Handling Network Roaming and Long Disconnections at Middleware Level

Denis Conan, Sophie Chabridon, Olivier Villin and 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

Andrei Kotchanov and Titos Saridakis
Nokia Research Center
PO Box 407
FIN-00045 Nokia Group, Finland
{Andrei.Kotchanov|Titos.Saridakis}@nokia.com

23rd August 2002

Abstract

As personal devices (PDAs, hand-held computers and smart-phones) become more and more indispensable in our professional and personal life, the need to deal with network roaming and long disconnections rises as one of the primary concerns of the application developer. This paper suggests a way to manage at the middleware level the concerns related to network disconnections. As a result, the application developer is alleviated from explicitly dealing with network switching and re-binding to application servers after a disconnection period. Our approach allows the application developer to define the type of connectivity (strong, weak, or null) as well as the candidate network types (e.g. Bluetooth, WLAN, GPRS) over which connections will be possible and the rules that will cover the network switching attempts. The presented approach builds on the combination of two middleware services based on CORBA which deal with network roaming (Implicit Connectivity Management or ICM) and long disconnections (Disconnected Object Management or DOM).

Keywords: disconnection, middleware, nomadicity, roaming.

1 Introduction

Since the early 90's, the area of mobile computing has witnessed tremendous research and technology advances. 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. 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 [MES95]; it results from intermittent communication, low-bandwidth, high-latency or expensive networks.

A similar functionality to the middleware is necessary when the mobile terminal roams on a number of different networks (*e.g.* from GPRS to Bluetooth). It is desirable that in such cases, an application connected to a remote application service over a given network can continue interacting with the same remote application service although the communication link is now switched to another network. The physical component, which provides the application service, might be a different one on each network, but the interfaces of these two components must be the same. This would allow the application to seamlessly switch to a network with ”better” characteristics (*e.g.* cheaper access charges, higher bandwidth, higher security level).

The wide diversity of applications and terminal resources demands flexibility in the choice of weak connectivity handling means. Some applications need only support for re-connection to the remote application service, the user does not need to work during disconnected period, whereas other applications require the possibility to continue working while being disconnected. Some terminals provide access through wireless network media with a relatively high disconnection probability, for which disconnection handling is important (*e.g.* Bluetooth), while other terminals provide access only through long-range network media with wide coverage and their own hand-over support (*e.g.* GPRS). In this case the possibility to adapt to the variations in the data transfer speed is more important than the disconnection handling.

This paper presents two generic services, an Implicit Connectivity Management (ICM) service and a Disconnected Object Management (DOM) service, which can be used either together or separately, providing suitable level of disconnection / weak connectivity handling. The ICM detects network disconnections and provides distributed applications with the possibility to re-connect, whereas the DOM serves for application-aware adaptation in addition to application-transparent adaptation. The DOM enables distributed applications to keep working in a transparent manner even when weakly connected or disconnected and it can use the ICM for the disconnection handling.

In the remainder of this paper, Section 2 exemplifies the usage of the two services through the presentation of three scenarios. The ICM and the DOM services are briefly discussed in Section 3 and Section 4 respectively. Next, Section 5 introduces the overall architecture with the interactions between the application and the DOM and ICM services, and the collaborations between the two services. Finally, we survey related work in Section 6, and conclude in Section 7.

2 Scenarios

This section presents three different scenarios demonstrating the usage of the two services developed in this paper, both each service separately and the two services together. The first scenario in Section 2.1 is mostly targeted at the DOM, which uses only a part of the ICM functionality, while the second scenario in Section 2.2 shows mainly the ICM roaming capabilities. In Section 2.3, the third scenario is the one that illustrates the ICM-DOM integration.

2.1 Involuntary disconnection

Both DOM and ICM services can be used to deal with involuntary disconnections. The user starts a distributed application while being connected to the fixed network through a wireless networking media (*e.g.* Bluetooth and GPRS), and then moves out of the reach of the base station. The ICM service monitors the network, catches the disconnection event, and informs the DOM service that sending data is impossible. In the mean time, while being strongly connected, the DOM service prepares for

this involuntary disconnection by transferring necessary objects from the wired hosts to the mobile terminal. The user continues working while being involuntarily disconnected. Later on, the same user moves again, this time back into the reach of the base station. The ICM service periodically probes the network for re-connection. When a network re-connection is established, ICM notifies the DOM, and the latter transparently transfers the operations logged during the disconnection and updates the server objects on the wired network.

2.2 Roaming

The mobile terminal can connect to the fixed network through different access points. They can belong either to different wireless networking media (*e.g.* Bluetooth and GPRS), or to the same networking medium (*e.g.* distinct remote Bluetooth neighbourhoods connected through a wired network). In this case the user continues working while moving and switching from one access point to another. The ICM service catches the disconnection and re-connects to another access point. But now the ICM service should check if the server objects on the wired network are available through the new access point, and only after that informs the DOM about the possibility to send data. The DOM service can either, like in the previous example, hide the disconnection period (if the terminal moves from one Bluetooth access point to another without switching the networking media), or allow for the application to adapt to various data throughputs of different wireless media.

2.3 Voluntary disconnection

Both DOM and ICM services can cooperate to deal with voluntary disconnections. The DOM service can be used to provide voluntary disconnection for one application among other running applications. The application graphical user interface is adapted to include a button to allow voluntary disconnection/re-connection. The user starts the distributed application while connected and then moves while working. For minimising the probability of unexpected disconnections, the user prefers to disconnect the application by clicking on the “disconnect” button (the DOM deactivates the ICM at this stage —*i.e.* the ICM stops monitoring networks on behalf of the application). In this scenario, the user may still be connected to wired hosts and use other distributed applications. The user continues working with the disconnected application and re-connects that application (the DOM activates the ICM for this) before terminating their work. When re-connecting, the DOM service transparently transfers the operations logged during the disconnection and updates the server objects on the wired network.

The DOM service can also be used to provide voluntary disconnection for all the applications (adapted to use the DOM service) launched by the user on the mobile terminal. A specific (small) application is built and launched on the mobile terminal to include a button, that we call ”global disconnect” button, to allow voluntary disconnection/re-connection of all the applications in only one user’s action. This “global disconnect” button could be placed in the application-independent screen area, exemplified by the task bar of a windows manager.

3 Implicit connectivity management

The objective of the ICM service is to provide some basic disconnection handling functionality, so that any applications, including those that do not need the application service continuity while disconnected, could use ICM. ICM must introduce minimum run-time overhead for the applications, and it is not responsible for the behaviour of the applications while in disconnected mode. However, it should enable the development of other middleware services on top of it (the DOM service is an example). Such

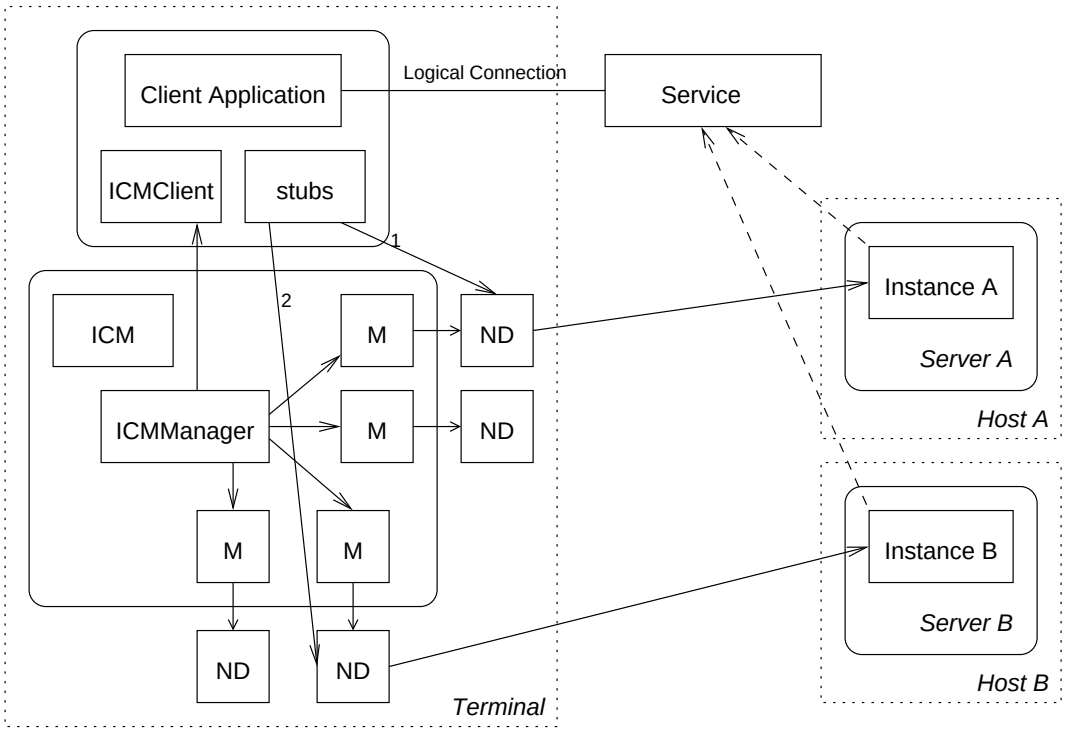


Figure 1: The architecture of ICM. “M” on the picture designates a monitor and “ND” stands for a network driver.

services can handle the outgoing application communication (*e.g.* buffering one-way communication) in order for the application to continue functioning without blocking. Section 3.1 presents the ICM service architecture. The interface specification of the ICM service is given in Section 3.2.

3.1 Architecture of the ICM

The ICM service architecture is presented in Figure 1. It depicts an example where the application running on the mobile terminal on the left side of the picture has established a logical connection with the application service at the right side of the picture. The logical connection is realised over a physical connection which starts by the link labelled `1` in the terminal and connected to the application service instance running on server `A`.

ICM monitors different types of networks available to the mobile terminal in order to enable the seamless switching among them for the applications which have requested ICM services. At a certain point in time, either the current network connection is interrupted or another network monitor informs ICM that a “better” (*e.g.* cheaper communication costs, higher bandwidth, *etc*.) network is available. When a network disconnection occurs on the link, which is used by the application, ICM detects the disconnection event and notifies either the application or some other service, which deals with the disconnection on behalf of the application. ICM keeps monitoring the network activity and periodically attempts to re-connect. If ICM fails to re-connect after a predefined time-out, it sends a notification informing the application that the network is permanently unavailable. When ICM re-connects, it contacts the Naming or Trading Service to check whether an instance of the same application service as the one the application was using before disconnection is available on the new network. If that is the case, ICM appropriately updates the application binding information (which is kept in the stubs) and

notifies the application (or the service managing disconnected operation, *e.g.* the DOM) that a network connection is again available. Now, the application can use the application service again without having to take any re-initialisation actions over the connection, which is established with the application service instance on the new network (link labelled 2 inside the terminal, which leads to a connection to server B)

If application-, terminal- or user-specific settings compel an order of preference regarding the network to be used for a connection to a remote application service, ICM can take the same re-connection actions as above even for connections that are active over another, less preferred network link. Note that ICM is interacting with the application only when appropriate conditions regarding the availability of new network bearers are met. Otherwise, it does not participate in the application to application service communication.

3.2 Interfaces of the ICM

ICM consists of two distinct parts: the one that is responsible for monitoring the connectivity at the network level (this corresponds to "monitors" designated by letter "M" in Figure 1) and the one that is responsible for interacting with the application (this corresponds to interfaces "`ICM`", "`ICMManager`", and "`ICMClient`" in Figure 1). This section briefly describes the ICM interfaces and their operations; monitors are not directly accessible by applications, they are configured through `ICMManager`; detailed description of the ICM service is presented in [Viv02].

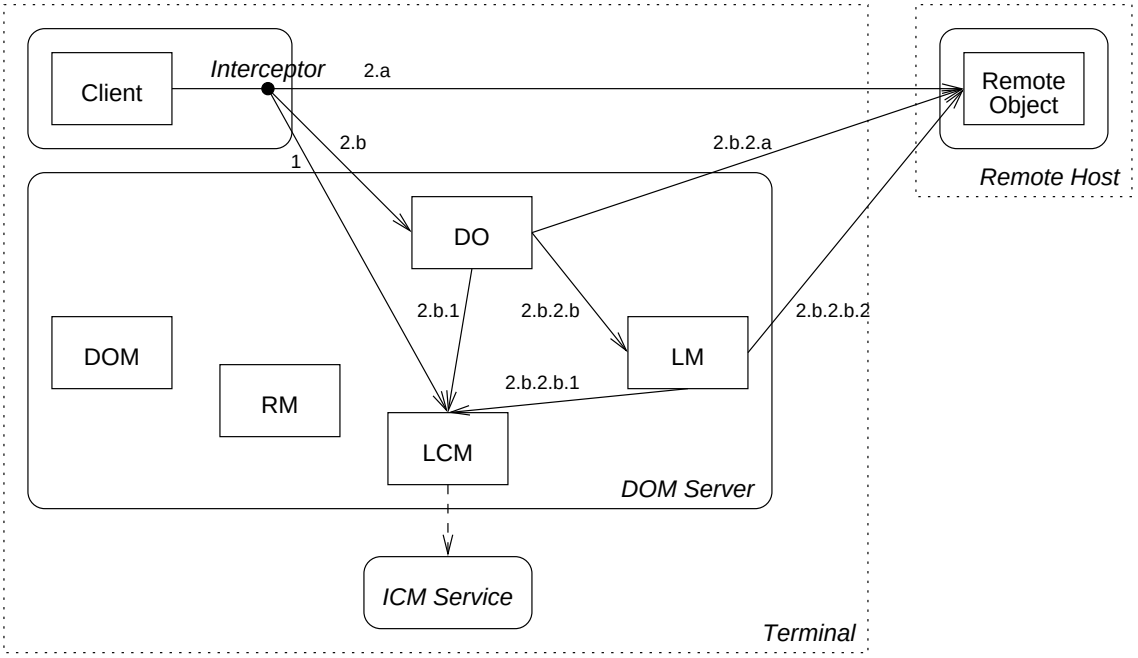
The `ICM` interface is the entry point of the ICM service. Applications use its `get_connectivity_manager()` operation to get an `ICMManager` object. The `disconnect_connectivity_manager()` operation can be used when the application does not need the ICM service anymore. The `get_networks_supported()`, `get_monitoring_interval()`, and `get_timeout()` operations are used to get the service default monitoring configuration.

The `ICMManager` interface provides an application with an access to the monitoring and re-connection functionality and with a support of application level settings. The application informs ICM regarding the application services it is interested in with the usage of the `add_service()` operation. If after the re-connection the ICM service discovers that the IOR (Interoperable Object Reference) of the application service has changed, the server part of the ICM service notifies its client part, `ICMClient`, of the event, where-after the client part fetches the new IOR from the `ICMManager` using the `get_ior()` operation, and then changes the IOR record in the stub. In order to get the information about which networks can currently be used for making connection, an application should call the `detect_connectivity()` operation. The application uses the `deactivate_manager()` and `activate_manager()` operations to temporarily deactivate and afterwards to again activate the ICM service with the same settings or to resume monitoring after receiving the `timeout` notification from the ICM service.

The `ICMClient` interface is implemented by the application and its `notify()` and `timeout()` operations are called by the ICM service to notify the application of the disconnection/re-connection and time-out events.

4 Disconnected object management

The objective of the DOM service is to deal with involuntary and voluntary disconnections. Without user's intervention, the system prepares for disconnection by transferring the necessary objects from the wired hosts to the mobile terminal. The user continues working while being disconnected. When re-connecting, the DOM service transparently transfers the operations logged during the disconnection



1: getCMInfo()
2.a: Client's request to the Remote Object
2.b: Client's request to the DO
2.b.1: getLCMInfo()
2.b.2.a: DO's request to the remote object
2.b.2.b: addLog()
2.b.2.b.1: <<periodic>> getLCMInfo()
2.b.2.b.2: DO's request to the remote object

Figure 2: The architecture of DOM.

and updates the server objects on the wired network. The DOM service was introduced in [CCB02] where we detailed the CORBA mechanisms (Objects By Value and Portable Interceptors) used in the design. In this section, we describe the architecture and the offered services in relation to the ICM service. Section 4.1 presents the DOM service architecture. The interface description of the DOM service is given in Section 4.2.

4.1 Architecture of the DOM

Figure 2 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).

All the rectangles in Figure 2 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 users want application-aware adaptation, the DOM is the entry point of the DOM service to find the

other managers. The resource manager RM is a factory of logical connectivity managers LCM. An LCM realises the abstraction of connectivity information related to one resource. The policy currently implemented associates one LCM to each logical link between a client and a remote object.

The interceptor obtains the connectivity information from the LCM (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). In the connected mode, clients experience no more penalty than the duration of the interceptions on the round-trip-time of their calls. 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.

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 LCM for connectivity information (2.b.1). We give two possible ends to this scenario: in the partially connected mode (2.b.2.a), the operations are immediately remotely transmitted; and in the disconnected mode (2.b.2.b), the operations are logged in the log manager LM to be remotely transmitted when the re-connection occurs. 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.

Finally, not shown in Figure 2, users can disconnect or re-connect voluntarily by calling operations `disconnect()` or `reconnect()`.

4.2 Service of the DOM

The DOM interfaces serves to configure the DOM service on the client terminal. This section briefly describes the DOM interfaces; detailed description of the DOM service is presented in [Viv02].

The DOM is the entry point of the DOM service. Its interface name is `DOManager`. `findLogManager()` and `findResourceManager()` methods serve to obtain the reference of the log manager and the resource manager. The method `disconnect()` is called when clicking on the global disconnect button so that distributed applications executing on the mobile terminal disconnect voluntarily.

The resource manager centralises the control of all the resources of the mobile terminal. The DOM service can deal with resource criteria related to the connectivity: network activity, available bandwidth, transmission cost, round-trip time, but also battery power. In the DOM-ICM integration, the DOM service relies on ICM resource criteria. The interceptor, the disconnected object and the log manager obtain the references of connectivity managers thanks to the `findLogicalConnectivityManager()` method of the resource manager. Logical connectivity managers are created and destroyed by the interceptor with the `createLogicalConnectivityManager()` and `destroyLogicalConnectivityManager()` operations.

The client can read and customise the constants of the hysteresis mechanism in order to specify which resources and resource levels correspond to bad, weak, or strong connectivity, thus improving agility (*Cf.* Section 5.2 for the description of the hysteresis mechanism). Thus, all this information is provided as attributes in the `LogicalConnectivityManager` interface: `lowDown`, `highDown`, `lowUp` and `highUp` attributes for the constants of the hysteresis; `resLevel`, `state`, `mode`, `forward`, `remoteObject` and `localObject` read-only attributes for the connectivity information; `voluntaryDisc` attribute to indicate whether the possible disconnection is voluntary.

The log manager is responsible for managing logged data: storing, compacting and, flushing with the methods `addLogRequest()`, `compactLog()`, and `treatLog()`. The data are organised by remote object

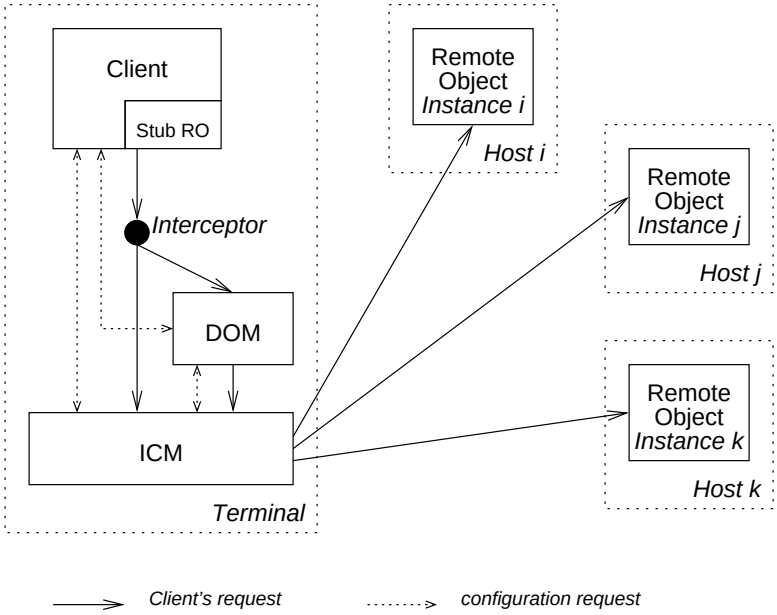


Figure 3: The DOM - ICM collaboration architecture.

reference. Logging can concern either subsets of the remote object state that is to be updated or operations to be performed on the remote object. The log manager cannot interpret the logged data, thus it receives some code and data (object by value conforming to an abstract interface in CORBA IDL) from the disconnected object via the method `receiveOBV()` to manipulate data containers (`any`s in CORBA). Finally, the `treatLog()` method takes a boolean value as an `in` that indicates whether the flushing is pessimistic (trying to flush all the log at once) or optimistic.

We do not address consistency and conflict resolution problems in this article. 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 [TTPD95], ICeCube [KRSD01] or operation transforms [VCFS00] can benefit to the DOM service.

5 Architecture

5.1 DOM - ICM Collaboration

The generic architecture of DOM and ICM collaboration is based on the following model. An application runs on a mobile terminal, which offers various network capabilities (*e.g.* wireless like Bluetooth, WLAN, GPRS, and IrDA but also wired like RS232, USB, and PCMCIA-based LAN access). The application contacts some application service outside the mobile terminal by establishing a network connection to the remote server where the application service resides. At some moment in time the network connection is interrupted without any prior notification to the application. Later on, and while the application is still running, a network connection over the same or a different network bearer becomes available and the server, providing the service that the application was using before the network connection interruption, is accessible through that network. The application must be able to continue working while being disconnected and to resume interaction with the application service after re-connection, although the actual network connection may be different from the initial one.

The way DOM and ICM collaborate is graphically illustrated in Figure 3. In a classical distributed application with strong connectivity, the client part is loaded on the mobile terminal and the server objects are hosted on machines of the wired network. The DOM is responsible for the application service continuity while disconnected, what 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.

The application is said to have established a logical connection with an application service when it has found and bound to a remote application service and while it has not explicitly shut down the connection to it. The ICM service is responsible for ensuring the re-establishment of a physical connection when the initial one corresponding to a logical connection is interrupted.

The DOM service acting on behalf of the application can configure ICM and use ICM monitoring and re-connection functionality. The logical connectivity managers (*Cf*. Figure 2) of the DOM service possess an `ICMClient` object of the ICM service so that they are notified of involuntary physical disconnections. In addition, the logical connectivity managers can have access to the monitoring and reconnection functionality of the ICM service: *e.g.* the list of available network bearers with the priority of each of them, the current physical connectivity, the monitoring interval of the network connection status, the time-out value which limits the whole period of physical re-connection attempts. Finally, voluntarily disconnecting (respectively re-connecting) a distributed application running on the mobile terminal means deactivating (respectively activating) the management part of the ICM service, keeping the monitoring part still running, by calling the `deactivate_manager()` (respectively `activate_manager()`) operation of `ICMManager` objects. Consequently, radio silence is not obtained by voluntarily disconnecting all the distributed applications, but by deactivating the monitoring part of the ICM service.

5.2 Connectivity Management

The DOM service and the ICM service both contain connectivity managers. The DOM connectivity manager presents an abstract logical view of connectivity information, and each application can define what is strong, weak, or null connectivity. The ICM connectivity manager deals with network bearers and network connection status at a lower level than DOM connectivity managers. In order to distinguish the two kinds of connectivity managers, the DOM connectivity managers (*resp.* ICM connectivity managers) are called logical connectivity managers (*resp.* physical connectivity managers).

Logical connectivity managers (in the DOM entity) handle logical connections between clients on the mobile terminal and remote objects on the wired network, however the number of wireless physical connections that link the objects at a given time and regardless whether the logical connections correspond to different wireless physical connections over time. In Figure 4, the horizontal axis represents the instantaneous available communication resource level. One bearer may correspond to only a small interval of the resource level values or may span all the values. Several bearers may need to be aggregated to cover all the needed/possible values. The ICM service provides the DOM service with instantaneous values and the DOM connectivity management has the responsibility to present to the application the corresponding abstracted connectivity mode, positioning instant values on the axis and taking into account previous sampled values.

In order to "insulate applications from insignificant variations in resource level" [NSN+97], the logical connectivity managers rely on an hysteresis mechanism for smoothing variations in resource availability (*cf.* Figure 4). The hysteresis defines the following three modes: *disconnected*, *connected*, and *partially connected*. On diagram 4.1, when the resource level increases and is lower than `lowUp` (resp. `highUp`), the mobile terminal is disconnected (resp. partially connected). When the resource level decreases but is higher than `highDown` (resp. `lowDown`), the mobile terminal is connected (resp. par-

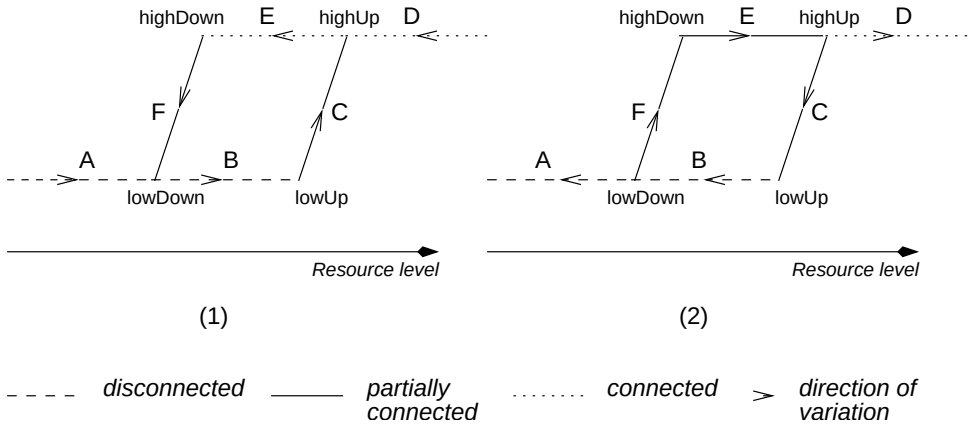


Figure 4: The hysteresis of logical connectivity management.

tially connected). Without diagram 4.2, observe that there would still exists a risk of "ping-pong effect" around the `highDown` value. Thus, when the connection reaches state `F` from state `E` and the resource level becomes higher than `highDown`, the mode remains "partially connected" up to the `highUp` value. In addition, the hysteresis keeps evolving whether or not the client asks for a voluntary disconnection. So, a `forward` boolean variable is computed to indicate whether the disconnected object and the log manager can transmit the DO requests. This variable depends on the voluntary disconnection occurrence.

6 Related work

Concerning long disconnection handling, the weak connectivity of mobile environments in conjunction with the relative resource poverty of hand-held devices lead to a trade-off between autonomous applications and interdependent distributed applications. This trade-off is well explained in [NSN+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 [JTK97, MES95, NSN+97, PST+97, TTPD95] 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, to allow concurrency of accesses to resources and to specify fidelity in terms of data consistency. An important pioneering work has been done for mobile information access as surveyed in [JHE99]. Our positioning is different with respect to the underlying technologies involved. Coda [KS92, MES95] and Odyssey [NSN+97] focus on file systems and Bayou is mainly targeted at database systems [PST+97, TTPD95]. Not unlike Rover [JTK97], we address object-based systems. However we exploit further the object paradigm; it allows us to log either subsets of the remote object state that is to be updated or operations to be performed on the remote object. The presence of proxy objects on the mobile terminal allows us to introduce a third mode —*i.e.* partially connected— and fully explore new methods to manage transitions between strong, weak, or null connectivity.

In the CORBA context, Π^2 [RSK00] and *ALICE* [HCC99, Lyn99], as well as Wireless CORBA [OMG01], focus on the management of the hand-off due to terminal mobility during the execution of a distributed application, and thus address the problem of dealing with short-time disconnections. Π^2 introduces two proxies (one on the terminal and one running in the wired network) for making involuntary

disconnections transparent to the user. By contrast, our design does not imply any proxy installation in the wired network, and we provide a support for voluntary disconnected mode. *ALICE* uses a proxy too for handling terminal mobility, and provides a mechanism for supporting both involuntary and voluntary disconnections. In addition, the level at which disconnections are handled in our proposition is different: in *ALICE*, when a disconnection occurs, an exception is sent by the ORB to the client, so that the appropriate code for switching to the disconnected mode has to be included in the clients; in our approach, disconnection events are trapped at the ORB level through the portable interceptor mechanism, so that the appropriate code is included in the portable interceptors, leaving the legacy application code unchanged. Of course, in order to improve agility, interfaces are provided to configure the services. In Wireless CORBA [OMG01], the transport end-point detection as well as the (re)connection procedure are out of the scope of the wireless CORBA specification. The wireless network disconnection/re-connection, access point and/or connection type switching can happen at any time: during an idle period when no data are sent or received, during a request transmission, after the request has been sent but before the response is received, and during the response receiving. As a result, a need emerges to provide a middleware-level service, able to manage in an implicit way the connectivity of the applications running on a mobile terminal.

7 Conclusion

This paper proposes the integration of the ICM and the DOM middleware services into a generic service. The integration of these two services enables the handling of network roaming and long disconnections in a way transparent for the application developer. The two services can be used both together and separately, providing suitable level of disconnection / weak connectivity handling.

The ICM service is responsible for ensuring the re-establishment of a physical connection over the same or another network bearer. The application level connectivity is established after re-connection without having to explicitly take any re-initialisation actions. The ICM monitors networks available to the mobile terminal, detects disconnection and re-connection events and notifies either the applications or the service managing disconnected operation, *e.g.* the DOM. The ICM uses the Naming or Trading Service to check whether an instance of the same application service as the one the application was using before disconnection is available on the new network. If that is the case, the ICM appropriately updates the application binding information. The ICM is interacting with the application only when appropriate conditions regarding the availability of new network bearers are met. Otherwise, it does not penalise the application execution with any overhead.

The DOM service enables distributed applications to keep working even when the distributed parts are weakly connected or disconnected. The DOM service deals with involuntary and voluntary disconnections. Without user's intervention, the system prepares for disconnection by transferring the necessary objects from the wired hosts to the mobile terminal. All the requests from and the responses to the client are intercepted. The interceptor detects possible communication failures between the sending of the request and the reception of the response. The information on network resource level is obtained from the service managing the network bearers, *e.g.* the ICM. The DOM service presents an abstract logical view of connectivity information to applications, and each application can define what is strong, weak, or null connectivity —*i.e.* disconnection at the application level. 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 local "mirrors" of the remote server objects. The user continues working while being disconnected. When the service managing the network bearers indicates a reconnection, the DOM service transparently transfers the operations logged during the disconnection and updates the server objects on the wired network.

The DOM-ICM integration provides substantial benefits to distributed applications running in mobile environments. It provides the user with all possible combinations of these four facilities: 1) roaming with the same network bearer or with different bework bearers, 2) short or long disconnections, 3) volontary or involuntary disconnections and 4) off-line disconnected work or reconnection to a different server. We introduced three different scenarios but many more possibilities exist as the service configuration allows for a flexible tuning. The prototypes of both services demonstrate that some today hand-held devices and *a fortiori* future mobile devices can embed the frameworks including the services and a complete ORB. A number of tests that were run on a combination of Linux laptops and PDAs showed that performance overhead of the ICM and the DOM services are negligible for the end user.

References

[CCB02] Denis Conan, Sophie Chabridon, and Guy Bernard. Disconnected Operations in Mobile Environments. In *Proc. 2nd IPDPS Work. on Par. and Dist. Comp. Issues in Wireless Networks and Mobile Computing*, Ft. Lauderdale, Florida, April 2002.

[HCC99] M. Haahr, R. Cunningham, and V. Cahill. Supporting corba applications in a mobile environment. In *Proc. of the 5th Annual ACM/IEEE International Conference on Mobile Computing and Networking (MOBICOM'99)*, pages 36–47, 1999.

[JHE99] J. Jing, A. Helal, and A. Elmagarmid. Client-Server Computing in Mobile Environments. *ACM Comp. Surveys*, 31(2), June 1999.

[JTK97] A.D. Joseph, J.A. Tauber, and F. Kaashoek. Mobile Computing with the Rover Toolkit. *IEEE Trans. on Comp.*, 46(3), 1997.

[KRSD01] A.M. Kermarrec, A. Rowstron, M. Shapiro, and P. Druschel. The icecube approach to the reconciliation of divergent replicas. In *Proc. of the 20th ACM Symp. on Princ. of Dist. Comp.*, Newport, Rhode Island (USA), Aug. 2001.

[KS92] J.J. Kistler and M. Satyanarayanan. Disconnected Operation in the Coda File System. *ACM Trans. on Comp. Syst.*, 10(1), Feb. 1992.

[Lyn99] N. Lynch. Supporting disconnected operation in mobile corba. Master's thesis, Trinity College Dublin, 1999.

[MES95] L.B. Mummert, M.R. Ebling, and M. Satyanarayanan. Exploiting Weak Connectivity for Mobile File Access. In *Proc. of the 15th ACM Symp. on Oper. Syst. Princ.*, Copper Mountain resort,CO, Dec. 1995.

[NSN+97] B.D. Noble, M. Satyanarayanan, D. Narayanan, J.E. Tilton, J. Flinn, and K.R. Walker. Agile application-aware adaptation for mobility. In *Proc. of the 16th ACM Symp. on Oper. Syst. Princ.*, 1997.

[OMG01] OMG. Wireless Access and Terminal Mobility in CORBA. Adpoted Specification. OMG Document dtc/01-06-02, Object Management Group, June 2001.

[PST+97] Karin Petersen, Mike J. Spreitzer, Douglas B. Terry, Marvin M. Theimer, and Alan J. Demers. Flexible Update Propagation for Weakly Consistent Replication. In *Proc. 16th ACM Symp. on Princ. of Dist. Comp.*, pages 288–301, Saint Malo, France, October 1997.

[RSK00] R. Ruggaber, J. Seitz, and M. Knapp. π^2 - A Generic Proxy Platform for Wireless Access and Mobility. In *Proc. 19th ACM Symp. on Princ. of Dist. Comp.*, Portland, Oregon, July 2000.

[TTPD95] Douglas B. Terry, Marvin M. Theimer, Karin Petersen, and Alan J. Demers. Managing Update Conflicts in Bayou, a Weakly Connected Replicated Storage System. In *Proc. 15th ACM Symp. on Princ. of Dist. Comp.*, pages 172–183, December 1995.

[VCFS00] Nicolas Vidot, Michelle Cart, Jean Ferri, and Maher Suleiman. Copies convergence in a distributed real-time collaborative environment. In *Proceedings of the ACM 2000 Conference on Computer Supported Cooperative Work*, pages 171–180, Philadelphia, PA USA., December 2000.

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