Technological University Dublin

ARROW@TU Dublin

Conference Papers

School of Science and Computing

2003

Managing Content-Initiated Application Delivery With a Client-Side Agent

Niall Roche
Technological University Dublin, niall.roche@tudublin.ie

Gary Clynch
Technological University Dublin, gary.clynch@tudublin.ie

Recommended Citation

Roche, N. and Clynch, G. Managing content-initiated application delivery with a client-side agent. 2nd IASTED Conference on Communications, Internet & Information Technology, Scottsdale, Arizona, USA, 2003.

MANAGING CONTENT-INITIATED APPLICATION DELIVERY WITH A CLIENT-SIDE AGENT

Niall Roche
Department of Computing,
Institute of Technology Tallaght,
Dublin, IRELAND

Gary Clynch
Department of Computing
Institute of Technology Tallaght,
Dublin, IRELAND

Abstract

Mobile devices have evolved with improvements in processing power and support for various application environments such as MExE [1]. Currently a large number of devices and platforms exist, each with various attributes. Such diversity results in problems providing appropriate applications to these devices. This paper outlines need for an integrated approach to application delivery for mobile devices. The solution allows suitable applications to be delivered to devices. Application management is removed from non-technical mobile device users through the presence of a client-side agent to manage the complexity of application provisioning, device resource management and application management.

Key Words

Software Agents, Application Provisioning, Application Management, Mobile Devices, Resource Management

1. Introduction

Due to the rapid expansion in the use of mobile devices and the demands of mobile users for increased functionality, a new method of managing mobile applications is needed to meet these growing expectations. In addition, the large numbers of differing mobile platforms and operating systems require application developers and content providers to provide specific content and application solutions to specific devices. It is becoming increasingly evident that an integrated approach to content generation and delivery is necessary to manage these multiple device types.

Management of device specific issues such as differing platforms, operating systems, application environments and connectivity options should be left to mobile middleware and not involve content authors or end users. This paper describes a solution based on an intelligent client-side agent interacting with a provisioning server to provide deliver small applications and intelligent post-delivery management of delivered applications.

Section 2 of this paper describes the general problem area. The overall solution architecture is described in section 3 including a description of the client-side agent. Section 5 explains the current progress of the project with section 6 detailing the conclusions made in this paper.

2. Problem Area

At present most services available to mobile users are in the form of browser-centric applications, the user must request the service he/she requires from a remote server that processes the request, producing an appropriate response. In this approach, the majority of the service logic resides on the server and, as a result, is only available to the mobile user during connected periods.

Due to recent advances in mobile device technologies, such as the support for application environments for example Mobile Execution Environment (MExE) [1], devices such as PDAs posses the power necessary to process required services, or a part of the logic required for such services, at the client side. The opportunities for taking advantage of client side processing rather than simple content rendering have only recently been considered.

Current approaches based on MExE Class marks such as the Java 2 Micro Edition (J2ME™) [3] and other mobile application environments such as Microsoft's .NET Compact™ [4] utilise client side application environments to do processing either locally or in communication with a remote server, moving some of the logic to the client and thus permitting a degree of offline usage. These application environments operate by the delivery of small applications to the device on request from the user such as that described by OTA (Over the Air) delivery specification [5]. Delivery of these applications is a specialised task requiring knowledge of various possible requesting devices and their capabilities, standards and limitations while taking into consideration the low bandwidth plus limited and unreliable connectivity of mobile devices, when compared to desktop devices.

To address the issue of providing an architecture that effectively utilises client side processing and provides offline usage, requires extensive knowledge of device application and processing capabilities and is a complex task when compared with traditional content development techniques involving mark-up and server side scripting. A way of specifying the behaviour required of the device, rather than what specific application is for a specific device class, using traditional content generation techniques, could reduce complexity for content authors.

In addition to the issues associated with content generation, end users have an increased responsibility to be familiar with the processing capabilities of their device and the tasks required to initiate delivery, installation updating and removal of applications. This may be familiar to users of desktop computer environment, but may be difficult for users of mobile devices that are not familiar with such concepts. A way of automating negotiation of required applications and management of delivered applications and components that does not directly involve the user would be preferable. In addition if users could interact with the logic supplied in delivered applications via the use of a familiar browser-centric environment, this would reduce the requirements of the user to be familiar with the various device specific application interfaces, everything being available from the browser.

An example application could be a currency converter that uses the latest rates from financial markets, that is referenced in a content page to carry out conversion of amounts of various currencies specified by the user, calculating and outputting the results. The currency converter component could be provisioned and stored in a cache of components and could be used by several different content pages e.g. banking, travel and stock portfolio pages. The component could be implemented as a client application or applet, depending on device type or as a server side component where a device does not have the necessary resources to support an agent.

Other examples could be a form validation application that users validation components to ensure correct user input and a generic calendar process that utilises built-in calendar and appointment functionality present on many devices. In general components are modular in nature, to ensure maximum utility and reusability and where possible can take advantage of native functionality of the device.

Application could be designed in a modular fashion requiring the collaboration of a number of reusable components. Some of these components may already be present on the device, while others may need to be delivered to the client to execute the application. These components may reside on the device and operate under the management of software present on the device, as described in section 3.1

The practice of developing applications for delivery from content requires contributions from a number of sources. Component developers are required to create the components required to implement the application on a particular platform and have knowledge of the strengths and weaknesses of the target platform. Application administrators are responsible for gathering all required components and resources and packaging them for delivery. Content authors create content and embed generic references to applications in content (independent of the actual application that will be delivered).

3. Architecture Overview

To deliver appropriate applications to requesting devices a number of components, both client-side and server-side are required. This section outlines how the various components interact to achieve this task and proceeds to describe the client-side agent managing application delivery.

At the server-side, publishing components are required to allow content providers to upload application variants and their associated resources and components. Management components are required to decide what application variants are delivered to which device, and or, user. In addition, content authors require access to a content server on which to publish content with generic application references embedded in the content. At the client-side, an agent manages communication with the server to achieve effective application delivery and post-delivery execution and management.

In order to address the issues associated with provisioning applications and components to devices that support them and to provide an appropriate representation of the application to devices that do not support execution of applications, the adoption of standards was identified as a key enabler to a successful solution. Existing and emerging standards such as MExE [1], OMA Generic Download [6], JSR 124 [2] (Client Provisioning Framework) and UaProf [7] were considered to be of most value.

MExE (Mobile Execution Environment): A set of recognised class marks that define a set of recognised functionality that is known to be present on the device. This permits devices to be considered in terms of available functionality when determining which potential variants of an application could be supported by, and delivered to, the device.

The Open Mobile Alliance (OMA) Generic Download Architecture: an open standard identifying accepted practices of how resources such as applications can be delivered to a mobile device utilising existing standards such as HTTP [9].

User Agent Profile: a standard defined by the OMA for specifying device identification and device attributes such as hardware specifications supported content types and application environments present and is useful for determining what application variants and representations are supported.

The Client Provisioning Framework: an open API based on how to deliver bundles that can contain and applications to devices and currently being defined by the Java Specification Request (JSR) number 124. JSR 124 also identifies a customisable Matcher API that is useful for comparing available bundles with device attributes (specified by UA-Prof) to compute the most appropriate bundle to be delivered to a requesting device based on device attributes and user preferences allowing for graceful degradation of service.

Provisioning of applications components and resources to the device can be achieved using the JSR 124 API as a basis of communication between the agent and the provisioning server. JSR 124 allows standardisation in how developers specify what application variants and resources/components are appropriate for which devices. The approach is standards based and extensible to incorporate new devices and application types.

The JSR 124 API defines the notion of a delivery adapter that encapsulates the logic of how to deliver entities to a particular device using a particular protocol. A simple example is an adapter that implements the OTA standard to deliver MIDP applications to J2ME devices.

This abstract delivery adapter approach has been adopted for interaction with the client-side agent and a Provisioning Server (as defined by JSR 124) through an agent adapter, which encapsulates the logic of agent – server component and resource negotiation and delivery. The required resources and components are packaged in a download bundle multipart mime format using compression where possible.

The solution also requires a transformation server to transform generic references to applications contained to content capable of referring to applications or components available on the requesting device. A content transformer using information supplied by the agent, about the device capabilities, manages this transformation process.

The various components of the architecture are illustrated in figure 1.

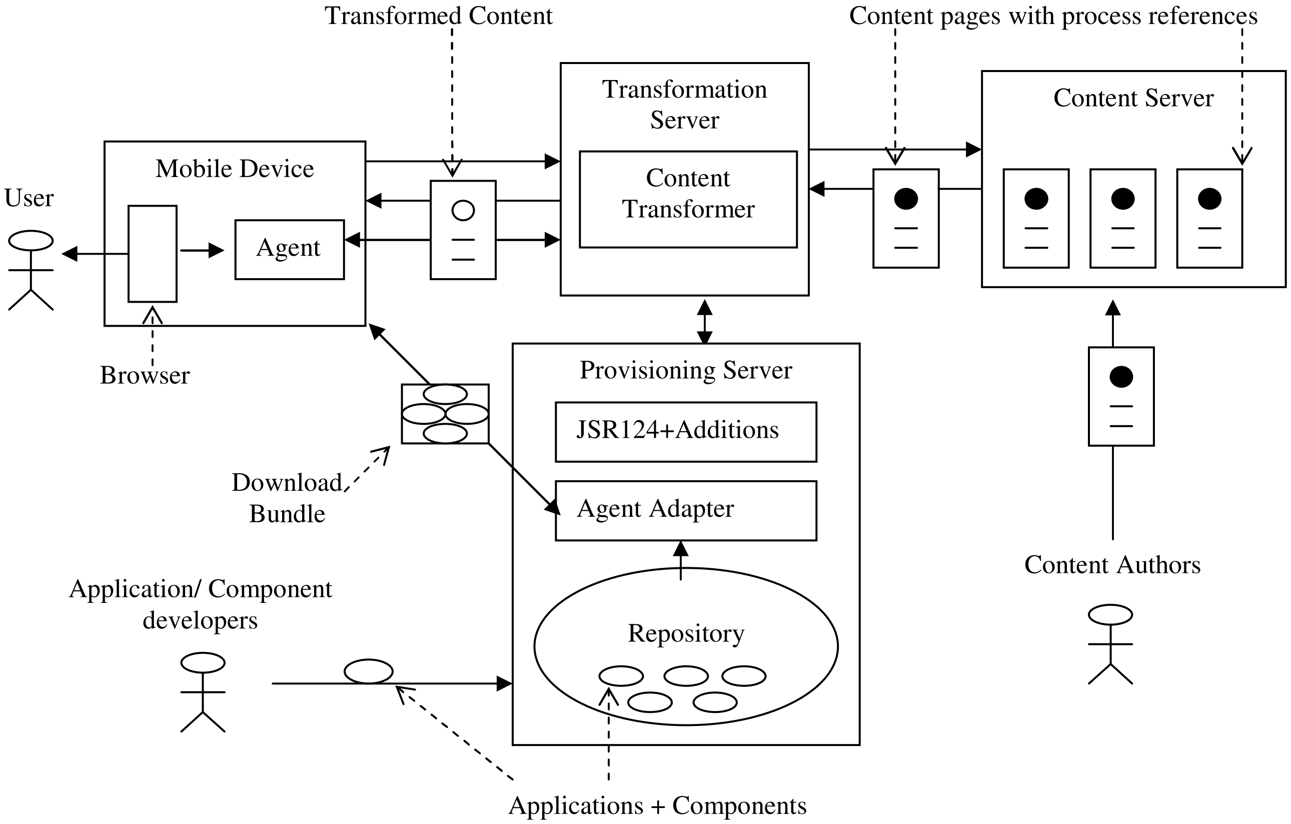

Figure 1: Solution Architecture

3.1 Client-side Agent

The key component of this architecture is a client-side agent that is required to automate the selecting, delivery, post-delivery execution, management and eventual removal of, applications, all in a manner transparent to the device user. The agent is necessary as without it the responsibility would be on the user to initiate application delivery, installation, management and removal, requiring knowledge of the device and the issues associated with application management on the device.

Once content containing an application description is returned to the mobile agent, the agent will determine if it requires resources or components to be delivered or updated. If items are needed then the agent will negotiate delivery of these items with the Provisioning Server. Once all application requirements are present on the device, the agent responds to user requests via the browser and manages the retrieval and updating of components in response to user actions.

3.1.1 Agent Description

The agent functions primarily as a resource manager for the host device, agents have advantages as a resource manager as mentioned in Bradshaw [9], having knowledge of the capabilities of the device and execution environment. The agent acts on behalf of the user to configure the device in order to use the application required by the content requested. The agent accomplishes this by managing device resources in response to changes in the device environment, e.g. changes in available memory.

A mobile agent can have advantages in the downloading of software as described in [10][11] in particular the automation of software deployment, installation and maintenance. The agent removes the responsibility of complex device configuration from a user by replacing it with agent intelligence. The user does not need to be concerned with what applications require what components/plug-ins and be required to endure complex installation and removal procedures. The agent will have the intelligence to know what components are required, how to get them and install and remove them.

The decision as to what components are to be stored and removed are based on the agent examining application usage patterns. The agent tracks component usage recording what components are used by each application and how often. When components need to be removed to make room for newer ones, the agent will make a decision based on a metric of how useful a component is. The metric is based on size required by the new components, the size of existing components and how often they are used. The decision is based on the best compromise of utility versus resource requirements and does not need user involvement in the decision making process.

The agent operates in a semi-autonomous fashion responding to user requests for content and content references to applications and components. In addition, it can act according to external events such as component/content expiration upon which it can obtain the latest version independent of user action. The agent can also respond to other events such as messages received from the provisioning server. The agent functions in both connected and disconnected scenarios fetching components and checking for updates when a connection is available. In order to preserve device resources, the agent may function continuously or may suspend operations if not required for long periods with operation resumed in response to events.

The interface between user and various types of agent has been described in various research [12][13]. The design of the agent as described in this solution makes use of the browser present on the device to communicate with the user. A user may not necessarily know that they are communicating with an agent, as they get a similar, though more powerful user experience similar to making a request to a regular remote web server. This approach removes the responsibility for interface and presentation logic to be part of the agent instead moving it to a medium more suitable and familiar for the user. An important aspect of this approach is the reduction of the impact of implementation specific user-interface features facilitating a multi-platform and implementation language neutral agent design.

3.1.2 Agent Functionality

Local server functionality: The Agent functions as a local web server on the device, browser requests for content are made to localhost and are converted to corresponding remote requests, once the requested resource is obtained it is returned to the browser. This approach allows a limited degree of mobile browsing in an offline scenario.

Component Management: The Agent maintains a cache of components that are available to a number of applications. The cache is updated according to space available on the device and in response to component usage patterns.

Application Management: The Agent is responsible for determining what components/resources an application needs to execute. The Agent will then compare cached components/resources and will negotiate delivery of resources/components from the provisioning server.

Version Management: If a component present on the agent has expired, or the agent determines that a newer of a component is available, it can obtain the latest version. The agent resolves compatibility issues such as a particular application requiring specific component versions, ensuring backward compatibility.

4 Agent Interactions

The following diagram (figure 2) illustrates a sample interaction between the user via a browser and an agent in order to download content requiring components.

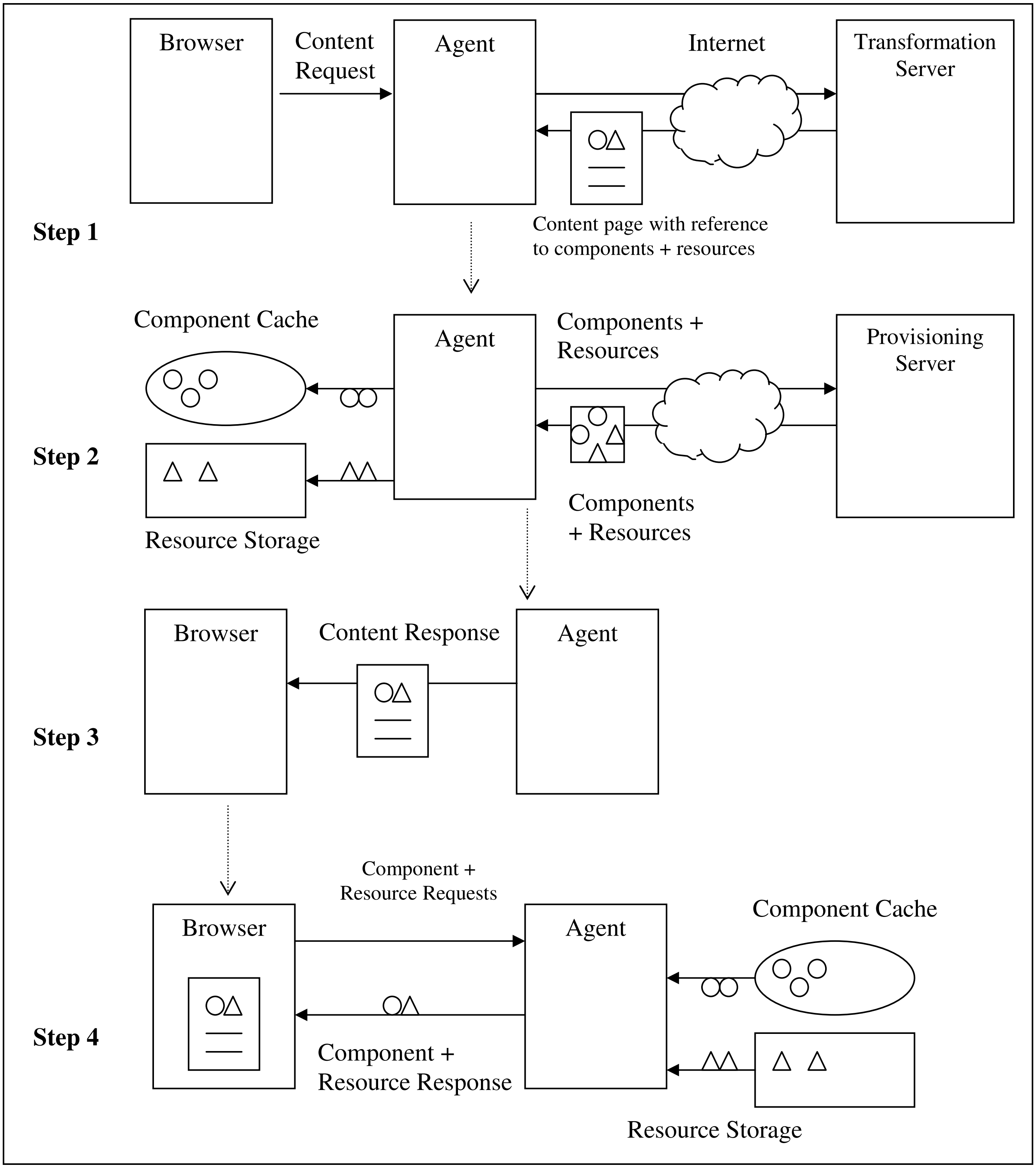

Figure 2, Agent Interaction

4.1 Agent Interaction

The interaction of the agent with the device browser and remote servers to deliver applications is illustrated in Figure 1 and can be outlined as follows:

Step 1: The device browser requests content from the agent, if a connection is available, the agent requests the latest version from the transformation server, which obtains and suitably transforms the content based on the capabilities of the requesting device, as identified by the agent. The server returns the content to the agent. E.g. a travel site requiring a currency converter component.

Step 2: Should all required resources and components not be present in the cache, or be up to date, the agent requests delivery from the provisioning server. The provisioning server responds with the required resources, components and, if required, updates. The new components are stored in the cache then installed/registered and made ready for use on the device. E.g. agent downloads suitable currency converter application from provisioning server.

Step 3: Once all application requirements are present and up to date, the content is returned to the browser for processing. E.g. content displayed with browser link to launch installed currency converter application.

Step 4: The agent handles requests from the browser for resources and components. Components either execute within the browser or are activated through the browser and execute externally. Finally, the agent will manage updating and removal of applications and components as required. E.g. agent checks for currency rate changes and updates local rates.

5. Status of Research

At present, a number of prototypes are being developed including agents for Windows™ notebook and Pocket PC™ 2002 clients using the .NET™1.1 and .NET Compact™ Frameworks [4], with a Symbian OS 7 deployment using PersonalJava™ 1.2 [14] under investigation. Sample applications include currency conversion and instant messaging applications. A server solution has been implemented in Java using Tomcat™ 4.1 with device identification and content mapping helped through the Everix™ mobility platform from MobileAware Ltd. (www.mobileaware.com).

6. Conclusion

In conclusion, to effectively address the issues associated with different mobile devices and platforms, an integrated approach to content and application management is necessary. An integrated approach is required due to the increasing responsibilities of content authors and application developers to create suitable content and applications for an expanding and increasingly diverse set of devices.

The solution described in this paper defines an integrated solution to addressing these issues, with mobile middleware removing the responsibilities from content authors and application developers. The solution provides an approach that separates the intent of the content author from the low-level implementation details.

A client side agent, where feasible, can be deployed to a device. The agent automates issues such as deployment and management of mobile applications that embody the specialised content the user requests. This agent in cooperation with a provisioning server, negotiate delivery and post execution management of applications and its required components, ensuring the user has the most current version of each.

References

[1] MExE Forum. "MExE Mobile Execution Environment White Paper", MExE Forum white paper, December 2000.
[2] Java Community Process. "JSR 124 J2EE Client Provisioning Specification", JCP Specification, May 2001.
[3] Java Community Process. "JSR 37 Mobile Information Device Profile for the J2ME Platform", JCP Specification, September 2000.
[4] Andy Wigley, Stephen Wheelwright, ".NET Compact Framework", Microsoft Press, 2003.
[5] Sun Microsystems, "Over The Air User Initiated Provisioning for Mobile Information Device Profile", May 2001.
[6] Open Mobile Alliance, "Download Architecture Version 1.0" OMA Technical Specification, June 2002
[7] WAP Forum, "WAG UAProf" WAP Forum Technical Specification, October 2001.
[8] Fielding, et al, "Hypertext transfer Protocol -- HTTP/1.1", RFC 2616 Internet Official Protocol, June 1999.
[9] J. Bradshaw *Software Agents* (Menlo Park, CA: AAAI Press/The MIT Press, 1997).
[10] S. Krause, T. Magedanz, Mobile Service Agents enabling Intelligence on Demand in Telecommunications, *Proc. IEEE GLOBCOM'96*, London GB, 1996, 78 – 85.
[11] O. Fouial1, N. Houssos, N. Boukhatem, Software Downloading Solutions for Mobile Value-Added Service Provision.
[12] P. Mihailescu, MAE – Mobile Agent Environment for Resource Limited Devices.
[13] A. da Silva, M. da Silva, A. Romão, Web-based Agent Applications: User Interfaces and Mobile Agents
[14] Sun Microsystems, "PersonalJava Application Environment Specification", August 1999.