibm.com/iac/ec99/

[28] Alvares, L. O., Sichman, J. S. "*Introdução aos sistemas multiagentes*". In:
MEDEIROS, C. M. B. (Ed.) Jornada de Atualização em Informática. Brasília: SBC,
agosto 1997. v. 16, Cap. 1, p. 1ss.

[29] Big in Japan, http://www.javasoft.com/features/2001/03/docomo.html?frontpage-
banner. Visited on march-29-2001.

[30] All about I-mode, http://www.nttdocomo.com/pr/recommend/d503i.html. Visited on
april-01-2001

[31] Castor, http://castor.exolab.org/

[32] kXML, http://www.kxml.org/

[33] TinyXML, http://www.gibaradunn.srac.org/tiny/index.shtml

[34] NanoXML, http://nanoxml.sourceforge.net/