Towards a Generic Architecture for Mobile Object-Oriented Applications

Mads Haahr, Raymond Cunningham and Vinny Cahill

Distributed Systems Group
Department of Computer Science
University of Dublin, Trinity College
Dublin 2, Ireland

{Mads.Haahr,Raymond.Cunningham,Vinny.Cahill}@cs.tcd.ie

Abstract—**Our previous work in mobility support for CORBA applications resulted in the design and implementation of the Architecture for Location-Independent CORBA Environments (ALICE). The first version of ALICE enabled CORBA objects running on mobile devices to interact transparently with objects hosted by off-the-shelf CORBA implementations without relying on a centralised location register to keep track of their whereabouts. This paper presents the second version of ALICE on which work is currently ongoing. The improved architecture retains the features of the original and adds support for disconnected operation in the form of caching of server functionality on the client side. Furthermore, the architecture is being generalised beyond CORBA in order to make it applicable to other distribution infrastructures, such as Java RMI and DCOM.**

Keywords—**Mobile Computing, Middleware, CORBA.**

I. Introduction

ALICE is an architectural framework that provides mobility support for a certain suite of client/server application-level protocols whose characteristics are explained in section I-B. ALICE enables implementations of such protocols to provide their own support for mobile clients, servers (including address translation and location management) and disconnected operation (including replication and caching of server functionality). In addition, ALICE includes connectivity management features to address the difficult network characteristics of wireless networks. Developers use the ALICE framework by writing a series of software modules to interface with a set of core ALICE modules. This paper describes the overall architecture and the core modules independently of any particular application-level protocol.

A. Mobility Challenges

Our previous work [5] identified three areas where operation in mobile environments poses a challenge compared to traditional (wired, fixed) environments:
Device Limitations of the mobile host itself in the form of limited processing power, battery life, memory restrictions, etc. These limitations require software for mobile hosts to use as few resources as possible.
Network Characteristics of mobile hosts are generally diverse and varying compared to those of fixed hosts. Even small mobile devices (such as handheld PCs) typically have at least two (and typically more) communications interfaces that are not only connected to various physical endpoints at various points in time but also vary dramatically with regards to latency, bandwidth, reliability and usage cost.
Physical Host Mobility causes connection endpoints to wired networks to change frequently. In the case of mobile servers this can cause server references held by clients to become obsolete rapidly.

In general, addressing the latter two issues (by increasing mobility support) is likely to increase the footprint on the mobile host, and there is likely to always be a trade-off involved. The approach taken in ALICE has been to solve these two in separate protocol layers while keeping the footprint as small as possible. For example, effort has been made to allow as much functionality as possible to be placed in the fixed network rather than on the mobile host.

B. Protocol Requirements

For an application-level protocol to fit into the ALICE framework, it must support certain features:
1. The protocol must be client/server oriented.
2. The underlying transport must be TCP/IP.
3. A server reference needs to contain several endpoints at which the server can be found. Clients should try endpoints in order.
4. It must be possible to store some extra information in a server reference.
5. Forwarding of client requests towards a different server location must be possible.

An example of such a protocol is the CORBA Internet Inter-Orb Protocol (IIOP) with which the first version of ALICE [5] was tested. Work is currently ongoing on supporting Java RMI and DCOM in a similar fashion.

C. Mobility Model

The model for communications used in the ALICE framework is that mobile hosts connect to remote hosts via mediators, or base stations, called *mobility gateways*, as shown in

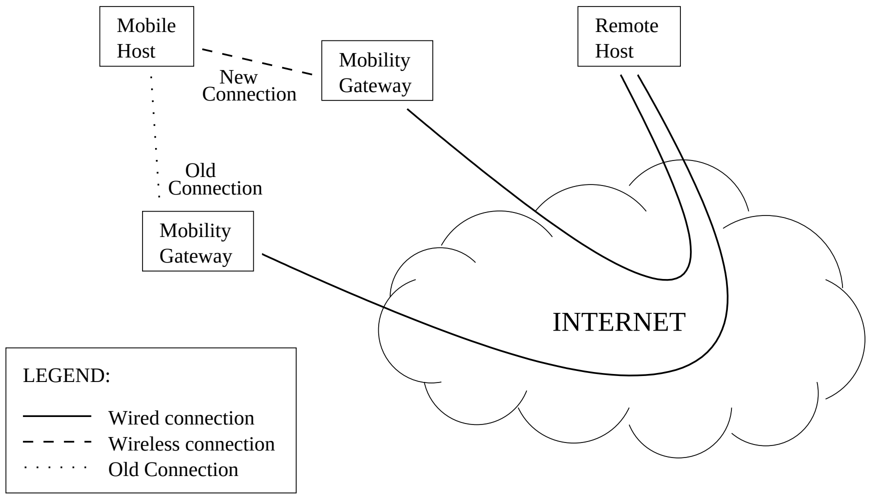

Fig. 1. The ALICE Mobility Model

figure 1. (Hence, we are not talking about ad-hoc type communcation between mobile devices.) The terminology is as follows:

MH Mobile Host. A device that moves between mobility gateways and is disconnected while in transit. A MH can hold programs that act as servers as well as clients.

MG Mobility Gateway. A fixed computer that acts as a base station for MHs. It has at least one interface through which a MH can connect in addition to a wired network connection to a LAN or the Internet.

RH Remote Host. A computer with which the MH communicates via the MG as mediator. A RH can contain client as well as server programs. It can be a fixed host or a mobile host. In the latter case, it is assumed to communicate via a MG.

This is the general ALICE mobility model. It may sometimes be preferable to connect a MH directly to the network rather than go through a mediator. In such situations, the MH can act as its own MG.

II. ALICE Overview

This section gives an overview of ALICE by first explaining non-standard notation and terminology used to describe the architecture and then presenting the architecture itself. ALICE consists of three basic pieces of functionality which are briefly introduced as part of the overview and treated in more detail in sections III to V.

A. API Notation and Terminology

Each of the ALICE layers has up to three different APIs: the *downcall*, *upcall* and *tuning* APIs. If a layer performs services for layers above it in the protocol stack, it will have a *downcall* API through which invocations from upper layers are received. If a layer receives upcalls (or callbacks) from layers below it, the layer also has an *upcall* API through which this happens. If a layer is runtime configurable (as some of the ALICE layers are), it has an additional *tuning* API for this purpose.

The notation used for describing the interaction between the ALICE layers is an extension of the traditional protocol stack notation, where the layers are placed on top of each other and communcation takes place vertically between adjacent layers. The notational extensions to this model include adding edges onto the protocol boxes to illustrate the three types of APIs. This is shown in figure 2 where two layers, each having all three types of APIs, interact.

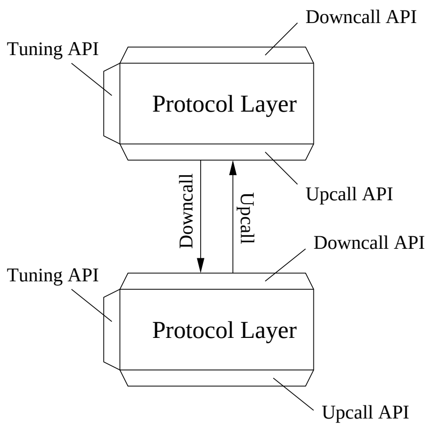

Fig. 2. API Notation

B. Layer Notation and Terminology

In total, ALICE consists of three different layers. Each of these consists of a number of components residing in different places. For example, the Mobility Layer (ML) consists of two components: one residing on the MH and one on the MG. When discussing an entire layer (say, the ML) it will be referred to either by its full name (the *mobility layer*) or its abbreviated name (the *ML*). When discussing a single component of a layer, the location at which it resides will be subscripted. For example, the part of the ML that resides on the MH will be referred to as ML_{MH}.

C. Software Architecture

Figure 3 shows the ALICE architecture in its entirety. The services offered by the architecture can be divided into three distinct areas of functionality, each of which is addressed by one of the architecture's layers.

Connectivity Management between the MH and the MG is handled by the Mobility Layer (ML). This layer hides the complexity of the MH's network characteristics from upper layers by performing transparent reconnections and tunneling. This is explained in section III.

Location Management of servers residing on mobile hosts is handled by the Swizzling Layer. This layer is used to support server mobility by translating ("swizzling") server references at various points in time and by redirecting clients with obsolete references towards more recent server locations. This layer is described in section IV.

Disconnected Operation of MHs is handled by the Disconnected Operation Layer. This layer lets clients cache server functionality during periods of disconnection. Support for conflict detection and resolution is also provided, although some issues are left to the application. This layer is described in section V.

The swizzling and disconnected operations layers are dependent on the protocol used between the client and server, and separate instances of these layers must be implemented for each protocol. The ML is protocol-independent and will work with any application-level protocol. We use the terms $S/Protocol$ and $D/Protocol$ to denote generic swizzling and disconnected operation functionality and substitute the name of an actual protocol when discussing implementations of the layers. The S/IIOP layer—the swizzling layer for the CORBA Internet Inter-Orb Protocol—is described in [5]. Work on D/IIOP—the disconnected operation layer for the same—is currently ongoing.

III. CONNECTIVITY MANAGEMENT

The MH has a set of physical communications interfaces that are used to connect to one or more different MGs at various points in time. (Figure 3 shows three interfaces whose transports are named TP1–TP3.) Typically, some interfaces will be wired whereas others will be wireless. The Mobility Layer (ML) manages these interfaces and hides the unreliability of wireless media from applications by performing transparent reconnections to MGs. In case the MH moves from one MG to another (for example, because it is no longer within range of the first) the two MGs negotiate to set up tunneling of any open connections there may exist between the MH and the RH. All connection state (such as unsent data) is transferred from the old to the new MG in a procedure called *handoff*. This happens transparently to applications running on the MG and the RH.

A. Mobility Layer API

The ML implements a superset of the BSD sockets API and can be used instead of the standard TCP-layer. This makes it simple to add mobility support (in the form of reconnection and tunneling) to legacy applications. For other applications, however, it is relevant to know the current state of connectivity and be notified whenever that state changes. We call such applications *mobile-aware*. For such applications, the ALICE architecture offers mobility information in the form of callbacks. Whenever there is a change in connectivity state (such as the loss of a connection to a MG or the creation of a connection to a new MG) the application receives a callback in the form of an invocation of a previously registered function. Callback functions are registered via the extended sockets API. The use of callbacks is optional, and applications need not use it.

B. Mobile Servers

For mobile clients, the standard ML functionality is sufficient to operate without changes to the application. Outgoing connections are proxied at the MG and tunnelled between MGs in case the connections persist for longer than the MH is connected to the MG through which the connections were initially made.

For mobile servers, the situation is more complicated. When a mobile server starts listening on a socket, a port is dynamically allocated on the MG and the listen operation takes place on that socket. When a MH moves to a new MG, another server socket is allocated on the new MG to replace that on the old MG. Hence, a mobile server cannot itself choose the endpoint (host name, port number) its clients need to connect to. Furthermore, the endpoint will change with every MH movement. The ALICE architecture addresses this problem with the Swizzling Layer described in section IV.

C. Interface Management

The ML continuously gathers statistics about the latency, bandwidth and error rate of the MHs communications interfaces. This information is used to pick a new interface to use for reconnection whenever the connection to a MG breaks. The statistics are accessible to mobile-aware applications via

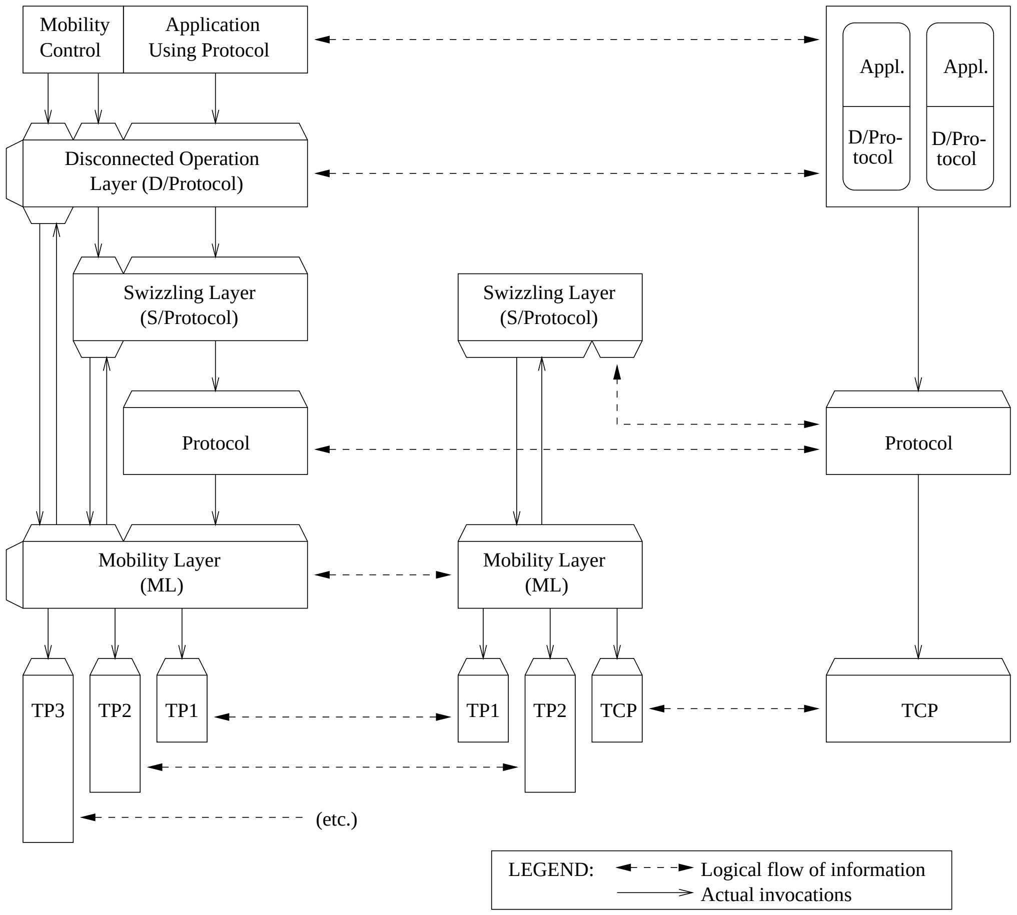
Mobility Control
Application Using Protocol
Disconnected Operation Layer (D/Protocol)
Swizzling Layer (S/Protocol)
Protocol
Mobility Layer (ML)
TP3
TP2
TP1
TCP
Appl.
D/Protocol
(etc.)
LEGEND:
Logical flow of information
Actual invocations

Fig. 3. The Abstract ALICE Architecture

the ML's tuning API. This API also allows a mobile-aware application (such as a ML configuration tool) to implement its own communications interface selection algorithm and configure the ML to use it. The default algorithm is based on a simple round-robin type scheme.

D. Operation During Disconnection

In the ALICE mobility model, it is normal for a MH not to be connected to a MG at all times. In this case, the ALICE architecture's default mode of operation is to synchronously queue unsent data between the MH and the MGs. Asynchronous queueing is not explicitly supported by the architecture but can be achieved by using application-level multi-threading in conjunction with the synchronous queueing mechanism.

IV. LOCATION MANAGEMENT

Although the ML performs handoff and tunnelling of connections, a mobile server will change its connection endpoint when its MH moves to a new MG. This may confuse clients that hold references to old endpoints, especially because ALICE does not assume these clients are aware that the server is mobile. The swizzling layer addresses this problem by performing swizzling (runtime translation) of server references and by redirection of clients towards more recent server locations. The former is done on the MH and the latter on the MG.

A. Swizzling

One function of the $S/Protocol$ layer is to manage server references held on the MH. The $S/Protocol_{MH}$ layer uses the callback mechanism of the ML to keep track of the current MG and uses this information to translate the server references every time the MH moves to a new MG. No attempt is made to update old references held by clients until they are used. The $S/Protocol_{MH}$ layer relies on requirements 3 and 4 from section I-B to encode information about endpoints in server references.

Swizzling a server reference occurs when a new reference is created and the MH is connected to a MG. In this case, each endpoint referring to a local interface is removed from the reference, saved for later unswizzling and replaced by an end-

point for $S/Protocol_{MG}$. The $S/Protocol_{MG}$ layer listens on a well-known port which allows swizzling to take place on the MH without involvement of the MG. When the mobile server starts listening on an endpoint, a logical connection between the MH and the $S/Protocol_{MG}$ layer is set up. This allows connection attempts that arrive at the $S/Protocol_{MG}$ layer to be forwarded to the server application on the MH.
Reswizzling a server reference occurs when a MH moves from one MG to another. In this case, the $S/Protocol_{MH}$ layer receives a callback from the ML that the MG address has changed. The $S/Protocol_{MH}$ layer reswizzles a server reference by replacing all endpoints referring to the $S/Protocol_{MG}$ layer with endpoints referring to the $S/Protocol_{MG}$ layer on the new MG.
Unswizzling a server reference occurs if the mobility support layers are removed from the protocol stack, for example in case the MH gets a direct connection to a LAN. In this case, all server references known to the $S/Protocol$ layer are unswizzled. For each server reference, the endpoints referring to the $S/Protocol_{MG}$ layer are replaced by the local endpoints that were saved during swizzling. Any endpoints referring to remote interfaces are unchanged.

B. Redirection

A client may hold a swizzled reference identifying a mobile server. The swizzled reference will contain one or more endpoints identifying the $S/Protocol_{MG}$ layer rather than the server on the MH itself. A client attempting to invoke a server with a swizzled reference will go through the following steps.
1. Attempt to connect to the first endpoint specified in the reference. Because the reference is swizzled, this endpoint belongs to the $S/Protocol_{MG}$ layer on the mobility gateway to which the MH was connected when the reference was published. If the MH is still connected to this MG, the connection attempt will succeed.
2. If the MH has moved and reconnected to a new MG (after a handoff between MGs), the $S/Protocol_{MG}$ layer will redirect the client to the $S/Protocol_{MG}$ layer on the new MG. The old MG will know about the new MG because it has been involved in a handoff.
3. If the MH has not reconnected and no handoff has taken place, the $S/Protocol_{MG}$ layer cannot redirect the client. In this case, it passes the incoming connection to the underlying ML which in turn waits for the MG to reconnect to this or another MG. The client is blocked until this happens.[1]

V. Disconnected Operation

When a MH is disconnected from MGs for a longer period of time, the default action of the ML (to queue invocations in both directions) may be infeasible. The disconnected operation layer $D/Protocol$ allows clients to replicate server objects and cache the replicas locally. While the client and server are disconnected, client requests to the server are redirected to the replica stored on the client side. This allows the client to operate in spite of not having access to the server. In figure 3, the client is shown as residing on the mobile host and the server on the fixed host because we believe this to be the typical case. The server could also reside on the mobile host in which case the remote client would cache server functionality from the mobile server. This is not shown in the figure.

A. Client Side

The $D/Protocol$ layer on the client side has two important functions: to *maintain a cache* of server objects and to *redirect invocations* to these cached objects. These two functions are independent and can be used separately by clients. For example, a simple client may not want to control which server objects are cached but would still like to invoke a cached copy if one happens to be available. A more complex client could be in explicit control of which server objects it wishes to cache. A third type of client could be a user-interactive 'hoarding-tool' which would interact with the cache even though it would never itself need to invoke any of the cached server objects.

B. Server Side

The replication of server objects is generally difficult to do without application involvement. For this reason, the server-side of the $D/Protocol$ layer takes the form of a replication support module rather than a separate layer in the protocol stack. In figure 3 this is shown by integrating the application layer with the disconnected operation layer. The two parts of the $D/Protocol$ layer perform vastly different functions. Whereas the client side performs cache management, the server side handles requests for replication and reconciliation of server objects and also provides the server application with support functions for performing conflict detection and resolution.

VI. Related Work

Related work in the area of mobile CORBA support includes the Dolmen Project [6], the JumpingBeans system [3] and the OnTheMove project [4]. More detailed discussion on these can be found in [5]. The Object Management Group (OMG) are also working on extending CORBA to deal better with mobile environments. The last Request for Proposal (RFP) [7] expired in May 2000 and resulted in two responses [2], [1].

[1] Accepting connections while the mobile host is disconnected is a design decision which may change in the future. Another approach could be to refuse new connections.

VII. Conclusion

This paper has presented the design of the second version of the ALICE framework. Since the first version, the architecture has evolved towards a more generic form in order to be applicable to other than CORBA environments. One feature retained from the original design is its modularity: the structuring of functionality into layers that each solves a portion of the problem space and can to a large extent be used independently[2] of the other layers when required.

Acknowledgements

The authors are very grateful to IONA Technologies plc for their generous support.

References

[1] Ke Jin (editor). Wireless Access and Terminal Mobility (telecom/00-05-05), 2000.

[2] Kimmo Raatikainen (editor). Wireless Access and Terminal Mobility (telecom/00-05-01), 2000.

[3] Ad Astra Engineering. Jumping Beans White Paper. http://www.jumpingbeans.com/, December 1998.

[4] Peter Kemp et. al. *Design of MASE V2* . http://www.sics.se/∼onthemove/docs/OTM_d33.doc, 1996.

[5] Mads Haahr, Raymond Cunningham, and Vinny Cahill. Supporting corba applications in a mobile environment. In *Proceedings of the 5th International Conference on Mobile Computing and Networking (MobiCom'99)*, pages 36–47. ACM, August 1999.

[6] P. Reynolds and R. Brangeon. Service Machine Development for an Open Long-term Mobile and Fixed Network Environment. Project deliverable, DOLMEN Consortium, December 1996.

[7] OMG Telecom TC. Wireless Access and Terminal Mobility RFP, May 1999.

[2] The disconnected operation and swizzling layers depend on the ML, but there are no other inter-layer dependencies.