A Model for CORBA Communications in Ad hoc Networks

Luiz Lima Jr. and Alcides Calsavara
Pontifical Catholic University of Paraná, Brazil
Post-Graduate Program on Applied Computing
{laplima,alcides}@ppgia.pucpr.br

Abstract. *The increasing popularity of wireless-enabled PDAs, laptops and smart mobile phones, in addition to a continuos explosion in the number of mobile services and networks, has made it indispensable the construction of frameworks to aid the development of distributed applications in ad hoc environments. Since CORBA is a mature and largely used architecture for distributed heterogeneous systems and because of OMG's recent move towards interoperation in wireless mobile environments, it seems appropriate to extend CORBA's interoperability model so that it can be used in ad hoc networks as well. This paper focuses on this issue raising architectural requirements for adapting CORBA's interoperability model to transparently deal with communication in ad hoc wireless networks. An architecture, based on federations of specialized name servers is proposed and a strategy for routing GIOP messages in such an architecture is defined. Finally, some implementation issues and performance considerations are discussed.*

1. Introduction

Wireless-enabled PDAs and smart mobile phones are becoming increasingly popular. As a result, a continuous explosion in the number of mobile services and networks has been recently observed. Wireless networks with static access points as well as ad hoc networks of PDAs, laptops and smart mobile phones are required in order to implement a whole set of new applications that range from collaborative work (during meetings and conferences, for instance) to extreme scenarios like disaster relief or battlefield environments. Other examples of applications are context-aware systems that update their behavior according to its operating environment [1] and sensor networks used in telemetry. The potential market of such mobile wireless applications is evidently raising the interest of business directors, technology managers, solution developers, service providers, software vendors and mobile devices vendors.

The development that has been achieved in the last years in the area of distributed systems in static networks makes it inconceivable not to take advantage of the well established solutions while adapting them to new contexts. One of the most mature platforms that gather a wide range of solutions is the Common Object Request Broker Architecture (CORBA) from the Object Management Group (OMG). Though OMG has recently adopted new standards regarding wireless access and terminal mobility [2], very few implementations are currently available [3]. This is due certainly to the complexity of wireless network protocols, interoperation details and also because the wireless technology field is still controversial, a battlefield with each vendor trying to enforce its own solution to the market. Besides that, the OMG's adopted standard only contemplates

CORBA communication between terminals and wired networks (the terminal may contain either server or client CORBA objects). There is no mention whatsoever to the possibility of the application of the concepts in wireless ad hoc networks. Therefore, we understand, a new architecture for supporting ad hoc CORBA network is necessary.

This paper looks into the problem of communication among CORBA objects located on ad hoc networks and proposes an architecture and implementation to deal with it. The model is built upon well-known research topics deliberately avoiding details about underlying wireless protocols or specific routing algorithms for ad hoc networks. It is organized as follows. Section 2 gives an overview of CORBA's interoperability model and terminal mobility specification in order to establish common grounds upon which to define the proposed solution. The general model is presented in Section 3 and detailed in Section 4 (federations of specialized Name Servers - ah-NSs). Section 5 shows how messages are routed in the networks using CORBA communication mechanisms and the implementation framework of the prototype developed together with some performance considerations are discussed in Section 6. In Section 7, some related works are commented and conclusions are drawn in Section 8.

2. Overview of Interoperability and Terminal Mobility in CORBA

A deeper look into some relevant features of CORBA is essential at this point in order to lay the foundations for the proposed ad hoc communication architecture. The following sections highlight some of these features.

2.1. CORBA

CORBA aims basically to provide *interoperability* among distributed objects (possibly located in heterogeneous environments) and additional useful distributed *services* (e.g. Naming Service, Trading and so on). This powerful combination of interoperability and services greatly aids the development of portable distributed applications by removing the developer's necessity of being concerned with low-level communication details.

A *CORBA object*, which is a "virtual" entity, is "incarnated" by a *servant* that implements the object's operations in a specific programming language. Servants exist within a *server application*. Clients issue *requests* to CORBA objects and receive *replies* from them through the ORB core that basically provides interoperability. In order to receive requests, CORBA objects must define their interfaces in a specific language called IDL (Interface Definition Language) so that all other components of the system become capable of understanding and, therefore, capable of using these interfaces. Additionally, CORBA objects that implement these IDL interfaces must somehow publish their references in order to be accessed by clients. These object references are analogous to C++ class instance pointers, except for the fact that they may denote distributed objects. These references:

- identify each one exactly one object;
- can be nil;
- can dangle (i.e. they may become invalid over time for they propagate in an uncontrolled way, from the perspective of the server);
- are opaque (i.e. they may carry standardized components as well as proprietary

information);
- are strongly typed and support late binding (of derived types - IDL supports inheritance);
- are interoperable (i.e. used by different ORB vendors - this is still a consequence of being opaque).

References are published and therefore acquired either from files (containing stringfied forms of it), or from well-known services (such as Naming Service or Trading Service), or from another object that may return it, or from some other way in which the reference can be sent (e.g. E-mail, web page, etc.).

When a client receives an object reference, it instantiates a proxy object (called *stub*) that is responsible for supplying the same interface as the target object. This way, the client invokes operations on the proxy's interface which, in its turn, sends a corresponding message to the remote servant. On the server's side, another proxy (called *skeleton*) receives the messages sent by the client and performs the calls to the actual servants. Replies follow the same (reversed) path. Messages among ORBs are coded using the General Inter-ORB Protocol (GIOP), generally mapped into the Internet Inter-ORB Protocol (IIOP). However, in order to support inter-ORB communication in wireless networks GIOP tunneling is required, so that GIOP messages are mapped into messages of the actual wireless protocol.

2.2. Interoperability

Though GIOP is the basic interoperability framework of CORBA, it is not a concrete protocol. It only describes how specific protocols can be created in order to meet the architecture requirements (IIOP is a concrete realization of GIOP, for instance). The GIOP specification makes the following assumptions about the underlying transport that is used to carry messages [4]:

- it is connection-oriented;
- connections are full-duplex and symmetric;
- it is reliable;
- it provides a byte-stream abstraction (i.e. neither receiver, nor sender have to be concerned about issues like message fragmentation, duplication, etc. - they both see the connection as a continuous stream of bytes);
- the transport indicates disorderly disconnection.

Any transport that do not meet these requirements needs an additional "adaptation" layer in order to emulate them. GIOP has basically eight message types, namely:

- `Request`;
- `Reply`;
- `CancelRequest`;
- `LocateRequest`;
- `LocateReply`;
- `CloseConnection`;
- `MessageError`;
- `Fragment`[1].

However, only `Request` (originated by clients) and `Reply` (originated by servers) messages are of interest here, since these two implement the basic RPC mechanism. The structure of a GIOP message is composed of a 12-byte GIOP message header (containing GIOP version, message type, etc.) and a variable-length GIOP message body. The GIOP message body contains itself a header and body which is specific to each message type. The main components of the header of the `Request` and `Reply` messages are shown in Figure 1 (in IDL). Note that both `Request` and `Reply` messages carry a variable-length (`sequence`) *service context* field that can be used to silently embed additional data to each request. Service contexts are mainly used to propagate information required by ORB services such as Transaction or Security.

```
module GIOP {
   typedef unsigned long ServiceId;
   struct ServiceContext {
         ServiceId         context_id;
         sequence<octet>   context_data;
   };
   typedef sequence <ServiceContext> ServiceContextList;

   struct RequestHeader {
         IOP::ServiceContextList service_context;
         unsigned long request_id;
         boolean response_expected;
         sequence<octet> object_key;
         string operation;
         // ...
   };
   enum ReplyStatusType { NO_EXCEPTION, USER_EXCEPTION, SYSTEM_EXCEPTION,
                          LOCATION_FORWARD };
   struct ReplyHeader {
         IOP::ServiceContextList service_context;
         unsigned long request_id;
         ReplyStatusType reply_status;
   };
};
```

Figure 1. IDL specification of GIOP Request and Reply message headers

IIOP is an “instantiation” of GIOP over TCP/IP. IIOP basically defines the encoding of the Interoperable Object References (IORs). In other words, it specifies how to represent TCP/IP addressing information inside an IOR, in such a way that a client will be able to establish a connection to the server to send a request. Therefore, the endpoint information contained in the IOR in this case is composed of the host name and listening port number of the server.

The structure of an IOR is represented in Figure 2. The `Repository ID` is a string that identifies the most derived IDL type. The `Endpoint Info` field contains the information required in order to establish a physical connection to the server implementing the object, and the Object Key uniquely identifies the target object (generally, it is formed by the composition of the name of the Portable Object Adapter (POA) and object identifier, but each CORBA implementation may use different information).

The important aspect here is that multiple profiles can be embedded into the IOR so that it may efficiently support more than one protocol and transport. Each profile (whose type is `TaggedProfile`) contains a `tag` field (`TAG_INTERNET_IOP` in the case of IIOP

1. Only in GIOP version 1.1 and 1.2.

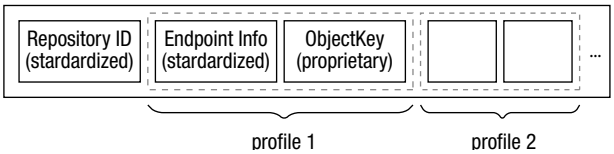

Figure 2. The structure of an Interoperable Object Reference.

1.1) and a `profile_data` member (version, host, port number, object key, etc. in case of IIOP). New user-defined profiles may be appended to an IOR without affecting its validity in systems that are not able to interpret and deal with these profiles. IORs are opaque.

2.3. Wireless Access and Terminal Mobility

Generally speaking, mobility may be supported at different levels [5]. The *link level* is an adequate level to treat the issue only when the mobility is inside one single link technology, location area and administrative domain, which is too restrictive in an heterogeneous world. Dealing with mobility at the *mobile IP* (IPv6) level is enough if micro-mobility is taken care of on the link level and mobility between administrative domains does not need any special arrangements. However, the *middleware level* is the best place for dealing with mobility between administrative or service provisioning domains.

OMG standards for wireless access and terminal mobility in CORBA [2] are based on the assumption that no modifications should be applied to a non-mobile ORB in order for it to interoperate with client or server objects running on a mobile terminal. This rationale led to the architecture shown in Figure 3.

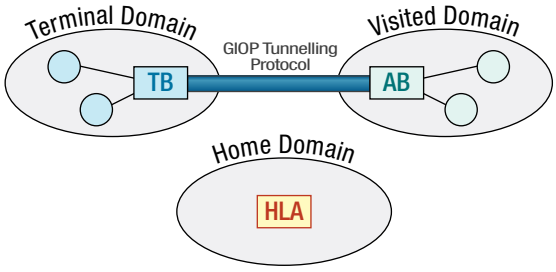

Figure 3. OMG's architecture for terminal mobility in CORBA

The *Home Domain* is the mobile terminal's administrative home (the organization that the terminal belongs to). The *Home Location Agent* (HLA) is responsible for tracking the location of each terminal belonging to the domain and for providing additional special CORBA services. *Homeless mobile terminals* have no Home Domain. The HLA provides operations (defined in IDL) for querying terminal locations.

The *Terminal Domain* comprises everything on the mobile terminal. The outside interface is the *Terminal Bridge* (TB). All CORBA invocations whose endpoint is on the mobile terminal go through the Terminal Bridge.

The *Visited Domain* contains the *Access Bridge* (AB), the counterpart of the Terminal Bridge. The Access Bridge is the passive side contacted by the Terminal Bridge. Both

HLA and Access Bridge interfaces are specified in IDL. These interfaces include operations to:

- query locations of mobile terminals;
- update locations as objects move around from one Access Bridge to another;
- perform hand-off (only the Access Bridge).

The specification includes initial reference operations in both the HLA and Access Bridge so that objects may register themselves with well-known *Name Servers* [2]. The references for servers located in terminals are called *Mobile IORs* (MIORs) which hide mobility from clients and even from the ORB that clients run on. Besides the default IIOP Profile (`TAG_INTERNET_IOP`), an MIOR contains a Mobile Terminal Profile (`TAG_MOBILE_TERMINAL`) defined in IDL in Figure 4. Instead of containing the host

```
module MobileTerminal {
   typedef sequence<octet> TerminalId;
   typedef sequence<octet> TerminalObjectKey;

   struct Version {
         octet major;
         octet minor;
   };

   struct ProfileBody {
         Version           mior_version;
         octet             reserved;
         TerminalId        terminal_id;
         TerminalObjectKey terminal_object_key;
         sequence<IOP::TaggedComponent> components;
   };

   struct HomeLocationInfo {
         HomeLocationAgent home_location_agent;
   };
};
```

Figure 4. IDL definition of the Mobile IOR

and port number of the actual object, the default IIOP Profile of an MIOR contain rather the endpoint (host and port number) of the terminal's HLA (or Access Bridge, in case of a homeless terminal). The Mobile Terminal Profile component contains information used by the HLA and Access Bridge to provide mobility transparency of server objects in mobile terminals.

One of the components in the Mobile Terminal Profile can be of type `HomeLocationInfo` which identifies the HLA of the terminal on which the MIOR was created. Section 5 will show that another specific profile will be needed to route ad hoc network requests.

When the HLA receives an invocation intended for objects located on known mobile terminals, it reads the Terminal Profile and sends a `LOCATION_FORWARD` response to redirect the invocation to the current Access Bridge.

Once the Access Bridge receives an invocation, it determines the target object and sends the invocation through the tunnel connecting itself to the Terminal Bridge. If the terminal is not any more connected to that Access Bridge, it can either respond with a

`LOCATION_FORWARD` exception (if it knows about the terminal's current location) or `OBJECT_NOT_EXIST` (which should cause the client to retry the invocation at the HLA).

Mobile Object Keys are used to identify objects in mobile terminals. This allows the HLA and Access Bridge to route invocations to the mobile terminal object in the same way it is done with Mobile Terminal Profiles.

Tunneling and Forwarding in Wireless Networks

GTP (*GIOP Tunneling Protocol*) is the communication protocol between the Terminal Bridge and the Access Bridge, and it has the same reliability requirements as the GIOP. GTP is designed to be mapped into concrete transport protocols (e.g. Bluetooth). The components of the GTP architecture are shown in Figure 5. The Access Bridge is responsible for converting GTP into IIOP and back into GTP.

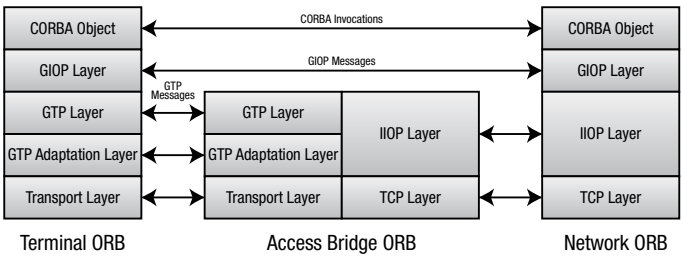

Figure 5. GTP architecture in wCORBA

GTP *Adaptation Layers* are required because generally the transport layers do not provide all the GIOP (and GTP) requirements such as reliability or orderly message delivery as mentioned above. This way, the Adaptation Layer is provided as the specification of a concrete tunneling protocol. Therefore, this layer needs only to define how the transport is to be used and the format of the transport endpoint.

GTP is composed of seventeen message types, most of which comprising Request-Reply pairs distributed in four main categories, namely: tunnel management, GIOP connection usage, GTP forwarding and specific purpose messages. There are two messages out of those that concern us the most: `GIOPData` and `GIOPForward`. `GIOPData` messages encapsulate GIOP messages (that will contain the routing information in ad hoc networks, as it will be seen in Section 3). `GIOPForward` messages are used to forward GIOP messages to another Access Bridge in case the target terminal is no longer connected to the Access Bridge specified.

OMG's specification [2] also defines in detail some handoff procedures divided into two different types, namely: *handoff* (when a terminal changes its access point) and *access recovery* (when a connection is re-established after a sudden loss). Since this topic is out of the scope of our consideration we will not get into it more deeply.

3. Ad hoc Model for CORBA Communications

Having briefly considered the main points regarding CORBA's interoperability framework and OMG's specification for wireless access and mobility, now we move on to the conceptual ideas behind ad hoc CORBA communications.

First, each terminal must contain a specialized naming server called ah-NS, so that local objects may publish their services and local clients may be able to search for suitable server objects. If a required service is not found locally, the search is propagated to a peer ah-NS, as it will be seen in detail in Section 4. Each ah-NS must be reachable at a well-known endpoint so that it can be easily found by other ah-NSs and a federation can be readily established. This network of ah-NSs has many similarities to a P2P network [6]. Inquiry operations are generally used to discover neighboring terminals. In order to protect individual ah-NSs from unauthorized service access or overloading, name service contexts and federation contracts can be used as it will be seen in Section 4. The general architecture is depicted in Figure 6, where the ah-TBs are the terminal bridges described in the following.

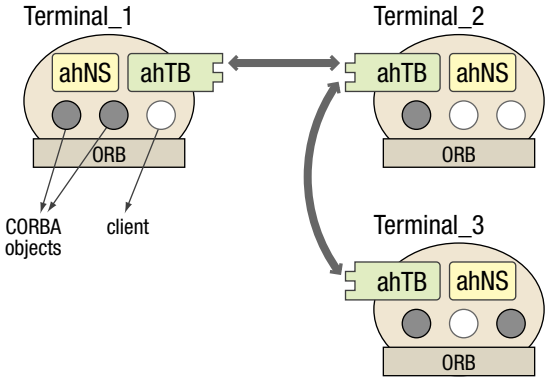

Figure 6. General architecture for ad hoc interoperability using CORBA

ah-NSs are of key importance in the architecture since they are responsible for the two main tasks in the distributed system: service publishing and finding (as mentioned above) and construction of Routing IORs (RIORs) that contain information about request and reply routing.

A traditional IOR contains all the information needed to establish an end-to-end communication channel between client and server (interface type and communication endpoint). However, in ad hoc networks routing information is also needed since a client may not be able to establish a direct connection to a server. Therefore, RIORs must carry additional information for this purpose. As seen in Section 2, the IOR may be composed of several customized tagged profiles that are ignored by ORBs that do not know how to deal with them. The idea is to add a new profile - the Routing Profile - which encapsulates information of each terminal in the request/reply path. This profile is gradually built by each ah-NS participating in a successful service resolution. The RIOR is an IOR embedding a Routing Profile (to be described in detail in Section 4).

Figure 7 shows how the Routing Profile is constructed by each ah-NS in the service path. Performing a distributed Breadth First Search (BFS) [7] in the ad hoc network graph (in order to minimize hop count), each ah-NS appends to the routing profile list the information needed so that the request would follow the same routing used in the successful service discovery. In this case performance is measured in terms of hop counting (though other metrics could be defined) and the nearer the service is, the faster it is found

and the faster requests are routed to it. That is why an efficient searching algorithm is needed, and, in this case, the distributed Breadth First Search was chosen.

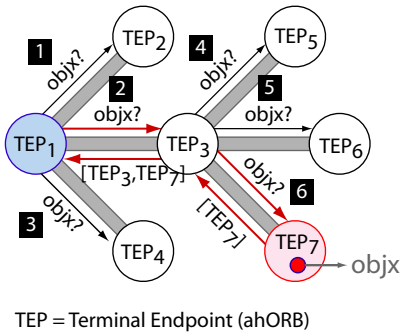

Figure 7. Building the Routing Profile while searching for `objx`

Upon receipt of the RIOR from the server object, clients are now able to issue requests and receive replies. The routing of each message is in fact supported by embedding the path information required into the message's context that is available in the standard GIOP specification (as seen in Section 2.2). The ad hoc Terminal Bridge (ah-TB in Figure 6) is an specialized Terminal Bridge responsible for including and extracting this routing information into and from the message's context, transparently forwarding messages that are not destined to it. Some changes in the ORBs implementation may be needed in order to carry other application-specific context information and to avoid any conflicts. The underlying GIOP adaptation layer can be the same used by wireless tunnelling (Section 2.3).

Of course, many details are involved in such a scenario, namely, the establishment of ah-NS federations, routing of GIOP messages and implementation and performance issues.

4. Federations of ah-Name Servers and the Construction of Routing Profiles

Ad hoc networks are typically formed by terminals that issue regular *inquiry* operations in order to detect reachable neighbors. Once a neighboring terminal is identified (and all the security policies are checked out) the communication channel (the GIOP tunnel) then is established. At each side of the channel lies an ah-TB responsible, among other things, to initiate the negotiation of ah-NS federation contracts leading eventually either to a federation establishment or to a disconnection. The establishment of ah-NS federations is mandatory in our model. They are essential to support interoperability between clients and remote servers. That is to say that one of the first steps in setting up the ad hoc middleware model is to bind ah-NSs together so they form something similar to a P2P network in which federations contracts determine the level and the rules of the interactions. For instance, only a subset of known services may be made available to a peer ah-NS and these policies can be determined in a peer basis. The network of ah-NSs is formed by inquiry operations and it is, therefore, equivalent to the physical ad hoc network (this is generally not true about wired P2P networks). The same protocols used in establishing the P2P networks (e.g. JXTA [6]) can be used to create ah-NS federations, provided that the differences above are considered.

Even before the establishment of an ah-NS federation, servers can publish their services locally making them, of course, available only to local clients. However, once the federation in established, any server can potentially be reached by any client (although some ah-NS may impose restrictions, as we have said).

The search algorithm follows a distributed BFS [7] pattern since the nearer the suitable service is found, the better the system overall performance and scalability is. Beginning locally, the search goes on the closest neighbors going up to the most distant terminals or till it finds a suitable reference (Figure 8).

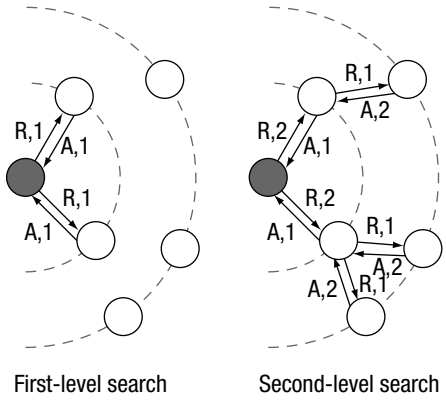

Figure 8. BFS search in the federation of ah-NSs. Ri = requests; Ai = answers

Since for some applications the number of hops may be critical, a new `resolve_bounded` operation was introduced to limit the search ray. If no suitable server is found within that ray, a `OBJECT_NOT_FOUND` exception is thrown. The well-known `resolve` operation is meant to find the service no matter how far it is. `Bind` and `rebind` operations concern the local ah-NS only, since there is no point in advertising remote services (i.e. services of other terminals) when a naming federation exists and when remote services may not be reachable. The main components of the IDL interface for an ah-NS are shown in Figure 9.

When a server is found, each ah-NS in the successful path to the server must add its terminal endpoint to the RIOR's Routing Profile. This way, the final RIOR returned to the client will also contain the routing information to reach the server object. This Routing Profile is transparent to ORBs that are not aware of the ad hoc network, just as it happens to Mobile Profiles (Section 2.3). The structure of the Routing Profile shown in Figure 10 follows the same pattern used by the Mobile IOR profile (Figure 4). A Routing Profile is an `IOP::TaggedProfile` with tag value of `TAG_AH_IOP` and profile data defined by the structure `AHTerminal::ProfileBody`. A Routing Profile may contain several `TAG_AH_TERMINAL_INFO` tagged components, each one identifying one terminal in the routing path to the server object. If the server is found locally or in the direct neighboring terminals, no `TAG_AH_TERMINAL_INFO` component is required. Otherwise, a new component is added to the Routing Profile by each ah-NS as described above. Figure 11 depicts the structure of an RIOR. T1 through T5 correspond to the identifiers of the ah-Terminal Bridges in the message routing path.

Though the construction of an RIOR is carried out by the federation of cooperating ah-NSs, the routing itself of GIOP messages (especially in the presence of mobility) requires an additional mechanism.

```
#include <CosNaming.idl>
module AHSystem {

   interface AHNS : CosNaming::NamingContext{
         // Exceptions
         exception RayReached {}; // indicates
                     //resolve ray has been reached
                     // and the service was not
                     // found
         // Federation operations
         void setupPeer (in AHNS peer);

         // NS operations
         void bind (in CosNaming::Name name, in Object obj) raises (...);
         void rebind (in CosNaming::Name name, in Object obj) raises(...);

         Object resolve_bounded (in CosNaming::Name name, in short ray)
               raises (RayReached, ...);

         Object resolve (in CosNaming::Name name) raises (...);

         void unbind (in CosNaming::Name name) raises (...);
         //...
   };
};
```

Figure 9. Main parts of the IDL specification for the ah-NS interface

```
module AHTerminal {

   struct Version {
         octet major;
         octet minor;
   };

   struct ProfileBody {
         Version          rior_version;
         octet            reserved;
         MobileTerminal::TerminalId   final_terminal_id;
         MobileTerminal::TerminalObjectKeyterminal_object_key;
         sequence<IOP::TaggedComponent> components;
   };

   struct AHTerminalInfo {
         MobileTerminal::TerminalId terminal_id;
         long distance; // distance to server in hops
   };
};
```

Figure 10. IDL definition of the Routing Profile

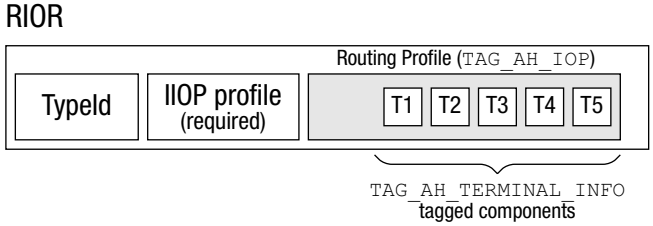

Figure 11. Structure of an RIOR

5. Routing of GIOP Messages

Once a client receives an RIOR, all the information needed to route the request to the target object is at hand. However, differently from the wired static network, routing is not automatically performed by the underlying protocols. Therefore, routing information must be embedded into the normal request so that the message is forwarded from terminal to terminal till it reaches the appropriate target. Embedding the routing information is straightforward since request/reply GIOP messages may carry *context information*. This context is a structure composed of an `id` and `data` of type `sequence <octect>` (see Figure 1). This means that a context may contain data of any type, and particularly, our routing information. Nodes that do not know the meaning of the routing context simply pass it on (normally to the target object it contains), while ah-TB-enabled terminals extract information from it in order to forward the request to another peer terminal.

An important aspect to point out here is what happens when the message path changes due to terminal mobility. In this case, if a peer terminal to which the message should be forwarded is not anymore reachable, then the current ah-TB in the path must issue another service request lookup though its local ah-NS interface in order to establish a new (alternative) route to the server or back to the client (if the message is a reply, the search should be for the client's terminal, i.e. the client's ah-TB). Doing so, the original routing context path can be updated. The structure of a routing context is defined in Figure 12. If the server (or client) is not reachable anymore, a queue is created to hold the message and possibly other messages that may arrive (in the case the server is unreachable) and a monitoring thread is started to perform regular tries to find the message target until a time out is reached. In this case, the terminal is supposed to be permanently unreachable and an `OBJECT_NOT_FOUND` exception is thrown in the client (server messages are silently discarded). Each ah-TB in the message path is also equipped with time-out mechanisms so that long waited replies become `OBJECT_NOT_FOUND` exceptions as well. The definition of these time out values clearly has a great impact on the system overall performance.

```
struct RoutingContext {
   TerminalId originating_terminal; // client's terminal
   sequence<TerminalId> path;// terminal ids for the msg path till target
   long current_terminal;  // # of the current terminal (dealing with msg)
};
```

Figure 12. Routing Context structure

When it comes to implementation, CORBA fortunately provides standardized meta-programming mechanisms that support inclusion of context information in request/reply messages in a way transparent to clients and servers [8][9]. These mechanisms in addition to CORBA's message forwarding techniques used in the prototype implemented are described in the following.

6. Implementation Issues and Meta-Programing Mechanisms

Mobile applications in ad hoc networks require two fundamental mechanisms in order to deal with the problems that arise in such contexts. Namely, message forwarding mechanisms and transparent (from the point of view of the programmer) attachment of additional information to the original message.

CORBA originally provides an efficient mechanism for forwarding messages through a GIOP reply exception called `LOCATION_FORWARD` (see Figure 1). When a server receives a request that it cannot, for some reason, deal with, it can send back to the caller a `LOCATION_FORWARD` exception with a new object reference which the request should be sent to. Upon receipt of a `LOCATION_FORWARD` exception, client ORBs transparently retry invoking the operation connecting to the new location. This mechanism is widely used in CORBA, especially in indirect binding through the Implementation Repository. `LOCATION_FORWARD` exceptions may be thrown by *interceptors* or by *servant managers* registered within the POA. Servant managers were used in the prototype implementation due to its simplicity and because much information about the request that is needed is readily available at that level (which is not always the case with portable interceptors).

On the other hand, meta-programming improves adaptability of distributed applications allowing behavior changes with a minimum (if any) impact on the existing software. Therefore, they can be used in the context of ah hoc communications to deal with routing and re-routing requirements. In our case, we used standardized portable interceptors that are called at specific points in the operation invocation chain both on the client and on the server sides. On the client side, the `send_request` interception point adds RIOR routing information into the context of the requests. On the server side, we used the `receive_request_service_contexts` interception point to extract the context information and determine if the request is to be forwarded to the next terminal or processed locally. More details on using interceptors and interception points can be found in [8].

6.1. Prototype framework and performance considerations

Using servant managers for request forwarding and portable interceptors for adding the information needed, a prototype framework for mobility simulation using the ad hoc model proposed was implemented. This framework is depicted in Figure 13.

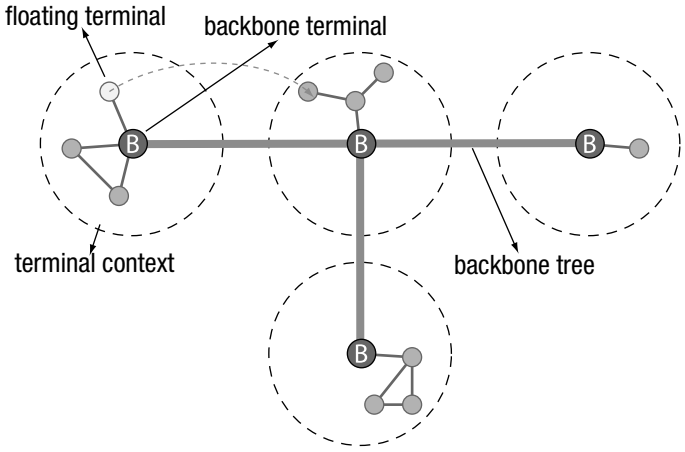

Figure 13. Prototype framework

In order to preserve connectivity, the prototype implemented assumes the presence of a network of *backbone terminals* (the *backbone tree*). Dealing with messages addressed to disconnected terminals (with asynchronous communication mechanisms, for instance) is out of the scope of this paper. Each backbone terminal define a *terminal context*. *Floating terminals* can move across terminal contexts and they are all connected

(directly or indirectly through a peer connection) to a backbone terminal. Our simulation involved communication between randomly moving floating terminals going from one terminal context to another (typically, terminals only move to neighboring terminal contexts).

Using the resolve operation, clients obtain appropriate server RIORs from their local ah-NSs. With that RIOR, they send messages (operation calls) to the server through all intermediate ah-TBs in the routing path. If a server moves from one terminal context to another, a new routing profile is computed (using again the resolve operation) and a `LOCATION_FORWARD` exception is sent back to client with the newly computed RIOR. When a client moves while waiting for a reply, the only way it can be found is by broadcasting its new location.

The implementation showed good scalability due to the intrinsic distribution of the model and the use of low level message interception points and factory pattern design. A time interval is defined for each floating terminal, at the end of which it moves to another terminal context. Of course, the higher the mobility of a floating terminal is, the lower the system performance is. This performance degradation occurs because of the need of re-locating the service and updating the routing profile by intermediate terminals, in addition to the normal message routing.

The use of a distributed BFS algorithm in the ah-NS federation also has the effect of improving significantly the system performance, since closest terminals require less routing effort.

7. Related Work

This section describes some current works in associated areas and discusses how they relate to the research proposed.

In [10], the authors point out that most of the ad hoc routing algorithms are complex demanding substantial resources (memory, CPU time, network bandwidth, etc.) just to maintain the routing information. They then suggest that, for most of collaborative systems, simply broadcasting messages to peer terminals is enough to do routing. However, despite of its simplicity and low cost, the strategy is certainly not scalable. The service searching phase is not avoided by this mechanism and maintaining routing information is only too much costly if the system is extremely dynamic. In such dynamic systems, the solution proposed can be easily adapted to do broadcasting. Just the ah-TB interceptors need to be changed.

Scalability is one of the main problems faced by ad hoc middleware solutions. Paper [9] introduces the paradigm of group management for ah hoc networks that allows better management of failures by assembling mobile nodes in groups that meet functional and non-functional requirements. Group initialization, member discovery and message broadcasting strategies defined can be used in the scope of service discovery in the federation of ah-NSs and GIOP routing. Scalability can be improved this way.

Typically object-based distributed systems assume that the connection between client and server object is long-lasting. Therefore, the possibility of disconnection is not addressed. The SOMA (Secure and Open Mobile Objects) project [12] and ORB/OS Task Force [13] strategy is to combine the paradigm of mobile agents with distributed

object systems using CORBA to address scalability and interoperability. However, in order to locate service objects, a centralized server is needed. Some extensions allow objects to maintain references when clients move, but this is not generalizable to a mobile ad hoc environment.

Paper [6] describes a middleware called “Expeerience” built on top of JXTA aiming at improving functionality of P2P wired networks (that are generally static) so that they can cope with intermittent connections thus increasing the potential of resource discovery in wireless mobile ad hoc networks. The middleware is heavily based on JXTA technology and therefore tries to overcome many JXTA specific limitations regarding mobility, some of those not present in the CORBA architecture (e.g. runtime message forwarding). Moreover, terminals are seen as peers (as they are), but there seem to be no distinction between the terminals and their internal components (that may be either clients or servers or even peers, as ah-NSs). Besides this, in Section 4, it was mentioned that ah-NS federations form a P2P network with specific contractual regulations to restrict access to undue services. It is not clear to us, however, whether Expeerience can deal with such contractual restrictions or even if this is possible.

8. Conclusions and Future Work

The increasing popularity of wireless-enabled devices and emergence of new applications demands make the topic of middleware for ad hoc networks more relevant. However, the difficulties are many and range from heterogeneity and interoperability to mobility.

This paper suggested extensions in CORBA, a well-known and widely used distributed platform, in order to deal with communication in an ad hoc environment. Service discovery and object reference construction are coped by federations of specialized name servers called ah-NS. To do so, a new reference format - the RIOR - was defined. Once a client receives an RIOR, intermediate terminals need to deal with GIOP requests and replies so that they are successfully routed to the target objects. This is carried out by embedding routing information into the messages’ contexts using portable interceptors. Our experiments showed that the model is viable and presents good scalability.

Evidently, many details have been left behind in our consideration and they are subject of our current work. They involve: communication across networks (where bridges are needed); the inclusion of ah-Traders to the model (similarly to ah-NSs); deeper investigation of performance aspects and solutions in highly dynamic environments; use of mobility event notifications [2] to improve the routing algorithm.

Also, no considerations have been made concerning the terminal’s limited resources. Federations of Implementation and Interface Repositories are not considered either - only static binding is taken into account.

References

[1] Musolesi. M. (2004) “Designing a context-aware middleware for asynchronous communication in mobile ad hoc environments”, Proceedings of the 1st international doctoral symposium on Middleware.

[2] OMG (2003), “Wireless Access and Terminal Mobility in CORBA”, Version 1.0, formal/03-03-64.

[3] wCORBA Research Project (2005) “MIWCO Wireless CORBA Support in MICO”, University of Helsinki - http://www.cs.helsinki.fi/u/jkangash/miwco.

[4] Henning, M.; Vinoski, S. (1999) “Advanced CORBA Programming with C++”, Addison-Wesley.

[5] Raatikainen, K. (2001) “Wireless Access and Terminal Mobility in CORBA”, Whitepaper, Dep. Computer Science, Univ. of Helsinki.

[6] Bisignano, M. et al. (2003) “Expeerience: a Jxta middleware for mobile ad-hoc networks”, Proceedings of the 3rd International Conference on Peer-to-Peer Computing.

[7] Lynch, N. (1996) “Distributed Algorithms”, Morgan Kaufmann Publishers Inc.

[8] Wang, N. et al. (2001) “Evaluating Meta-Programming Mechanisms for ORB Middleware”, IEEE Communications Magazine, Volume 39, Number 10.

[9] Baldoni, R., C. Marchetti, and L. Verde (2003), “CORBA request portable interceptors: analysis and applications”. Concurrency and Computation Practice & Experience, 15(6): p. 551-579.

[10] Hans-Peter Bischof, Alan Kaminsky, and Joseph Binder (2003) “A new framework for building secure collaborative systems in ad hoc network” 2nd International Conference on AD-HOC Networks and Wireless (ADHOC-NOW '03), Montreal, Canada.

[11] Liu, J., Saihan, F. (2005) “Group Management for Mobile Ad Hoc Networks: Design, Implementation and Experiment”, Proc. of International Conference on Mobile Data Management.

[12] Universitá di Bologna (2005), “SOMA”, http://www-lia.deis.unibo.it/Software/SOMA.

[13] OMG, (2005) “ORB and Object Services Task Force”, http://orbos.omg.org.