

DEVELOPMENT OF MOBILE DISTRIBUTED APPLICATIONS

Eugene M. Burmakin, Prof. Juha O. Tuominen

*Helsinki University of Technology
Department of Computer Science and Engineering,
Industrial IT Laboratory, PL9700, 02015, Espoo, Finland*

eugene.bourmakine@hut.fi

juha.tuominen@hut.fi

Abstract

Object-oriented analysis and design of mobile context-aware applications is discussed from the formalizing of the development process viewpoint. Solutions to several challenges of mobile distributed systems by means of applying design patterns and CORBA middleware technology are also presented in the paper. The presented results are based on a design example of a context-aware application operating in a mobile environment.

Keywords

Distributed Systems, Mobile Computing, Design Patterns, CORBA, Context-Awareness

1. INTRODUCTION

The market of mobile handheld devices and mobile application is growing rapidly. Mobile terminals are becoming more capable of running rather complex applications, due to the rapid process of hardware and telecommunication technologies. There is a need to develop new mobile information infrastructures with applying emergent standards and techniques in order to make 2.5-3G mobile technologies more attractive to the public. These mobile information systems have a big impact to the every-day life in business, education, research, entertainment etc.

From the technological point of view, these information infrastructures are distributed computing systems. They enable timely access to information, a collection of data, online and offline collaboration between people, performance of control tasks, etc.

One of the interesting types of applications is a context-aware [9] system that updates its behavior according to its operating environment, such as the mobile positioning-aware application presented in the paper. Within this paper, object-oriented analysis and design of mobile context-aware application is discussed from the formalizing of the development process viewpoint. Solutions to several challenges of mobile distributed systems by means of applying design patterns and CORBA middleware technology are also presented.

An introduction to mobile distributed systems is presented in Section 2. An overview of designed patterns that were used during a development is shown in Section 3. The design procedure, the results, and the conclusions are addressed in Sections 4, 5 and 6, respectively.

2. DISTRIBUTED SYSTEMS AND MOBILITY

2.1 Distributed systems

Computer networks are everywhere. The Internet is the biggest network in the world [11]. It is comprised of corporate networks, mobile phone networks, home networks, and even in-car networks. The above-mentioned networks share the same characteristics and can be investigated under one general header – Distributed Systems [2]. “A distributed system is one in which components located at networked computers communicate and coordinate their actions only by passing messages.” [2]

The motivation for developing and using distributed systems is based on the desire to share resources within the systems for common usage. The term ‘resource’ refers to everything from hardware components, such as hard discs, printers, industrial controllers etc., to different types of software objects, such as files, databases, application logic components etc.

Technological achievement in the fields of device size minimizing and wireless networking enables the integration of portable computing devices¹ and embedded systems² into distributed systems.

2.2 Mobile computing

Properties, such as portability and ability to connect to networks in different places, made mobile computing possible. “Mobile computing is the performance of computing tasks while the user is on the move, or visiting places other than their usual environment”. [2] In the case of mobile computing, a user who is away from his “home” environment can still get access to different resources that are too computing or data intensive to reside on the mobile terminal.

Mobile distributed systems are based on wireless networks that are known to suffer from low bandwidth, low reliability, and unexpected disconnections [10]. Mobile terminals in turn, are nodes of a mobile distributed system with, for example, low CPU power, limited battery capacity, varying data input/output capabilities (screen, keyboard etc.) and limited amount of memory.

From an architectural point of view, the mobile distributed systems may have architectures, such as client-server and, ad-hoc dynamic architecture. The topology and structure of the mobile networks is dynamic and unstable. Clients are connecting and disconnecting, moving from one access point to another and organizing spontaneous networks. The routing in such systems is a very complex task, because network addresses of mobile devices are not static like in the fixed networks. Hence, the mobile environment provides several additional problems to the ones originating from the regular distributed systems. Many of which addressed in this paper.

3. DESIGN PATTERNS

“A pattern is a named problem/solution pair that can be applied in new contexts, with advice on how to apply it in novel situations and discussions of its trade-offs” [8]. Practically, a pattern is an idiomatic, stable and proven solution to a certain typical problem. One of the most important features of a pattern is that it can be reused in many different fields and repetitively. The patterns provide formalization and remove ad-hoc from the development process. It is much more desirable to use verified design patterns than to make the design of the system from scratch and “re-invent the wheel”. Several patterns are presented below.

- Information Expert

In complex software systems, a lot of different responsibilities that actually are derived from requirement specification. The responsibilities must be distributed between possibly hundreds of objects. The main idea of the Information Expert pattern is to assign responsibility to such an object that has enough information to fulfill the requirements of the responsibility. In other words, “the

¹ For example laptop computers, handheld devices, personal digital assistants, mobile smart phones, and mobile communicators.

² For example the embedded controllers within white appliances, cars, and home electronics.

objects do things related to the information they have” [8]. The information in the systems usually is spread across many objects. In order to fulfill the responsibility requirements objects need to collaborate. Sometimes the design that is obtained from this pattern is not really desirable because of tight coupling and overloading of the objects with responsibilities. Hence additional criteria, such as Low Coupling and High Cohesion presented below, are required to evaluate the design quality.

- Low coupling

In object-oriented systems, the most important concepts are reusability, low dependency. The design of the system must comply with satisfy these requirements. The Low Coupling pattern offers to assign responsibility in such a way that coupling between objects remains low. “Coupling is a measure of how strongly one element is connected to, has knowledge of, or relies on other elements” [8]. As mentioned earlier, the Low Coupling pattern is an evaluating principle that is used by a designer to make the design decision.

- High cohesion

A serious problem of object design is to keep the complexities of objects manageable. Otherwise, objects become hard to maintain and inconvenient to reuse. In practice, objects can be quite overloaded with responsibilities from different domains. To prevent these problems, the High Cohesion design pattern is proposed. “In terms of object design, cohesion is a measure of how strongly related and focused the responsibilities of an element are” [8]. In order to evaluate the quality of the design, the patterns of High Cohesion and Low Coupling need to be applied throughout the project.

- Indirection

There are many situations in the object design where direct coupling between two objects is to be avoided. The solution is to follow the Indirection pattern [8] that suggests assigning some responsibilities to an intermediate object that will provide decoupling and mediation between the objects. The Indirection pattern also usually raises the reusability of the objects.

- Factory object

Let us assume that we have many objects of the same type in our system, requiring us to solve a number of problems. First, who creates these objects, and second, who is going to manage them in the system? Also, the creating logic can be complex. According to the Factory Object pattern [3], there should be a “factory” that is responsible for creating and managing of all of these objects. The idea of this design principle is also supported by High Cohesion and Low Coupling patterns. Factory Object pattern has a few advantages: a task of creation is assigned to cohesive object; a potential complexity of the creation is hidden inside a factory; there are possibilities to provide centralized management of objects in the memory.

- Protected variations

Every system consists of a set of components or subsystems. These components must have a stable interface that is used to communicate and collaborate with other components. In such systems the stability of interfaces is very important, since even minor changes can cause an undesired influence to other parts of the system. The Protected Variations pattern [8] is one of the patterns provided by Craig Larman. It presents an approach to solving the problem of instability of the component’s interface in the system. The solution consists of two parts: to investigate the components of a system, and identify the components with possibility of instability; then a designer of a system has to create a stable interface around these unstable objects. In this case components have to be wrapped by a new interface.

- Façade

The Façade Design pattern [3] is inherited from Protected Variations pattern. An idea of the pattern is to create a façade for the subsystem. This façade hides all the implementation details of the subsystem. It is responsible for handling all external inputs from other subsystems of user of the system. Practically the façade object does not perform any actions but rather redirects the calls to the appropriate components of the subsystem to execute the operation. This pattern removes undesired direct coupling with other system components.

4. DESIGN OF A CONTEXT-AWARE DISTRIBUTED APPLICATION

A key idea of the application is context awareness [9]. The position of a user in a building and a user identity were considered as context information in the application. The user is able to access different services inside the building. The system adapts its behavior according to changes of the current context. Also, the context has an influence on the service availability. Main functional requirements are presented below:

- A user of the system has to be able to:
 - Obtain his coordinates inside the building;
 - Obtain context map data;
 - Use the services (remote device activation);
 - Authenticate himself in the system;
- The system has to be able to verify the authority of the user to use the services.

Overall structure of the systems is decomposed to a number of components:

- Positioning Service – responsible for providing coordinates of a mobile client inside a building;
- Activating Service – for activating different devices that are close to the client;
- Map Service – for providing context map data to the mobile client according to coordinates;
- Registry Service – responsible for keeping references and other data about other value-added services;
- Clearing System – responsible for control of access and logging of device (services) activation. The system architecture is presented in Figure 1.

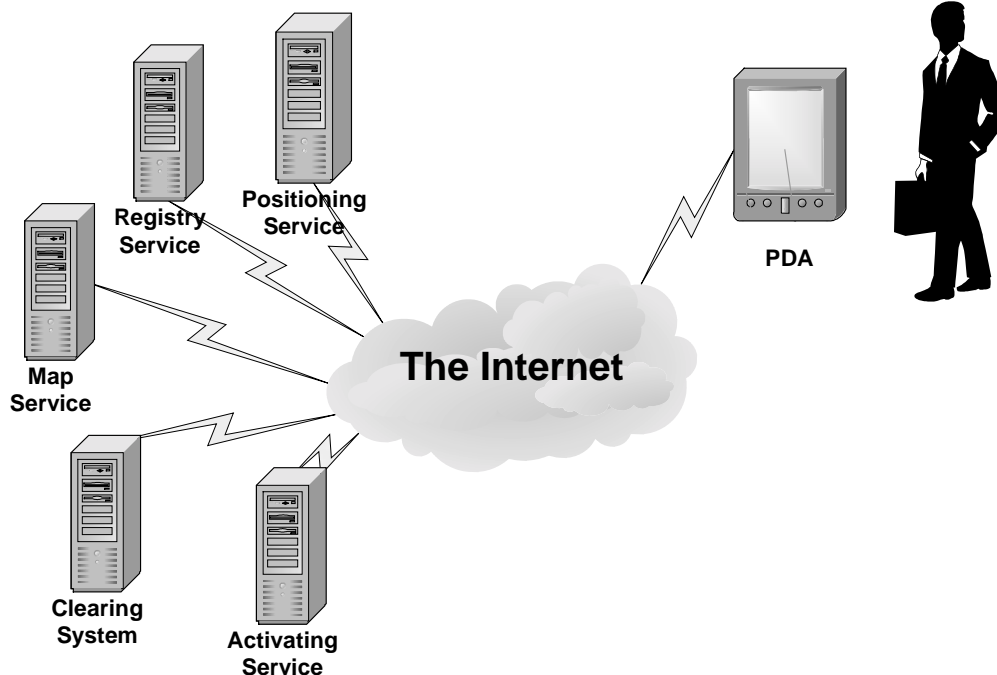


Figure 1. The system architecture of the system

4.1 Design of the Positioning Service

The aim of the Positioning Service is to provide local physical coordinates of the mobile entity within a building. From the architectural point of view, the service is implemented by applying client-server architecture. Service is implemented as a CORBA [1] Server. The Positioning Service is responsible for handling requests from a number of mobile clients. The amount of clients can grow very fast. A special design approach has to be used to provide scalability and maintainability for the system. The Factory Object pattern is applied in the prototype application. The Factory Object design pattern is a paradigm, where one factory object is creating other objects dynamically on demand. When the

factory object is created and registered in the naming directory, it can handle requests for creating and managing other objects of the same type. The CORBA server needs to keep only a few concurrent objects in memory. The client can ask server (factory object) to load specified object on demand or to create a new object. Only the Factory Positioning Object is registered in the Registry Service, but not hundreds of Positioning objects.

A mobile client can suddenly lose the connection to the server. By using this pattern and some features of CORBA technology, we can partially solve the problem of unexpected disconnections. The status of an object that is dedicated for the client has to remain in memory, because the client should maintain the results of the interactions with the server in case of a disconnection. On one hand, the factory object design pattern provides the basis for the realization of a dedicated object and the implementation of the object with CORBA middleware on the other hand provides the statefulness to the object. This approach does not allow performing any actions off-line, but as soon as the cause for the disconnection is solved, the client can reconnect to the server with the same identification name and continue working with the server functionality from where it was interrupted.

4.1.1 Positioning techniques

An important issue of the Positioning Service is how to receive physical coordinates of the mobile client. There are many techniques to obtain coordinates inside and outside a building, such as [12] Outdoor GPS system; Outdoor GSM; Indoor GPS; Infrared based; RFID tags and Smart Cards; Bluetooth [5], Capacitive sensors [6], and Ultra sound sensors. Due to the aspects of availability and maturity of the technologies, we decided to make software emulation of the positioning system before the actual indoor positioning system being developed by one of our partners will be available.

The services software is designed in such a way that it does not depend on a particular positioning technology. The Protected Variations pattern is applied for providing possibilities to plug-in different positioning techniques without affecting the rest of the service parts.

4.2 Design of the Activating Service

The Activating Service is a component of the system that provides the possibility to activate different devices and services inside a building. The system has to make authentication and authorization of a mobile client. The Clearing System is responsible to check the ability of the client to perform any actions with devices.

The service architecture is of a client-server type. The same Factory Object pattern is applied to the Activating Service as to the Positioning Service in order to solve the scalability problem. Additionally, according to the Low Coupling and High Cohesion design patterns the client-dedicated object is designed as a Façade object. It does not execute the interface methods, but rather redirects the requests to the object for device activation and to the Clearing System for obtaining the information about devices and checking the ability of the client to activate it. This approach increases reusability of the software components of the Activating Service.

4.3 Design of the Registry Service

The server may also need to cope with dynamic aspects, especially if the services are provided by mobile entities within a mobile environment. With mobile servers, the access and availability of a service needs to be monitored and controlled in order to avoid contacting non-existing servers.

Application of Leasing design pattern for the Registry Service solves this problem. “The leasing pattern simplifies resource management by specifying how resource users can get access to a resource from a resource provider for a pre-defined period of time” [4]. The idea is that after registration in the Registry Service all services obtain a leasing period of their availability. That period has to be prolonged constantly; otherwise the Registry Service removes a record about the service from its database. As a result, clients would not obtain a reference to non-existing services.

5. RESULTS

The main result of the development of the prototype application is that we have proved the technological feasibility of building mobile distributed systems in wireless environments with mobile terminals by applying middleware technologies.

The use-cases related to the Indoor Guide prototype can be applied for building commercial applications, such as information infrastructures for public places, airports, museums, and supermarkets, for providing the users with a context-related information, such as gateway position, flight information, time reminders, and all kinds of location-based services. The work came out within VIVIAN project is already being exploited by the following IST projects:

- “MyGrocer” that is about building information infrastructure for shopping centers.
- “mEXPRESS” that is about developing of information infrastructure for exhibition places.

From the technical point of view, the aims of the prototype were to demonstrate how to provide solutions for various challenges of the mobile distributed systems; to define what kind of problems can be solved using existing technologies and design methods and what can not. Due to time requirements of the project, we could use only existing standards and solve the problems by means of design methods.

One of the most fundamental problems in distributed systems, including mobile computing systems, is their heterogeneity. The Indoor Guide is a perfect example of such, since it includes

- Different mobile and stationary computers such as NetBook, laptops, and PCs
- The software components were implemented with different programming languages, such as C++ and Java
- Different operating systems: SYMBIAN, MS Windows, Linux. Also, different communication protocols were used

All these heterogeneity problems were solved by means of CORBA middleware technology, which brought an additional indirection layer to the system and hid the details of the operating systems, programming languages, and communication means.

The formal methods of design of software systems were studied; particular attention was paid for the object-oriented design patterns that were followed for making a proper design of the prototype.

Scalability was another problem during the design; applying the Factory Object design pattern solved it. All the servers have to be able to provide services for a number of clients. At the design stage, we decided to make a separate factory object that would create, delete and manage all user-dedicated objects in the system. The factory can create as many client-dedicated objects as needed. Hence, scalability is added to the system.

By using this pattern and some features of CORBA technology, we partially solved the problem of unexpected disconnections. As it was mentioned before, a mobile terminal can suddenly loose the connection to the server. Our task was to maintain the state of the object that is dedicated for the client, since the user should not lose the results of the interactions with the server after the disconnection. The factory object design pattern provides the basis for the realization a dedicated object and the implementation of the object with CORBA middleware provides persistence to the object. This approach does not allow performing any actions off-line, but the client just reconnect to the server with the same identification name and continue working with the server functionality from the status preceding, the disconnection.

The performance and capacity problems related to mobile terminals are mostly cost related. Already nowadays, mobile terminals have enough CPU power, operational and storage memory to run rather complex applications. They can support Java and CORBA technologies, attractive graphical user

interfaces etc. The bandwidth of wireless channels has also increased significantly over the last few years. Hence, the above-mentioned problems of mobile terminals are not fundamental; they will be solved as existing technologies mature and become more affordable.

Some of the problems of mobility, such as routing in ad-hoc networks and handover³; cannot be solved with technologies that are not adopted for the mobile environment. New standards have to be developed and standardized. Some of our partners prepared proposals for OMG consortium for adopting CORBA technology for mobile use.

As a topic for future investigation and development, we are considering downloadable software components and applying mobile agents [4] technology for providing management and coordination of mobile distributed systems. These issues are to be taken into account as building more flexible, adaptable, and usable mobile systems.

6. CONCLUSIONS

Design of mobile distributed applications is discussed from the formalizing of the development process point of view. Solutions to many challenges of mobile distributed systems are presented. The presented results are based on an example of designing of a context-aware system operating in a dynamic environment.

7. ACKNOWLEDGEMENTS

The research and development was done within the EU ITEA project – VIVIAN. The authors would like to acknowledge all our colleagues from Industrial IT Laboratory of Helsinki University of Technology and other partners of VIVIAN project for their contribution.

8. REFERENCES

1. CORBA Architecture Specification,
<<http://www.omg.org/technology/documents/formal/corbaaiop.htm>>
2. Coulouris, G., Dollimore, J., Kindberg, T., Distributed Systems Concepts and Design, 3rd edition, Addison-Wesley, (2001), 772p.
3. Gamma, E., Helm, R., Johnson, R., Vlissides J., Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley, (1995), 395p.
4. Jain, P., Kircher, M., Leasing Design Pattern
<<http://jerry.cs.uiuc.edu/~plop/plop2k/proceedings/Jain-Kircher/Jain-Kircher.pdf>>
5. Krassi, B., Reliability of Bluetooth, 12th Conf. On Extreme Robotics, RTC, St. Petersburg, (2001)
6. Krassi, B., The Large Range Capacitive Sensor for Human Motion Recognition, BOAC (2000)
7. Lange, D., Oshima, M., Programming and Deploying Java Mobile Agents with Aglets, Addison-Wesley, (1998), 225p.
8. Larman, C., Applying UML and Patterns. An Introduction to Object-Oriented Analysis and Design and the Unified Process, 2nd edition, Prentice Hall PTR, (2002), 627p.
9. Mari Korkea-aho, Context-Aware Application Survey, Helsinki University of Technology, 2000,
<<http://www.hut.fi/~mkorkeaa/doc/context-aware.html>>
10. OMG, Wireless Access and Terminal Mobility in CORBA Specification, OMG Document dtc/01-06-02, Object Management Group, (June 2001).
11. Tanenbaum, A., Computer Networks, 3rd edition, Prentice Hall PTR, (1996), 814p.
12. VTT Information Technology, Mobile Location Workshop, Espoo, (2001).

³ Switching between different data communication channels that is transparent for a user.