A Middleware for Supporting Disconnections and Multi-Network Access in Mobile Environments

Mejdi Kaddour, Laurent Pautet
École Nationale Supérieure des Télécommunications
CS & Network Department
46, rue Barrault
F-75634 Paris CEDEX 13, France
{kaddour, pautet}@enst.fr

Abstract

The middleware infrastructure to support applications is becoming critically important to the new horizons of mobile and pervasive computing. In this context, the goal of our work is to propose architecture and a prototype of a middleware which is adapted to the usage and the requirements of the future mobile systems. While message oriented middleware (MOM) is an established concept for a long time on the fixed environments, we work to put forward the idea that it proposes an effective solution of many issues in the mobile environments. Our contribution is to bring current MOMs closer to the specifities of these environments, through providing more adaptability and context-awareness, and also through supporting more specific problems like intermittent connection and multi-network access.

1. Introduction

Over the last few years, mobile technologies have drawn the attention of a growing large public. Proliferation on the market of increasingly powerful and autonomous mobile devices and the considerable success of the local area networks (802.11b, bluetooth) will undoubtedly allow the emergence of complex applications and enable a mobile access to systems which are now available only on a fixed infrastructure. As it is unlikely that a single wireless network technology will impose itself in all kind of usage, a challenge will be to enable applications to communicate in a transparent way through various access networks.

In order to deal with this increasing complexity, a lot of research is performed on the design of new middleware platforms dedicated to mobile environments. The conventional platforms which are generally based on a request/reply paradigm revealed their limits in this field [6]. The extremely variable and dynamic nature of wireless networks, and the mobile device constraints with respect to processing power and autonomy, highlighted the need for a more adapted communication mode. In this context, we think that *Message Oriented Middleware* (MOM) [1] may effectively address most issues raised by these environments.

Indeed, MOMs inherently allow separate, loosely coupled components to reliably communicate. They can exchange messages without having an explicit reference on each other and without being necessarily connected at the same time. This information dissemination model is appropriate for an environment characterized by the mobility of entities (change in the location and the physical address), by the interaction of a priori anonymous and variable entities, and also by intermittent network connections. Moreover, MOMs provide a certain level of network fault-tolerance using persistent storage.

However, existing MOM products don't support some specific problems of the mobile environments because they are originally designed for fixed environments. Among these problems: frequent disconnections, bandwidth variability, scare resources on mobile devices, etc. For example, a connected application can always send messages whatever the connection state of their recipient, but this application will be often constrained to repeat the same tasks or even to face irrecoverable errors after a short disconnection.

Our approach introduces a specific support for mobility in a MOM. Our contribution is to allow the MOMs, and specifically those based on Java Message Service (JMS) [4], to have a better ability to react and to adapt their behavior in the variable conditions of the mobile environment. With this intent, we introduced an adaptable communication support based on *dynamic negotiation* and *environment monitoring*. This communication support is also designed to deal with the problems of intermittent connection and multi-network

access.

This paper is organized in the following way. Section 2 introduces briefly the Java Message Service (JMS) specification and sets out our approach. Section 3 describes the following features of our MOM: adaptable communication, value-added services and in particular the store-and-forward service, dynamic negotiation and transparent reconnection. In section 4, we present some experimental results. Section 5 concludes this paper outlining our future works.

2. A middleware for mobility

2.1. Java message service (JMS)

JMS, part of the J2EE (Java 2 Enterprise Edition) suite, is a set of APIs allowing the development of messaging applications in java. These APIs represent the common functionalities of actual messaging systems.

A JMS system is organized around *clients* and a *provider*. The provider receives messages from "producer" clients and delivers them to "consumer" clients. The provider is the node of all messages in transit which are waiting to be recovered by their respective recipients. JMS enables applications to communicate asynchronously via two messaging models:

1. Point-to-point (PTP): each message is addressed to a specific queue on provider. The receivers retrieve their messages on the queues established to maintain them, and each message has only one receiver.
2. Publish/Subscribe: in this model, the producer addresses a message on a particular topic on the provider. This message can have several consumers which are anonymous for its producer.

Two concepts are fundamental in JMS: connection and session. A connection encapsulates a virtual connection with a JMS provider. It could represent an open TCP/IP socket between a client and a provider. Connections are used to create one or more sessions. A session is a single-threaded context for producing and consuming messages. Sessions are used to create message producers, message consumers, and messages.

2.2. Our approach

The basis of our approach is to enrich an existing JMS implementation designed for fixed environments (OpenJMS [8]), while providing a specific support for mobility. This solution preserves the essential JMS functionalities. The main purpose is to introduce more flexibility and adaptability in order to better deal with the variability of the environment (e.g., intermittent connection, bandwidth variability).

The adaptability is introduced by the use of a flexible communication layer and through the permanent monitoring of environment conditions. This communication layer is formed by a protocol graph which uses the interfaces of the communication framework defined by the Jonathan platform [3]. Jonathan is a minimal framework being used to develop object oriented middleware. It provides a whole set of basic interfaces and components which carry out tasks like communication between objects, buffer control and data marshalling. We extended these interfaces in order to add the possibility to restructure the protocol graph dynamically. Protocol layers can be thus added or removed during runtime. This behavior enables the platform to adapt itself to the current environment conditions. We don't describe in this paper how and when this adaptation mechanism is triggered, this issue was described in a previous paper [5].

In a more specific way, we propose a solution to the problems of frequent disconnections and multi-network access. A JMS application can continue its execution even in the case of a disconnection or if the device changes its access network (e.g., 802.11b, Bluetooth, GRPS). These events are handled by the middleware and hidden to JMS applications. The anonymity property of JMS doesn't prevent the association of a JMS connection with several physical connections. So, the client access to a topic or to a queue on the provider, through a JMS connection, doesn't depend on its network address. Besides, the asynchronism property of the communication guarantees that the messages intended for a client will not be lost during the period of its disconnection. For example, a client can send its request to a server through the provider by using a GPRS connection, disconnects itself, and then retrieves the server's reply on the provider by using an 802.11b connection. This issue is addressed by the transparent reconnection mechanism (see section 3.4).

3. Communication patterns

3.1. Value-added services

These services are not essential to the logical behavior of the middleware, but aim at improving its performance and usability under certain conditions. They are used in general to reduce the resources necessary to the transmission of messages or to ensure a certain quality of service. They are implemented as protocol layers and can thus be added and removed dynamically from the protocol graph. We describe hereafter the implemented services.

Compression service: it is used to preserve the bandwidth by reducing the size of the messages sent and receipts between the JMS clients and the provider. Compression also permits to reduce the memory occupied if the messages are

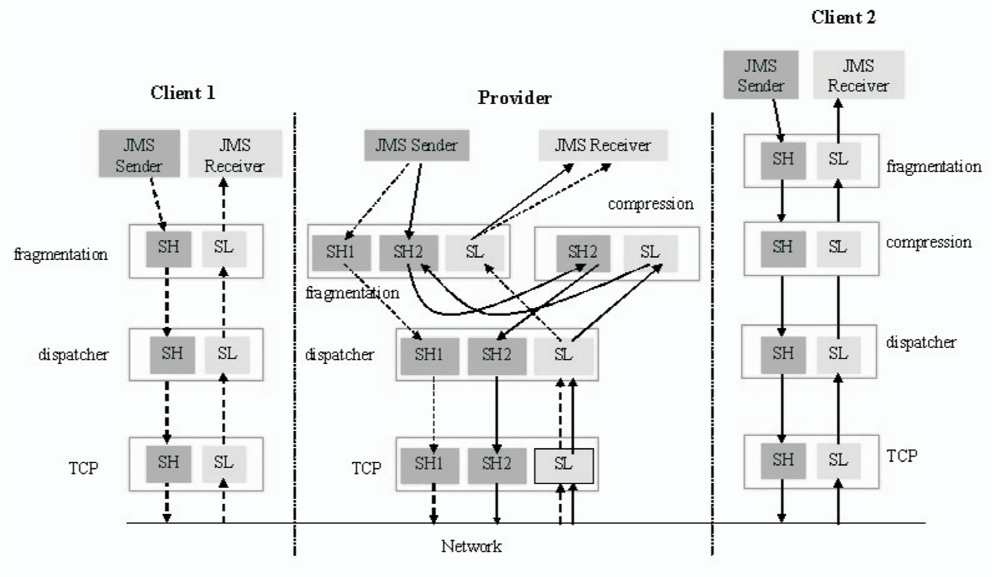

Figure 1. Communication between 2 Clients and a Provider

cached in the store-and-forward service. The compression algorithms implemented are zip and gzip [2].

Fragmentation service: it consists in sending large size messages in several packets. This mechanism enables, in the case of failure during message transmission, to re-sent only missing packets. On the receiver side, it carries out the reconstitution of the original messages after the reception of all the fragments, it then delivers them to the higher layers. This service also supports the reception of duplicated and out-of-order packets.

Store-and-Forward service: it enables applications to continue their processing even in the case of disconnection or weak connection with the provider. Therefore, it can prove to be very useful in the mobile environment where the disconnections are frequent. Messages are temporarily stored in a local cache while waiting to be transmitted when the network connection is available again. During disconnection periods, the messages stored in the S&F cache are reordered, respectively, according to following criteria: JMS-defined priority, blocking (must be transmitted before the application can proceed), persistency, size and time of reception. After reconnection, messages which have these characteristics are sent in first (see [5] for more details).

3.2. communication between several clients and a provider

On the JMS client side, we use the Jonathan communication framework without any significant modification. Each protocol layer consists of two objects which are the implementations of the `Session_High` (SH) and `Session_Low` (SL) java interfaces of Jonathan. SH object carry out a specific processing on the messages and transmits them towards the SH object of the lower layer. The SL receives messages from the lower layer, processes them, and then transmits them to the SL object of the higher layer. The messages sent and received from JMS layers always follow the same route through the protocol graph.

The task is slightly different on the provider side, since Jonathan doesn't manage different client connections simultaneously. We extended the framework to enable the provider to manage these connections through a single protocol graph. This latter is composed of all the protocol layers used by the actually connected clients. Each protocol layer contains only one SL object and several SH objects corresponding to each client. The SL object maintains a table which associates, for each client using this layer, an SH object in the lower layer and an SL object in the higher layer. When an SL object sends a received message from a client to the higher layer, it also sends the identity of the SH object which this layer must contact to send the reply to this client. This explains why each client must have its own SH object because it is a way to be identified through the protocol graph. Thus, SL object is used to redirect the client's messages to the higher layer according to the route negotiated before (Section 3.3).

The figure 1 represents an example of communication. Client 1 uses a graph composed of a fragmentation layer, a dispatcher layer and a TCP connector (network transport protocol); Client 2 uses the same layers, except that it adds a compression layer below the fragmentation. The dispatcher layer which, doesn't make any specific processing on the

message core, has special functions. First, it multiplexes several connector layers (TCP, UDP) with the lower part of the other layers, which enables the use of several connectors at the same time. Second, on the provider side, it identifies the first higher layer which must receive the message. The fragmentation layer, on the provider, instantiates two SH objects corresponding to each client, while the compression layer instantiates only a SH object for the client 2. The messages received from client 2 must be decompressed before being defragmented.

3.3. The dynamic negotiation mechanism

The client and the provider must undertake a negotiation phase before the beginning of the communication. This phase is necessary to both of them in order to determine the protocol graph which should be used, and thus to be able to interpret the messages correctly.

At the initialization of the JMS connection with the provider, the client instantiates an initial protocol graph (e.g., made up from of the following layers: store-and-forward, compression and TCP connector). Then, it sends through the `createConnection` message, a graph identifier which is a symbolic representation of this graph. At the time of its reception, the provider after having decoded this identifier, decides whatever it can instantiate or not a similar graph. In the first case, it instantiates also this graph and replies by a positive acknowledgment. So, the client will be able to initialize the communication there by using this configuration. In the second case, the provider replies by a negative acknowledgment containing the identifiers of the rejected protocol components. At this time, the client must initiate a new negotiation phase.

This negotiation phase can also be undertaken after connection initialization because the environment conditions can vary in a considerable way. Thus, the protocol graph is likely to become unsuited to a given moment. For example, a decrease of the available bandwidth for a JMS client may require the addition of a compression layer to the graph. The provider also can require the removal of the compression layer from the graph of some clients if it is heavily loaded.

This negotiation mechanism remains valid when it's the provider initiates the procedure. The figure shows only the case when the provider accepts the client proposal. The protocol graph remains unchanged on both if the proposal was rejected.

3.4. The transparent reconnection of a client

On conventional JMS systems, the loss of a client's network connection with the provider systematically implies the closing of the JMS connection on the two sides and the loss of the messages not yet sent by this client. This behavior is, in our opinion, inadequate in the mobile environment where the disconnections are rather frequent. For this reason, we have designed a mechanism which hides the loss of network connection to the higher layers, and allows them to proceed their execution normally after connection is again available. The change of the physical address of the client after reconnection doesn't matter - the provider identifies the client globally and uniquely using a JMS-defined identifier. This mechanism takes place in two phases:

Phase 1: by going up from the lower layers towards the higher layers of the graph, the disconnection event is intercepted by the S&F of the client. After that, the non-blocking messages sent by a JMS session and received by the S&F are stored in its local cache. A blocking message received from a session implies its interruption. All the protocol layers remain instantiated, except the connector layer which is closed. During this time, the provider interrupts on its side the JMS connections and sessions associated with this client. A connection timeout is associated with these objects. If the client doesn't reconnect before the expiry of this timeout, these objects are definitively closed.

Phase 2: after its reconnection, the client sends a message including its JMS `client ID`. This message is sent directly on the connector layer. When it receives this message, the provider checks whether JMS objects associated previously with this client still valid. If so, the previous client's route on the provider's graph is associated with this new connector layer. The provider sends after that a reply message to the client. If this reply is positive, the old graph of the client is re-associated in its turn with this new connector layer. The messages cached in the S&F are then sent, and the sessions stopped before are gradually resumed. The client sends a confirmation message to inform the provider that is can also resume the stopped JMS connections and sessions.

4. Experimental results

We evaluated the performance improvement and the cost generated by the use of some value-added services on the client side. This experience consists of a JMS client which sends consecutively huge messages ($\simeq 50ko$) to the JMS provider. We took the average time of the provider replies to these messages (time elapsed between sending of a message and reception of an acknowledgment from the provider). JMS client runs on an Ipaq H3800 PDA with a linux OS and a J2SE 1.3.1 Java Virtual Machine (JVM), the JMS provider runs on a fixed machine with a 2.4 GHz Pentium IV processor. The Ipaq is connected to the provider on an 802.11b wireless LAN. In order to be able to test the client under different conditions, the connection is made through the NistNet network emulator [7]. This emulator,

installed on a third machine, enables to control different network parameters as the bandwidth or the packet loss rate. We set this rate at 20% which corresponds to an average value in an 802.11b network [9]. The figure 2 shows the results obtained by using the following protocol graphs: TCP, TCP+compression (gzip with the highest compression level), TCP+fragmentation+compression (the size of each fragment is 1500 bytes) .

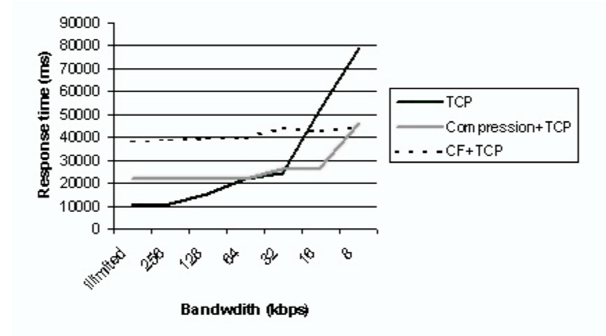

Figure 2. Performance Results of Value-Added Services

The use of compression adds an anticipated execution overhead when the bandwidth is more than 64kbps. This result illustrates the fact that the bandwidth on this level is sufficient to send the uncompressed messages and also that the compression adds a significant processing on an Ipaq. Nevertheless, this result is reversed when the bandwidth decreases under a certain level (20kbps). Under these conditions, the benefit obtained by the reduction in message size compensates largely the necessary additional processing, and the reply time obtained with compression is lower.

The figure shows also the overhead generated by the use of fragmentation. This service slows down the sending of messages but the reply times remain rather stable. However when bandwidth is low as in the case of 8kpbs, which corresponds approximately to a GSM network, fragmentation gives a little advantage compared to compression alone. This advantage is principally provided by the partial retransmission of a message in the case of packet loss. We obtain better performance when the fragment size corresponds to the maximum transmission unit of the network (MTU). Fragmentation is thus quite useful in a noisy environment.

5. Conclusion

We discussed in this paper some aspects of our platform. We argued that MOM is an interesting alternative to others middleware systems in a mobile environment. We believe that many low-level issues of the mobile environment can be solved by the adoption of a high-level view without regarding to network and material platform diversity. The decoupling between message-passing semantics and low-level transport and network support, isolates the applications from the unpredictable events which can occur in the network. We used this feature to design a mechanism which enables applications to continue their processing even in the case of disconnection/reconnection or access network change. To manage the environment variability, we introduced an adaptable communication support. The concept of value-added service which is at the base of this support can be generalized to carry out other more specific processing on the messages. A semantic transcoding can be applied when necessary to convert huge data (multimedia) into less bandwidth-consuming data (text).

The execution environment should not be completely hidden to applications. When several access networks are available, applications should also select the suitable one according to some given parameters (e.g., available bandwidth, access cost). We take a first step toward this objective by introducing an environment monitoring module. We will generalize this adaptability and context-awareness concept on the application level. An application can define its own policy of execution or quality of service. This policy, which is a kind of contract between the application and the middleware, has two functions. First, it specifies application requirements to the middleware; second, it enables application-awareness of the status of some environment parameters. Finally, we plan to use more advanced applications to serve as benchmarks.

References

[1] G. Banavar, T. Chandra, R. Strom, and D. Sturman. A case for message oriented middleware. In *ISDC*, pages 1–18, 1999.

[2] P. Deutsch. Gzip fi le format specifi cation version, 1996.

[3] B. Dumant, F. Horn, F. D. Tran, and J.-B. Stefani. Jonathan: an open distributed processing environment in Java. In *Proceedings of the IFIP International Conference on Distributed Systems Platforms and Open Distributed Processing*, 1998.

[4] M. Hapner and al. *Java Message Service specifi cation 1.1*. Sun Microsystems, April 2002.

[5] M. Kaddour and L. Pautet. Towards an adaptable message oriented middleware for mobile environments. In *Proceedings of ASWN'O3*, Bern, Switzeland, July 2003.

[6] C. Mascolo, L. Capra, and W. Emmerich. Middleware for mobile computing. *LNCS 2497. Springer Verlag*, 2002.

[7] National Institute of Standards and Technology. NIST Net. `http://snad.ncsl.nist.gov/itg/nistnet`.

[8] Openjms.org. OpenJMS Project. `http://www.openjms.org`.

[9] C. Steger, P. Radosavljevic, and P. Frantz. Performance of IEEE 802.11b Wireless LAN in an Emulated Mobile Channel. In *IEEE Vehicular Technology Conference (VTC)*, Jeju, Korea, April 2003.