Building Flexible Mobile Applications for Next Generation Enterprises

Karun Karunanithi, Khurram Haneef, Bruno Cordioli, Amjad Umar, and **Ravi Jain.**

Telcordia Technologies Inc.
445 South Street, Morristown, NJ, 07960-6438
Phone: 973-595-7990 Email: *karun@research.telcordia.com*

ABSTRACT

In order to understand and gain practical insights into various aspects of Next Generation Enterprises, we are building a testbed for flexible mobile applications. We started with a generic architecture and used it to build different applications in different domains. Our testbed consists of a multi-tiered architecture incorporating various user interfaces and access technologies, middleware components, enterprise integration systems, supply chain management and a host of currently emerging technologies. The first application we have developed is an information application for Olympics to provide personalized information for mobile users through a variety of end-user devices. The second experiment is a typical e-commerce online purchasing application that uses mobile agents (brokers) to buy products from multiple suppliers based on user specified criteria. The third example demonstrates a typical workflow scenario that may be needed in a highly collaborative and reliable NGE environment. In this paper, we present our preliminary experiences in building these applications. Based on our experience, we are pursuing further research and evaluation of various technologies for flexibility, mobility, scalability and ease of integration.

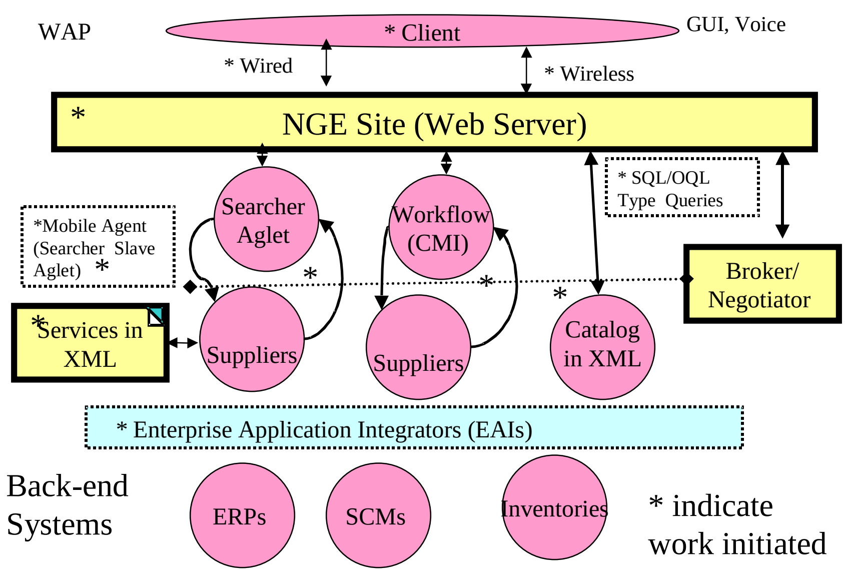
1. INTRODUCTION

Next Generation Enterprises (NGE) that are currently emerging can be characterized as follows: 1) use internet/intranet as corporate wide communication infrastructure, 2) integrate multiple vendors and suppliers through a zero latency supply chain management, 3) integrate back end legacy systems, ERP and other systems using EAIs (Enterprise Application Integrators), 4) implement virtual inventories and virtual offices, 5) utilize host of middleware components to access back end systems from application clients, 6) facilitate dynamic service creation and disassembly of services when not needed, 7) allow automated contract negotiation and match-making, 8) incorporate integrated billing, customer care and decision support systems, and 9) allow both wireline and wireless (web) access to users from desktop computers, PDAs and cell phones or traditional phones and/or any other custom designed access devices. One of the important classes of NGE applications is mobile application, which typically have one or more of the following features:

- Mobility of customers, employees, and partners over wireless networks.
- Mobile agents to handle mobile/nomadic platforms, software, and data
- Advertise, negotiate, and settle services and deals through XML
- Specialized middleware for wireless
- Virtual enterprises to support virtual inventories and virtual offices.
- Dynamic workflow and high level of collaboration
- Integration of services from multitude of suppliers (EAI, ERP).

To gain practical insights into building mobile applications, we started with a generic mobile application framework that can be customized for different business segments. This framework, shown in Figure 1, consists of a multi-tiered architecture. This framework introduces a wide range of research and operational issues in integrating wireless networks, mobile agents, wireless middleware, workflow, data replication, XML, adapters, virtual operations, trading hubs, etc. While building our testbed, we used an incremental approach to add building blocks and related technologies as when needed. We first developed an OlympicInfo application, described in section 2, that has only one information store (an Oracle Database that contains the Olympic information) and a wide range of information users. In this application we deployed a variety of nomadic support middleware components. This experiment showed us how new middleware can be used to adapt for wireless situations. In section 3, we extend this model to a mobile e-commerce application with multiple stores (XML-based content) and brokering through mobile agents. In Section 4 we describe an implementation of dynamic workflow model to highlight high-level collaboration between a broker agent, various suppliers and buyers. Section 5 concludes this paper by discussing the lessons learned and outlining future areas of work.

Figure 1: NGE Prototype Architecture



2. A Mobile Information Service Application

Based on our generic NGE framework shown in Figure 1, we have developed a mobile information application called OlympicInfo (Figures 2 and 3). This mobile application was designed for a large number of visitors expected at the Olympics. OlympicInfo allows a user to register which of the many Olympics sports events are of interest to him or her, and receive up-to-the-minute information relevant to those sports before and during the Olympics. The information to be delivered could include the scheduled times, contestants and venues of events for each sport of interest, changes in times and venues, the results of events as they occur, directions to travel to the events by foot or by buses provided in the Olympics Village, etc. In addition, from time to time OlympicInfo will provide the user some general informations e.g. weather forecasts and possibly advertising messages from the service provider or other sponsors. The user can choose to receive this information by a variety of media, e.g. pager, fax, screen phone, or PDA, although obviously some types of information (e.g. images) can only be delivered if the user has a terminal device of the correct type.

OlympicInfo offers users a variety of ways to enter their choices about sports and events they are interested in, and these choices are stored in user *profiles*. These include operator-based methods where the user calls an (800 or 900) telephone number and speaks to a live operator, or obtains and faxes back a form to a clearinghouse, Web-based methods where a user enters a form and entry via OlympicInfo information kiosks.

Figure 2 shows the OlymicInfo prototype. Two classes of clients are used in the prototype. “Thin” clients are those that have limited computing, display and communication capabilities. In the prototype these are limited to one-way alphanumeric pagers, PCs or digital cellular handsets with small screen displays, and browsers that can display Handheld Device Markup Language (HDML) cards sent using the Handheld Device Transport Protocol (HDTP). For the purposes of this prototype the main difference between thin and thick clients is that the latter can run a CORBA Object Request Broker (ORB).

Figure 2: OlympicInfo System

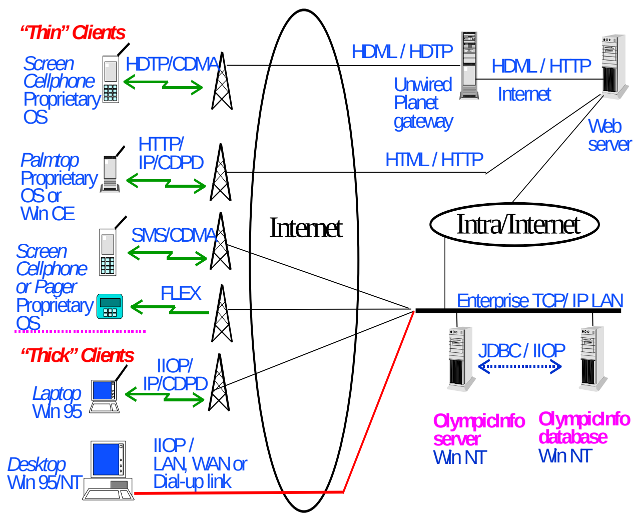
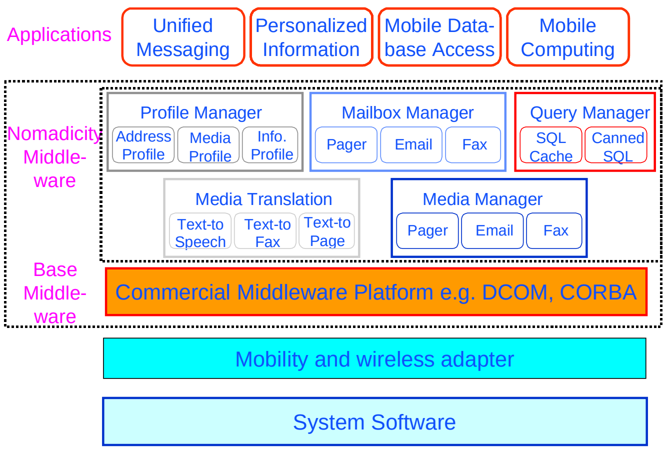


Figure 3: General software architecture for OlympicInfo



Example of thick clients that will operate in the prototype are laptops running Windows 95 that can be attached to CDPD wireless modems and which can run a CORBA ORB, specifically Inprise's VisiBroker ORB. The clients run a CORBA application written in Java that communicates using CORBA's Internet Inter-ORB Protocol (IIOP) over TCP/IP, which in turn will run over the wireless link.

The prototype server resides in an enterprise TCP/IP LAN environment and consists of a Windows NT platform running a CORBA ORB, specifically Inprise's VisiBroker ORB. The application is a CORBA application written in Java. The actual information is stored in a database (an Oracle version 8 database) that runs on a Windows NT platform running a CORBA ORB with access to the database using Java Database Connectivity (JDBC) APIs.

Figure 3 shows the software architecture of OlympicInfo. This architecture is designed to support a variety of applications such as unified messaging, in addition to mobile applications -- the focus of this paper. The architecture includes an extensive middleware stack for nomadic users that need support for profiles management, mailbox management, and media management/translation. The nomadic middleware stack at present runs on top of basic middleware such as CORBA. Adapters for wireless and mobility are included in the architecture but were not implemented (these adapters are expected to be provided by Information Distribution Manager [1] and Wireless CORBA). Details of the OlympicInfo software architecture are beyond the scope of this paper (see [2] for details).

3. A Mobile e-Commerce Application

In this application, we further extend the general mobile e-commerce application framework in several ways: multiple suppliers, mobile agents, and XML content representation. As part of this extension, we evaluated Voyager [4] and Aglet [3] platforms for mobile agents (we chose Aglet due to its openness), developed an Aglet application that goes around multiple sites and chooses the best deals for the customers, constructed user interfaces, and specified a wide range of products in XML. We are currently investigating WAP(Wireless Application Protocol) and voice processing as further extensions.

Figure 4 shows the mobile e-commerce aspect of the mobile application. It consists of 3 shopping sites:

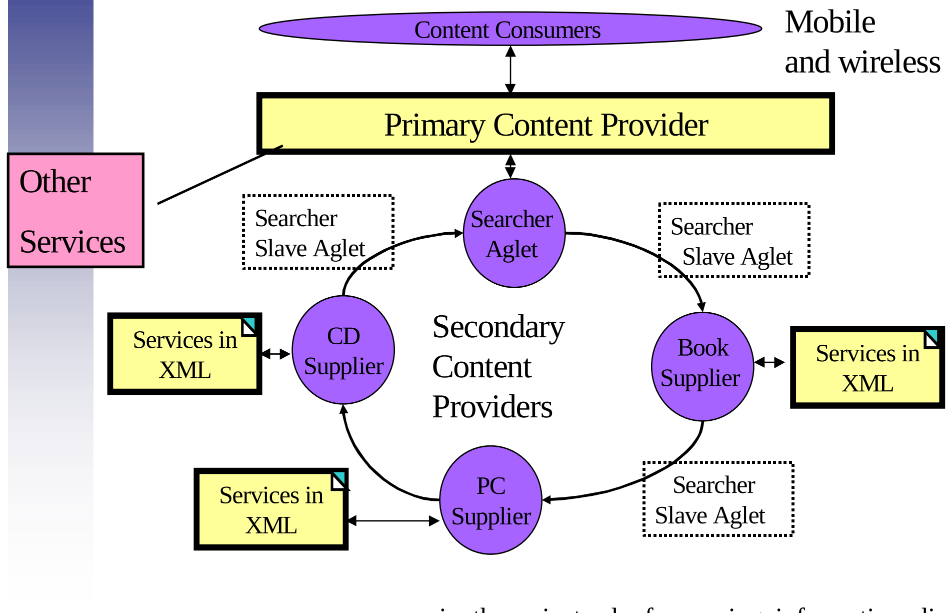
- "BookStore" selling books and CDs
- "CDStore" selling CDs and software
- "PCStore" selling hardware and software

The products that each of these site sells are defined as XML files. At each of theses sites, an Aglet server runs a "Supplier Aglet". The Supplier Aglet reads the XML file corresponding to its site, and parses it using the Sun's Java XML Standard Extension API (see http://java.sun.com/products/xml/). Once the XML file is parsed, it resides as a Java object in the Supplier Aglet. In addition to its products, each site defines its selling policies, also in XML, and store policies (i.e., a return/exchange policy, and credit cards the site accepts).

The buyer's interface to the demo is through a web page. In this web page, the buyer specifies the product it is interested in, in addition to the policies that the buyer desires. Once the client specifies, and submits its request, the request is handed over to a "Searcher Aglet" in the form of a CGI string. The Searcher Aglet, which has the capability of accepting http requests as messages, parses the CGI string to obtain the unique code of the product in the request, and creates a "Searcher Slave Aglet" that compares the user query with the local Supplier Aglet to determine a match. Once the Searcher Slave Aglet obtains the query results, it dispatches to the next site, and so on. The Searcher Slave Aglet maintains an itinerary that it should follow to various sites. In the case where a site on the itinerary is not up, the Searcher Slave Aglet will skip that site. Once the Searcher Slave Aglet has completed its trip, it returns to its original site and delivers the search results to the Searcher Aglet. The Searcher Aglet, in turn, embeds these results in a dynamic HTML page that it creates, and then writes that HTML page back to the browser in a manner similar to CGI applications.

Note that the communication between the Searcher Aglet and the Searcher Slave Aglet, as well as that between the Searcher Slave Aglet and the Supplier Aglet is carried locally through the Aglet messaging mechanism, and thus it does not consume network bandwidth. Unlike classical client/server database applications, the connection between the database client (in this case the Searcher Aglet) and the server (in this case the Supplier Aglet), need not be established while the search is being performed. This characteristic is especially important in the case of wireless network, since the nodes may get disconnected frequently.

Figure 4: Mobile Ecommerce Prototype Architecture



Instead of maintaining a stationary Supplier Aglet at each site, we might have allowed the Searcher Slave Aglet to access the XML files directly. However this design option provides at least three advantages:

- *More secure*: the Aglet security model differentiates two kinds of Aglets:

i. Trusted Aglets: downloaded from a local codebase or a trusted host. These Aglets have access to the local resources of the host, (files, etc.)

ii. Untrusted Aglets: downloaded from an untrusted host. These Aglets have limited access to local sources, but they can communicate with Aglets on the host.

From the supplier's point of view, trusting a Searcher Slave Aglet represents a security threat. The Supplier Aglet, therefore acts as a gateway between the searcher and the resources that contain the services.

- *More efficient*: allowing the Searcher Slave Aglet to perform the query search will require it to be more complicated, and thus larger in size. This, in turn, will cause its dispatching to consume more bandwidth, and be less efficient.

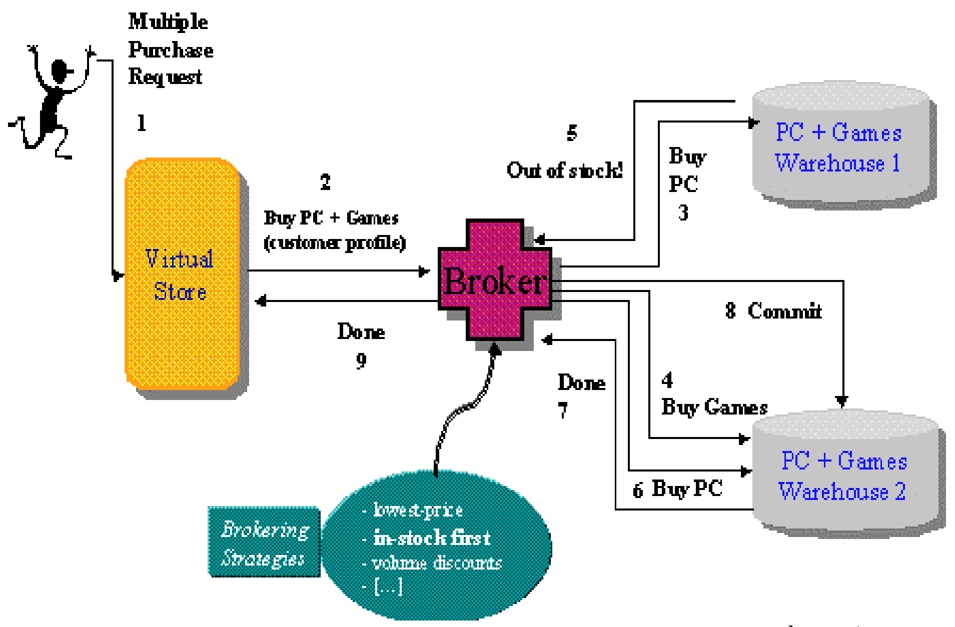
- *More modular*: consider the case where the service implementation at each site is different. This requires a different interface to the service at each site. This heterogeneity should reside in the individual stationary (Supplier) Aglets instead of residing in the roaming Aglet. This remark illustrates a design principle that, we believe, every mobile agent application should follow: target sites should provide stationary agent to interface with mobile agents, and thus hide the complexity required in local computations in those instead of exposing information directly to the roaming agents.

4. Dynamic Workflow and High-level Collaboration

In this example we track the order flow of a typical consumer purchase from the client end to various back-end subsystems and provide a high-level collaboration between the buyer and the system via a broker agent. This system consists of a virtual store, a broker and two warehouses (suppliers). The virtual store is implemented as a web server though which a customer can order goods (in this case PCs and Games) using a browser interface (Java). The virtual store does not maintain its own inventory and it fully relies on its suppliers. For each registered buyer, the store has a profile maintained in a local database.

This profile typically consists of some payment information and preferences. The suppliers maintain their own inventory of various brand name PCs and Video Games. Each warehouse has its own demand forecast, pricing and delivery options. The broker in the middle acts as an agent to negotiate and match a best price and/or other criteria. When a purchase order comes to the server, the server sends the request to the broker along with the buyer's profile. The broker will then match the order (along with its buyer's constraints) with designated suppliers' ability to meet the order. It should be noted that the broker has its own discretion to deal with partial purchases, if necessary, from different suppliers and then package a complete order fill to the buyer. Figure 5 illustrates a typical scenario of this example where a buyer makes a purchase of a PC and Games. All work flow steps involved in this typical purchase are labeled 1 through 8 in Figure 5. In this example scenario, Warehouse 1 ran out of PCs and the broker then purchased all items from Warehouse 2. Once broker gets confirmation from the Warehouse(s) of all items matching buyer's constraints it will issue a 'Commit' and relay that as a 'Order Confirmation' to the buyer. In actual prototype, we have implemented several scenarios involving different levels of interaction between the buyer, broker and suppliers. Usually, interaction occurs between the buyer and the broker. But in certain situations, for example, the order may have to be shipped by the Warehouse to a different address than that of the buyer and the Warehouse may need a shipping Zip code for optimum shipping cost or delay. Is such a scenario, the Warehouse may directly prompt the buyer to enter additional information. It is also possible that the buyer may cancel the order at any time (before the final 'Commit'), even if the broker finds a matched order.

Figure 5: A Typical Workflow Scenario



In our implementation, both the virtual store server and the broker reside on the same server while the warehouse servers reside on different machines. The communication between the broker and the warehouse is implemented using special protocol on top of Unix Pipe. Each warehouse maintains its own database and the warehouses are implemented in TCL. This example illustrated that in emerging NGE platform, workflow is an important aspect if we have to assure quality of service, reliability and high level of customer satisfaction.

5. Lessons Learned and Future Directions

We have learned a great deal by developing this testbed. We have mainly learned that at present the application designer has to know apriori whether the application will run on a wired versus wireless network. This is why the IDM capabilities of adapting for wireless networks transparently are quite useful. Other problems, beyond the scope of IDM, also need to be considered. For example, we found that the Web Browsers for hand-held devices are not Java-enabled and also do not support JavaScript and Frames. Thus a single wireless user can force change/re-structure of web interfaces (we had to redesign our Web site for wireless users because we initially used JavaScript to access XML documents). In addition, we found unique firewall issues due to wireless. For example, our Web site is within Telcordia’s firewall so we could not access our web site using Wireless IP Modems (in order to make it happen we need to put our web server outside Telcordia’s firewall). The Nomadic Middleware discussed in the OlympicInfo application addresses these issues. In the long run, standards like WAP and Wireless CORBA should address these problems.

We have also learned design principles that mobile agent application developers should follow. For example, mobile agents are particularly suitable for wireless networks because they can find their way on a frequently disconnected network (i.e., a mobile agent can stay at a node until a suitable connection is found). In addition, the target sites should provide stationary agents to interface with mobile agents and thus hide the complexity required in local computations by roaming agents.

We have also found XML to be quite effective in specifying information to be exchanged. In particular, the behavior of the participants can be represented by using scriptlets in XML

From our workflow example, we found out that giving proper feedback to the buyer about the order progress is very important. Furthermore, providing facilities to monitor order flow and, if necessary, allow the buyer to intervene the order flow to correct or change the order is very valuable. Also maintaining a log of order flow steps and interactions is very valuable for post analysis and data mining.

We are considering following future directions:
1. Investigate the role of emerging standards such as WAP and Wireless CORBA on mobile applications in the army environments.
2. Research into integration with back-end systems through Enterprise Application Integrators.
3. Investigate suppliers running on mobile devices. This raises many issues. First, if the device cannot run a standard Java VM, then an alternative such as Java for Windows CE must be found. Second, in this case, the mobile agent infrastructure should be able to take advantage of wireless middleware.
4. Allow the mobile agents to perform sophisticated contract negotiations and trading/brokering with the Suppliers.
5. Build analytic models to decide under what conditions what approaches (e.g., mobile agents) should be used.
6. Investigate the role of an Open API-based compensating middleware package such as IDM in building mobile applications quickly.
7. Investigate the role of automated code generators for automatically generating the adapters needed for different aspects of mobile applications. The code generators should accept, for example, UML information and generate adapters.

Overall, we have found that a testbed that can be used to build generic mobile applications to be an effective research and learning tool. We plan to conduct research towards a knowledge-based workbench with open APIs for building large scale mobile applications of the future. At the core of the workbench will be a generic application model, somewhat similar to the one described in this paper, that will be used to quickly generate different applications through inheritance/specialization. The workbench will provide a set of components (e.g., Java Beans, Enterprise Java Beans), adapters to hide the wireless versus wireline issues and the necessary tools to compose mobile applications and to reason about them. In particular, this workbench should provide intelligent decision support facilities for helping to decide, for example, when to use mobile agents and when not to, and what type of data conversions, routings, and workflow capabilities will be needed for the target application.

REFERENCES

[1] Umar, A. et al, “An Open API for Information Distribution in Mobile Environments”, submitted to ATIRP 2000.

[2] Jain, R., et al, “OlympicInfo Architecture", Telcordia Document, 1999.

[3] Aglet API documentation and architecture, http://www.trl.ibm.co.jp/aglets/documentation.html.

[4] Voyager, Application Server Platform, http://www.objectspace.ibm.co.jp/aglets/documentation.html

Trademarks:

The following are trademarks of respective companies: OlympicInfo - Telcordia Technologies Inc., Aglet – IBM Corp., Java – Sun Microsystems Inc., Windows 95 and Windows NT – Microsoft Inc.

BIOGRAPHIES

Karun Karunanithi is a Research Scientist at Telcordia Technologies since 1992. He received his PhD from Colorado State University in Computer Science. He has published in several IEEE journals and conferences. His current research interest are in E-Commerce, Enterprise System Integration, Specialized Portals, GUI using advanced visualization technologies.

Amjad Umar is the Director of the Advanced Distributed Systems Group at Telcordia Technologies and an Adjunct Professor at Rutgers University. His more than 20 years of experience includes software development, research, management and consulting assignments in the telecommunications industry, manufacturing organizations, educational institutions, and organizations in England, Singapore, China, Italy, Argentina, and Canada. He is the author of three Prentice Hall books: "Application (Re)Engineering: Building Web-based Applications and Dealing with Legacies", "Object Oriented Client/Server Internet Environments", and "Distributed Computing and Client-Server Systems". He has a Ph.D. in Information Systems Engineering from the University of Michigan

Ravi Jain received a Ph.D in computer science from the University of Texas at Austin in 1992. Prior to that he worked for several years on developing communications and systems software, performance modeling and parallel programming. Currently he is director of the Middleware and Mobile Applications Research group at Telcordia Technologies. His interests include programmability, middleware and applications for next generation networks, mobile Internet access and applications, and mobile and wireless networking

Khurram Khaneef is a research scientist at Telcordia Technologies. His Research interests are CORBA and JAVA as applicable enterprise and wireless systems.

Bruno Cordioli is a student visitor from Italy.