Weak Connectivity and Disconnected CORBA Objects on Hand-Held Devices

Denis Conan, Sophie Chabridon, Olivier Villin, and Guy Bernard

Institut National des Télécommunications, 9 rue Charles Fourier, 91011 Évry, France
{Denis.Conan|Sophie.Chabridon|Olivier.Villin|Guy.Bernard}@int-evry.fr

1 Introduction

With wireless communications and mobile hand-held or wearable devices becoming a reality, new applications where users can have access to information anytime, anywhere are made possible. Distributed applications development is facilitated by the use of standard middleware technology. Perhaps the most popular model is object-oriented middleware in which applications are structured into potentially distributed objects that interact via location transparent method invocation. Our focus is on the CORBA architecture standardised by the Object Management Group (OMG) [8].

Our main contribution is to propose a framework, called Domint, which adapts legacy CORBA applications so that they can keep working even when weakly connected or disconnected. Weak connectivity results from intermittent communication, low-bandwidth, high-latency or expensive networks. We distinguish between two kinds of disconnections: voluntary disconnections when the user decides to work on their own and involuntary disconnections due to physical wireless communication.

Important pioneering work has been done for mobile information access as surveyed in [3]. Our position is different with respect to the underlying technologies involved. Coda [6] and Odyssey [7] focus on file systems and Bayou [9] is mainly targeted at database systems. Not unlike Rover [4], we address object-based systems. However, our intention is to take advantage of the most recent standard techniques so that we can cater for a large number of applications. We propose to benefit from state-of-the-art middleware technology and validate our approach by demonstrating the feasibility of running an Object Request Broker (ORB) on a Personal Digital Assistant. This comes with all the advantages of language, operating system and network interoperability of CORBA middleware technology.

The remainder of this paper presents the Domint architecture in Section 2 and prototype performance in Section 3.

2 Design Rationale and 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. Keeping working while being disconnected implies transferring some elements from the servers to the mobile terminal before losing connectivity, logging operations or state changes during the disconnection, and re-integrating when re-connecting. This section first presents the design rationale and then the architecture of Domint.

CORBA is chosen for its ability to be used in multiple domains and for providing extensibility mechanisms such as portable interceptors to build application-transparent services. In order to make the functionality of application server objects available even when being disconnected, proxy objects that we call disconnected objects are created on the mobile terminal. Being an object means that both data and code are present thus the benefits of the mobile code technology can be valuable [2]. A disconnected object is a CORBA object which is similar in design and implementation to the remote object, but specifically built to cope with disconnection and weak connectivity. It is the application designer's responsibility to balance between a straightforward design and a more complex one that adapts better to connectivity variations. Disconnected objects, being CORBA objects, are accessible from all the applications of the mobile terminal and can use standard CORBA services such as naming, event notifications or transactions independently of the Domint framework.

In order to deal with multiple applications concurrently, some parts of resource management and log management (respectively the management of resources such as network bandwidth and the propagation of logged requests) are centralised and application-transparent. In addition, these services are achieved by CORBA objects and accept requests from the application for better adaptation. The application-aware resource management service abstracts, to applications in the CORBA world, connectivity information provided by the operating system. More precisely, it accepts requests to modify the per-application perception of which resources and resource levels correspond to bad, weak or strong connectivity, thus improving agility. The application-aware log management service is run by a CORBA object able to act as a representative of the disconnected objects during the re-transmission of the logged requests. It also accepts some code from the latter objects in the form of CORBA Objects By Value (OBV) to interpret the logged requests and to perform for example, log compaction, hence improving fidelity.

The architecture of Domint is depicted in Figure 1. The two sub-figures present UML-like collaboration diagrams of the client sending the first request to a remote object when the connectivity is strong and then sending a request in the case of weak connectivity, respectively. The collaborations are explained in [1]. In terms of entities, all the rectangles represent CORBA objects. The portable interceptor PI is also a CORBA object but a local one. All the requests from and the replies to the client are intercepted by the PI. On request sending, the PI acts as a switch between the disconnected object DO, and the remote object. When the client user interface (Client in the diagrams) starts, a client-side interceptor is registered at the creation of the ORB. Depending on the state changes of the logical connection, the client-side request interceptor can

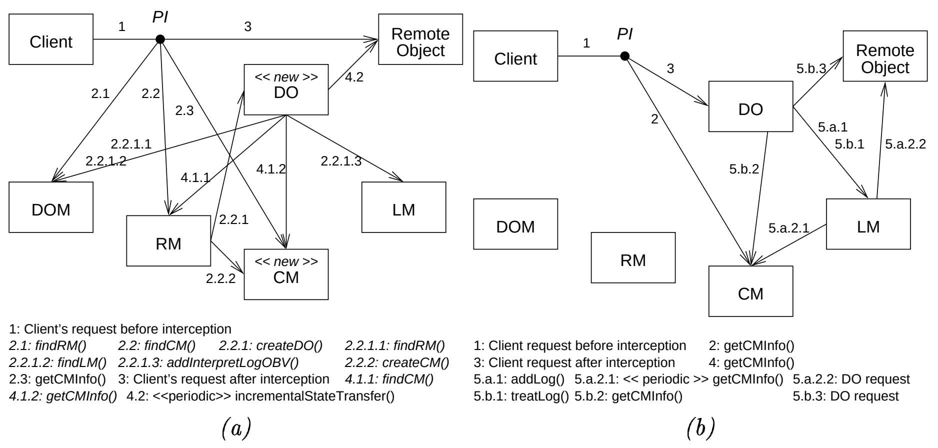
Client
PI
Remote Object
<< new >> DO
<< new >> CM
DOM
RM
LM
1
2.1
2.2
2.3
3
4.2
2.2.1.1
2.2.1.2
2.2.1.3
4.1.1
4.1.2
2.2.1
2.2.2
1: Client's request before interception
2.1: findRM() 2.2: findCM() 2.2.1: createDO() 2.2.1.1: findRM()
2.2.1.2: findLM() 2.2.1.3: addInterpretLogOBV() 2.2.2: createCM()
2.3: getCMInfo() 3: Client's request after interception 4.1.1: findCM()
4.1.2: getCMInfo() 4.2: <<periodic>> incrementalStateTransfer()
(a)
Client
PI
DO
Remote Object
DOM
RM
CM
LM
1
2
3
5.b.3
5.a.1
5.b.1
5.a.2.2
5.b.2
5.a.2.1
1: Client request before interception 2: getCMInfo()
3: Client request after interception 4: getCMInfo()
5.a.1: addLog() 5.a.2.1: << periodic >> getCMInfo() 5.a.2.2: DO request
5.b.1: treatLog() 5.b.2: getCMInfo() 5.b.3: DO request
(b)

Fig. 1. The Domint architecture: *(a)* interactions during the first call to a remote object in the case of strong connectivity, corresponding to the connected mode; the next client's requests sent in the connected mode do not generate the requests italicised; *(b)* interactions during a call to the same remote object in the case of weak connectivity, corresponding to the partially connected mode. The remote object is hosted on a machine of the wired network while all the other entities are executed on the mobile terminal.

build a CORBA `ForwardRequest` exception indicating the change of request target IOR and raise that exception. The exception is automatically managed by the ORB. The effect is a transparent switching of target object: from the remote object to the disconnected object and *vice versa*. The decision table contained in the PI for the transparent switching between modes is given in [1]. On response reception, the PI detects possible communication failures between the sending of the request and the reception of the response. The client and the PI belong to the same execution entity whereas the disconnected objects manager DOM, the resource manager RM, the connectivity manager CM, the DO and the log manager LM are grouped in another execution entity, also on the mobile terminal. Except the connectivity manager, the managers are single objects. The DOM is the entry point to find the other managers. The RM is a factory of CMs. A CM accomplishes 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. We can easily imagine other policies such as one CM per application or one CM per remote host.

A connectivity manager handles a logical connection between a client on the mobile terminal and a remote object on the wired network, however the number of wireless physical connections that link the two objects at a given time and regardless of whether the logical connection corresponds to different wireless physical connections over time. Connectivity managers rely on network monitoring entities that effectively measure the resource levels: network activity, available bandwidth, transmission cost, round-trip time... The monitoring can be provided by non-CORBA entities, and preferably, by the operating system. Connection monitors can be organised, for instance, in hierarchical structures controlled by a resource manager; the Domint's resource manager could be the representative in the CORBA world of this resource manager.

In order to not "*punish strongly-connected clients*" [6], while strongly connected, client's requests go directly to the remote object. Secondly, in order to "*insulate applications from insignificant variations in resource level*" [7], an hysteresis mechanism detailed in [1] is designed to avoid too frequent state transfers and switchings between the disconnected object and the remote server object. For this purpose we introduce a first-class partially connected mode, in addition to the connected and disconnected modes. Thirdly, in order to "*expose network connectivity to applications and permit applications*" and users "*to be involved in connectivity related decisions*" [4], users' interfaces can customise the constants of the hysteresis mechanism and can obtain and display connectivity information. Of course, users can disconnect or re-connect voluntarily by invoking operations `disconnect()` or `reconnect()`; these calls are not addressed to the connectivity managers, but to the remote objects, and are intercepted and treated by the PI.

The design of disconnected objects is highly application-dependent. [1] gives patterns for the development of such disconnected objects. We do not address consistency and conflict resolution problems in this paper. We have designed a generic log service that can easily be adapted to integrate domain specific consistency protocols; research is currently in progress to evaluate how the mechanisms of Bayou [9], ICeCube [5] or operation transforms [10] can benefit to the DOM service.

3 Performance Results

We have conducted a first 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. Each test was run 100 times so as to compute meaningful averages. A garbage collection occurs before each run on the client and server sides in order to have no interference with previous operations.

In Table 1, we show a subset of the performance results obtained on the client side with an iPAQ (206 Mhz, 16 Mb of ROM and 32 Mb of RAM) running Windows CE 2.0 and ORBacus 4.1.0 as the ORB for sending data. A first lesson is that the use of an ORB alone (line #3) induces little overhead as compared with standard remote TCP communication for data larger than 16Kb. Secondly, intercepting requests introduces a negligible cost with regard to the associated processing. In addition, lines #5 and #6 experiments teach us the following facts: 1) for data smaller than 16Kb, the overhead ranges from 30% to 100%, but in value, it stays imperceptible for the end user; 2) for data larger than 128Kb, the overhead is acceptable, extra Domint messages being short compared to large data messages. Finally, not shown in Table 1, the duration of a transparent switching performed by the ORB is estimated around 240ms.

Experiment	**1 b**	**128 b**	**16 Kb**	**128 Kb**	**512 Kb**	**1 Mb**
1. TCP local	7±0	8±0	39±13	248±6	960±4	1919±23
2. TCP remote	9±0	10±2	61±7	456±74	1878±214	3717±286
3. ORB conn. without PI	13±1	18±50	63±4	462±96	1843±48	3630±70
4. ORB conn. with PI doing nothing	17±2	17±2	66±5	471±104	1831±88	3675±200
5. ORB conn. with PI DOM	54±2	54±2	100±2	508±78	1912±128	3757±238
6. ORB part. conn. with PI DOM	55±2	55±2	103±3	504±83	1919±168	3798±205

Table 1. The performance results on iPAQ with Windows CE: times (ms) for sending data (bytes).

References

1. D. Conan, S. Chabridon, O. Villin, and G. Bernard. Domint: Weak Connectivity and Disconnected CORBA Objects on Hand-Held Devices. Technical report, Institut National des Télécommunications, Évry, France, Aug. 2002.
2. A. Fuggetta, G. Picco, and G. Vigna. Understanding Code Mobility. *IEEE Transactions on Software Engineering*, 24(5), May 1998.
3. J. Jing, A. Helal, and A. Elmagarmid. Client-Server Computing in Mobile Environments. *ACM Computing Surveys*, 31(2), Jun. 1999.
4. A. Joseph, J. Tauber, and F. Kaashoek. Mobile Computing with the Rover Toolkit. *IEEE Transactions on Computers*, 46(3), 1997.
5. A. Kermarrec, A. Rowstron, M. Shapiro, and P. Druschel. The IceCube Approach to the Reconciliation of Divergent Replicas. In *Proc. 20th ACM PODC*, Aug. 2001.
6. L. Mummert, M. Ebling, and M. Satyanarayanan. Exploiting Weak Connectivity for Mobile File Access. In *Proc. 15th ACM SOSP*, Dec. 1995.
7. B. Noble, M. Satyanarayanan, D. Narayanan, J. Tilton, J. Flinn, and K. Walker. Agile Application-Aware Adaptation for Mobility. In *Proc. 16th ACM SOSP*, 1997.
8. OMG. The Common Object Request Broker - Architecture and Specifications. Revision 2.4.2. OMG Document formal/01-02-01, Feb. 2001.
9. D. B. Terry, M. M. Theimer, K. Petersen, and A. J. Demers. Managing Update Conflicts in Bayou, a Weakly Connected Replicated Storage System. In *Proc. 15th ACM SOSP*, Dec. 1995.
10. N. Vidot, M. Cart, J. Ferri, and M. Suleiman. Copies convergence in a distributed real-time collaborative environment. In *Proc. ACM CSCW*, Dec. 2000.