A Platform for Experimenting Disconnected Objects on Mobile Hand-Held Devices

Denis Conan — Sophie Chabridon — Olivier Villin — Guy Bernard

Institut National des Télécommunications
9 rue Charles Fourier
91011 Évry cedex
France

{Denis.Conan,Sophie.Chabridon,Olivier.Villin,Guy.Bernard}@int-evry.fr

ABSTRACT. *Mobile databases and distributed systems relying on wireless communication networks have to deal with variable levels of connectivity. This demonstration proposes a generic Disconnected Object Management (DOM) service, called* DOMINT*, that enables work continuity even when weakly connected or disconnected. For better agility and fidelity, we offer both application-aware and application-transparent adaptations.*

RÉSUMÉ. *Les bases de données mobiles et les systèmes répartis utilisant les réseaux sans fil doivent faire face à de fortes variations du niveau de connectivité. Cette démonstration propose un service générique de gestion d'objets déconnectés appelé* DOMINT *qui permet de continuer à travailler en modes faiblement connecté ou même déconnecté. Pour améliorer l'agilité et la fidélité, nous offrons une adaptation transparente pour l'application ainsi qu'une adaptation avec collaboration entre l'application et le système.*

KEYWORDS: *Mobility, disconnection, CORBA, wireless networks.*

MOTS-CLÉS : *Mobilité, déconnexion, CORBA, réseaux sans fil.*

1. Introduction

An important characteristic of mobile environments is that they suffer from frequent disconnections. A disconnection is a normal event in such environments and should not be considered as a failure. This has a profound impact on how transaction management is implemented and how data consistency is guaranteed in such environments [BAR 99]. We distinguish between two kinds of disconnections: voluntary disconnections when the user decides to work on their own for saving battery or communication costs or when radio transmissions are prohibited as aboard a plane, and involuntary disconnections due to physical wireless communication breakdowns such as in an uncovered area or when the user has moved out of the reach of a base station. We also handle the case where the communication is still possible but not at an optimal level. It corresponds to what has been called weak connectivity [MUM 95]; it results from intermittent communication, low-bandwidth, high-latency or expensive networks.

The weak connectivity of mobile environments in conjunction with the relative resource poverty of hand-held devices leads to a trade-off between autonomous applications and interdependent distributed applications. This trade-off is well explained in [NOB 97] where the range of strategies for adaptation brings out three design alternatives: no system support (*laissez-faire* strategy), collaboration between the applications and the system (*application-aware* strategy), and no changes to the applications (*application-transparent* strategy). Previous works [JOS 97, MUM 95, NOB 97, PET 97] have demonstrated the possibility that a system can provide good performance even when the network bandwidth varies over several orders of magnitude, but also the need for application intervention to improve agility (speed and accuracy) in reaction to changes in resource availability and to specify fidelity in terms of data consistency.

The contribution of this demonstration is to propose a Disconnected Object Management (DOM) service for application-aware adaptation in addition to application-transparent adaptation. It is a service that enables work continuity in a transparent manner even when weakly connected or disconnected. This work should be seen as a first step towards a complete mobile information management system; it defines the main basic components of a generic infrastructure to experiment deployment, replication and various reconciliation strategies.

The remainder of this presentation details the architecture (Section 2), the example application (Section 3) and the demonstration (Section 4).

2. Architecture

In a classical distributed application with strong connectivity, the graphical user interface is loaded on the mobile terminal and the server objects are hosted on machines of the wired network. Service continuity while disconnected implies transferring some elements of the servers to the mobile terminal before loosing connectivity, logging operations or state changes during the disconnection, and re-integrating when re-connecting. In order to support multiple applications concurrently, some parts of resource management and log management are centralised and application-transparent. For application-awareness, these services are realised by objects that accept requests from applications. The application-aware resource management service abstracts to applications connectivity information provided by the operating system and applications can specify which resources and resource levels correspond to bad, weak, or strong connectivity, thus improving agility.

Figure 1 presents the architecture of the DOM service. More precisely, it depicts UML-like collaboration diagrams of the client sending a request to a remote object when the connectivity is strong (case 2.a) and then sending a request in the case of weak connectivity (case 2.b).

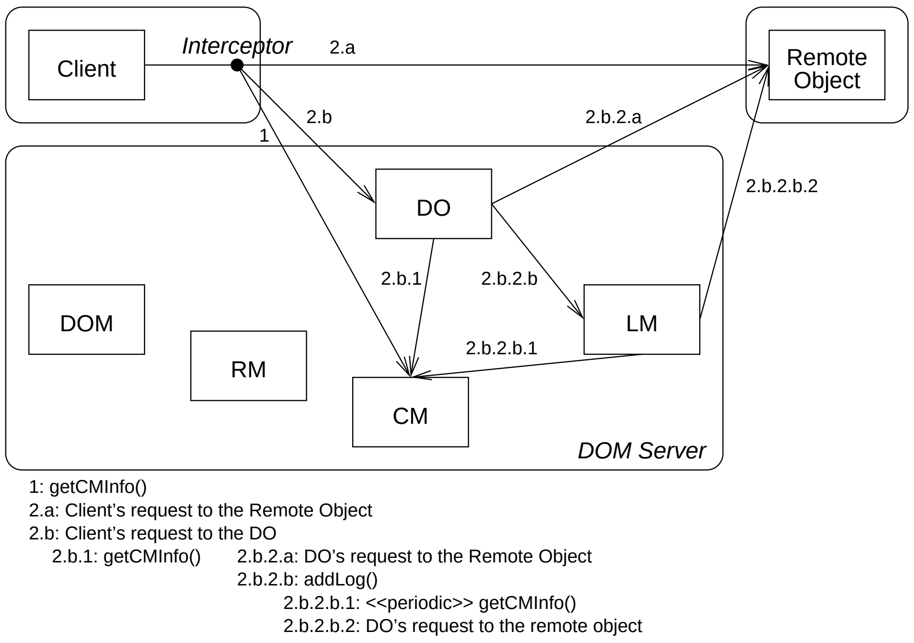
Client
Interceptor
2.a
Remote Object
2.b
2.b.2.a
1
DO
2.b.2.b.2
2.b.1
2.b.2.b
DOM
LM
RM
2.b.2.b.1
CM
DOM Server
1: getCMInfo()
2.a: Client’s request to the Remote Object
2.b: Client’s request to the DO
2.b.1: getCMInfo()
2.b.2.a: DO’s request to the Remote Object
2.b.2.b: addLog()
2.b.2.b.1: <<periodic>> getCMInfo()
2.b.2.b.2: DO’s request to the remote object

Figure 1. *The DOM service architecture.*

All the rectangles in Figure 1 represent objects. All the requests from and the responses to the client are intercepted. On request sending, the interceptor acts as a switch between the disconnected object DO and the remote object. On response reception, the interceptor detects possible communication failures between the sending of the request and the reception of the response. A disconnected object is an object which is similar in design and implementation to the remote object, but specifically built for supporting disconnection and weak connectivity. It is the application designer’s responsibility to balance between an easy design and a more complex one that adapts better to connectivity variations.

Disconnected objects are associated via application-transparent portable interceptors to the client. If the client wants application-aware adaptation, it obtains the reference of the disconnected object manager DOM, for example, from a file stored on the mobile terminal. The DOM is the entry point of the DOM service to find the other managers. The resource manager RM is a factory of connectivity managers CMs. A CM realises the abstraction of connectivity information related to one resource. The policy currently implemented associates a CM per logical link between a client and a remote object; it is the finest granularity at the middleware level.

The interceptor obtains the connectivity information from the CM (1), and then, decides where the client's request must be issued. When the connectivity is strong —*i.e.* in the connected mode—, the client's request leaves the mobile terminal to reach the remote object (2.a). As a result, the DO cannot keep up to date with the latest requests. Therefore, the DO should periodically call the remote object for an incremental state transfer [KHU 02].

When the connectivity becomes weak or null, forcing the client to enter into the partially connected or disconnected modes respectively, the client's request is issued to the DO (2.b). The requests that follow in the scenario are application-dependent because the DO is built by the application's designer. The DO updates its state and prepares a new request, called a DO request, for the remote object. The simplest case is that the DO request is equivalent in parameters' content and operation name to the client's request. Next, the DO asks its CM for connectivity information (2.b.1). We give two possible ends to this scenario: 2.b.2.a and 2.b.2.b.

In the partially connected mode (2.b.2.a), the operations are executed locally and remotely; the DO sends the DO request to the remote object and updates its state again if necessary. In the disconnected mode (2.b.2.b), the operations are executed only locally. If appropriate, the DO encodes the DO request in a data container and sends it to the log manager LM. Periodically, the LM decodes the logged request with a method provided previously by the DO, tests the connectivity (2.b.2.b.1) and, if possible —*i.e.* partially connected mode—, forwards the DO request to the remote object (2.b.2.b.2). Clearly, the execution when disconnected is not equivalent to an execution while connected or partially connected. This is acceptable provided that the connectivity information is visualised by an iconic image in the client's user interface. This is very close to the relaxed check-out mode proposed in [HOL 00] where the users know that they are viewing stale data but it is still useful for them to do so.

In addition, users can disconnect or re-connect voluntarily by calling `disconnect()` or `reconnect()` operations; these calls are addressed to the remote objects, intercepted, and (re-)directed to the DO. In case of voluntary disconnection, the DO is responsible for initiating the loading of the state from the remote object. When reconnecting voluntarily, the DO asks the LM to flush the logged data pessimistically, that is the re-connection is successful only if the log is empty when the call returns.

For the partially connected and disconnected modes, we log and propagate operations instead of state contents/changes like in [PET 97]. In fact, this is application-dependent since the log manager does not interpret logged requests. The code that can parse and forward the logged requests is provided at initialization time by disconnected objects (via object by values (OBV) in the CORBA environment) that we name DO request interpreters. Disconnected objects and DO request interpreters can log and propagate either operations or state changes. The log manager receives as an `in` parameter a DO request interpreter, that is a description of the state and the code of the object that is responsible for the interpretation of future logged requests, and a new instance is automatically created in the execution entity of the log manager. Provided that all the DO request interpreters inherit the same abstract interface, the log manager is generic and application-independent.

For more details on the DOM service, more especially on interfaces for application-aware adaptation, the reader can refer to [Viv02].

3. Example application: A wireless email browser

This section illustrates the adaptation of an email browser to wireless environments. Our email browser offers the basic functionalities of well-known software such as Netscape Messenger or Microsoft IE. The user handles messages composed of a body and a header, itself divided into an identifier, the names of the sender and the receiver, a subject, the date of sending, and a status (read or unread). The main functionalities provided by the graphical user interface (GUI) are sending, replying to, forwarding, receiving and deleting a message.

In the first version of the email browser named "centralised", the GUI is executed into the same execution entity as the user mailbox object. The mailbox object plays two roles: *(1)* the mailbox object stores received emails ; *(2)* for sending a message, the GUI sends the message to the mailbox object, the latter gets the receiver's address from the mailbox manager object and forwards the message. A second execution entity contains the mailbox manager that is responsible for creating, deleting and localising the mailbox objects.

The second version of the email browser named "distributed" is obtained by separating the GUI and the user mailbox object into different execution entities. The GUI is launched by the user in the mobile terminal and communicates with the corresponding mailbox object via wireless links. Since some of the data that are copies of the mailbox object's data are locally stored within the GUI, the distribution leads to the separation of GUI's operations into two groups: the operations that only impact local (GUI's) data and the ones that are carried over the mailbox object straight after being executed by the GUI. A typical operation of the first group is the operation `changeStatus()` of the GUI saying that a message has been read or unread. In the centralised version, the operation is synchronously performed on the mailbox object. Now, it is applied and logged by the GUI in order to avoid generating too many requests on the wireless network. At the next remote operation execution, for instance a call to `receiveMessage()`, the log of local operations is transmitted as an argument and applied to the mailbox object before the processing of the remote operation. So, the effects of `changeStatus()` are seen before the effects of `receiveMessage()`, as in the centralised version. Therefore, the GUI logs all its local operations since the last remote operation that do not need to be processed remotely in a synchronous way. Another consequence is that all the remote operations have as their first argument an array containing the list of local operations. This design pattern is rather simple and can be applied apply to distributed applications that are piece-wise deterministic.

However the quality of the wireless link, in order to load data from the mailbox object at a convenient rate and quantity, messages are read in two steps: first the header, next the content. The user browses the set of headers and loads only the desired contents. The reasons for this distinction are that contents are usually much larger than headers, and being optimistic, users will not read every content whereas they read all the headers. Another way to adapt to the wireless link is the addition of "collective" operations that read and delete groups of messages: *e.g.* all the read/unread messages, all the messages.

In the connected mode, the client's requests are directly sent to the remote object. Hence, the state of the disconnected object on the mobile terminal does not evolve. The advantage of this mode is that there is no indirection and the state of the disconnected object can be empty, thus saving memory. When the mobile terminal becomes partially connected, the interceptor calls the `disconnect()` operation on the disconnected object which in turn calls the `disconnect()` operation on the remote object to transfer the state.

In the partially connected mode, the operations are executed locally and remotely. If the prototype of the operation contains only `in` parameters, the operation is executed locally first and then remotely so that the disconnected object remains up to date. If the prototype contains only `out` parameters and a return type, the operation is executed remotely first and then locally. The consequence is that the disconnected object remains up to date with the data loaded from the remote object before it responds to the client. The mixing of `in`, `inout`, and `out` parameters and a return value is let as an open issue in our first study. Another open issue is the support of exceptions thrown by the servers and sent as responses to the clients.

In the disconnected mode, the operations are executed only locally. If the prototype of the operation contains only `in` parameters, the operation is logged. If the prototype contains only `out` parameters and a return type, whether or not the operation is logged depends on whether the state of the target object changes. The mixing of `in`, `inout`, and `out` parameters and a return value and the throwing of exceptions raises the same difficulties as mentioned previously. In addition, recall that every operation has as its first argument an array representing a log of operations that were local to the GUI. This log is also added to the log of the local copy. Of course, this first argument is an `in` parameter but does not take part in the previous discussions. Finally, an important hypothesis of this study is that the remote object cannot be accessed concurrently by other clients while the current client is disconnected. Thus, the reconciliation is eased and kept simple. The transition between the disconnected mode and the partially connected mode corresponds to the replay of the operations logged by the local copy.

4. Demonstration

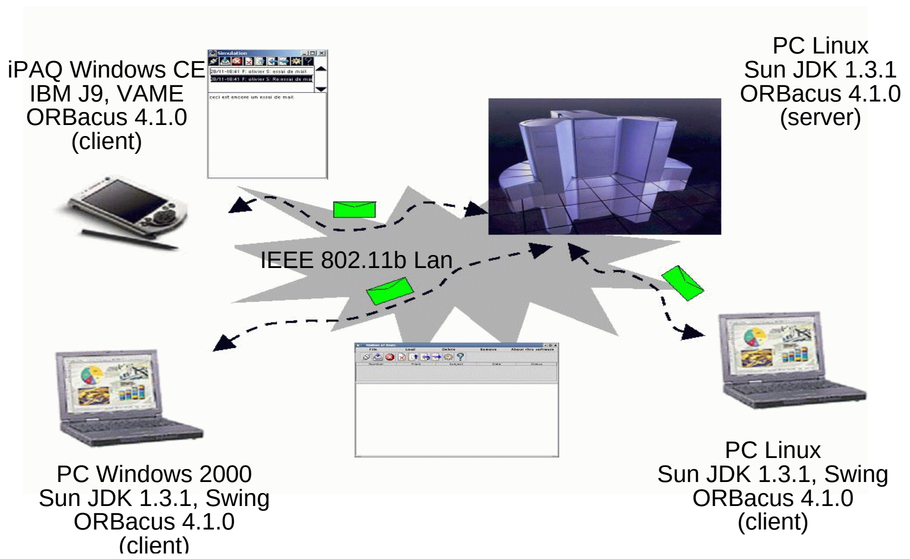

Figure 2. *The demonstration with the wireless email browser.*

For implementing and validating the DOM service, CORBA [OMG 01] has been chosen for its ability to be used in multiple domains and for providing extensibility mechanisms such as portable interceptors to build application-transparent services, and objects by value to build application-aware services. Disconnected objects, being CORBA objects, are accessible from everywhere, so from all the applications of the mobile terminal. Another rationale for letting the disconnected object being a CORBA object is that it can use standard CORBA services such as naming, event notifications or transactions independently of the framework. Details of the design and implementation can be found in [CON 02].

We have conducted series of performance measures in different software and hardware combinations (laptop PC and iPAQ PDA, running Windows or Linux). For wireless communications, a Compaq IEEE 802.11b WL110 card at 11Mbps was plugged in all devices and we used a software base station. The demonstration shows the wireless email browser example application involving three clients with three different hardware and software configurations and one server. Figure 2 draws an overview of the demonstration.

5. Conclusion

This demonstration proposes a generic service for Disconnected Object Management that can benefit to distributed applications running in mobile environments. It decomposes into a disconnected object management interface, a resource manager, a connectivity manager and a log manager. In the example application we have presented, disconnected objects are proxies of server objects, but could be proxies of databases. Similarly, the log manager logs operations on server objects, but could log data base operations. The prototype we have realised demonstrates that some today hand-held devices and a fortiori future mobile devices can embed our framework, that includes a complete ORB. We have presented a prototype of an email browser example application. Tests were run on both a laptop PC and an iPAQ PDA. The performance results show that the DOM service overhead is negligible for the end user.

6. References

[BAR 99] BARBARÁ D., "Mobile Computing and Database — A Survey", *IEEE Transactions on Knowledge and datad Engineering*, , num. 1, 1999, p. 108–117.

[CON 02] CONAN D., CHABRIDON S., BERNARD G., "Disconnected Operations in Mobile Environments", *Proc. 2nd IPDPS Work. on Par. and Dist. Comp. Issues in Wireless Networks and Mobile Computing*, Ft. Lauderdale, Florida, April 2002.

[HOL 00] HOLLIDAY J., AGRAWAL D., EL ABBADI A., "Planned Disconnections for Mobile Databases", *3rd DEXA Int. Work. on Mobility in Databases and Distributed Systems*, Greenwich, U.K., Sep. 2000.

[JOS 97] JOSEPH A., TAUBER J., KAASHOEK F., "Mobile Computing with the Rover Toolkit", *IEEE Trans. on Comp.*, vol. 46, num. 3, 1997.

[KHU 02] KHUSHRAJ A., HELAL A., ZHANG J., "Incremental Hoarding and Reintegration in Mobile Environments", *Proc. of the Int. Symp. on Appli. and the Internet*, Nara, Japan, Jan. 2002.

[MUM 95] MUMMERT L., EBLING M., SATYANARAYANAN M., "Exploiting Weak Connectivity ofr Mobile File Access", *Proc. of the 15th ACM Symp. on Oper. Syst. Princ.*, Copper Mountain resort,CO, Dec. 1995.

[NOB 97] NOBLE B., SATYANARAYANAN M., NARAYANAN D., TILTON J., FLINN J., WALKER K., "Agile Application-Aware Adaptation for Mobility", *Proc. of the 16th ACM Symp. on Oper. Syst. Princ.*, 1997.

[OMG 01] OMG, "The Common Object Request Broker - Architecture and Specifications. Revision 2.4.2", OMG Document formal/01-02-01, Feb. 2001, Object Management Group.

[PET 97] PETERSEN K., SPREITZER M., TERRY D., THEIMER M., DEMERS A., "Flexible Update Propagation for Weakly Consistent Replication", *Proc. 16th ACM Symp. on Princ. of Dist. Comp.*, Saint Malo, France, Oct. 1997, p. 288–301.

[Viv02] "The Vivian Consortium. VIVIAN deliverable report: Platform Services Specifications.", report , Sep. 2002, http://www-nrc.nokia.com/Vivian.