Object Oriented Multi-Layer Router with Application on Wireless Sensor-Actuator Networks

D. Villa*, F. J. Villanueva*, F. Moya*, G. Urzaiz†, F. Rincón* and J. C. López*
*Dept. of Technology and Information Systems
University of Castilla-La Mancha
School of Computer Science. 13071 - Ciudad Real. Spain
† Universidad del Mayab, Facultad de Ingeniería
Carretera Mérida-Progreso Km.15.5 A.P. 96, 97310, Mérida, Yucatán, México

***Abstract*—This paper introduces a novel approach to transport messages between a backbone network and a device network (SAN) in a seamless way, even when these networks use incompatible protocols or technologies. The paper shows how to leverage a general purpose distributed object oriented middleware to encapsulate and send messages as object invocations.**

Devices and routers are modeled as standard distributed objects. Furthermore, two different facets are available: the programmer may choose whether this mechanism is transparent to the client application using a socket-like API or by contrast she may directly invoke methods on a remote object. An early prototype will be discussed to illustrate this approach.[1]

I. Introduction

The wide variety of current protocols and technologies makes more difficult to develop interoperability mechanisms for applications running in nodes connected to different networks. Sometimes it is not even possible or advisable to rely on a single network protocol. Accessing nodes in a wireless sensor network constitutes a good example of this problem.

In short, there are two main approaches to solve the above problem by means of intermediate bridges or gateways. First, the special purpose sensor network may use the same protocol stack as the main network (typically TCP/IP). This approach has the following drawbacks:

- Nodes require a considerable amount of memory and computing power to be able to implement a TCP/IP stack. Even with reduced implementations such as uIP [1].
- Achieving an acceptable performance using TCP/IP in WSAN usually requires special adaptations such as header compression [2], modified packet encapsulations, addressing, etc.
- A TCP/IP stack may not be suited to the specific needs of sensor and actuator networks (SAN). For example, wireless links would usually add redundant information to recover lost datagrams due to frequent errors in such a noisy environment. By contrast, TCP reacts to transmission errors reducing the source data rate assuming there was a congestion problem [3].

An alternative approach would be to allow sensor nodes to use a specific protocol better suited to their characteristics and capabilities. Gateways or *base stations* hold a cluster of proxies representing each sensor or actuator and clients invoke services on the gateway using standard protocols such as TCP/IP. The software running in the gateway forwards the invocations to the right sensor or actuator using the protocol specific to that sensor network (see figure 1).
This alternative also has some drawbacks:

- Explicit gateways. Usually the programmer must be aware of the intermediate gateway. In order to request a service from a given sensor we need to know the address of the intermediate gateway instead of the address of sensor node.
- Lack of autonomy. Nodes in the sensor network may not directly interact with external elements. It is not obvious how they can offer services or demand services from nodes in a different SAN.
- Multi-gateway. The gateway becomes a single point of failure. Using multiple gateways for a single SAN is not immediate since it may lead to ambiguity in sensor/actuator access semantics or loops in message routes.

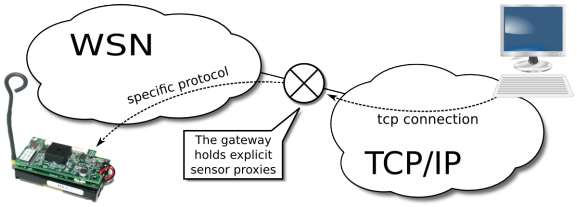

Fig. 1. Sensor network with an explicit gateway.

A. Inter-Networks

Internet is based on a single network protocol (IP). This global addressing scheme allows the interconnection of networks based on different technologies for the link and physical layers. Therefore in order to become a part of an inter-net it must satisfy certain constraints:

- Hosts must understand the logical addressing schema.
- Hosts must be able to build and frame datagrams using the underlying network technology.

[1]This research is partly supported by the Ministry of Education and Science (under grant CENIT Hesperia and TIN2005-08719) and by FEDER and the Regional Government of Castilla-La Mancha (under grants PAI08-0234-8083)

- When there is a diffusion medium, it must allow mapping logical addresses to physical addresses.
- A gateway or router is required to forward packets between the specific purpose network and the core network.

Traversing a network with an incompatible network protocol is usually implemented through tunneling. Unfortunately this approach disallows homogeneous routing schemes. Another obvious problem is how to achieve data transmission between devices using incompatible network and logical addresses.

B. Seamless Routing

Our goal is to provide an end-to-end data transport service, which is independent of any network or protocol technologies. It uses its own addressing and routing mechanisms even when the lower layers may have their own. The combination of the middleware and the multi-layer end-to-end routing provide many and valuable advantages:

Addressing. IDM has a true global and technology independent addressing scheme.

Virtual Network. It makes possible the deployment of virtual networks over any type of existing or future network.

Quality of Service. It provides facilities to apply high level QoS policies, since it may access richer information from the upper layers.

Flexibility. It may better suit the application requirements. Using a communications middleware does not necessarily involve protocol overhead or a significant increase of complexity. For example, it is possible to encapsulate application data within Ethernet frames, whenever the QoS constraints can be satisfied.

Platform Services. Since we use a general-purpose communications middleware, we may use many common services it provides. In the case of our current prototype (built on top of Ice), we will highlight a few: automatic persistency, remote deployment, implicit activation, load balancing, transparent service replication, etc.

Location transparency. Our routers are logic entities and thus they may be migrated to other nodes transparently.

Functional routing. This feature allows routing algorithms and schemes based on nodes grouped by function instead of geographic location.

Efficiency. Our routers may be implemented in embedded systems. They are suitable for embedding into a range of targets, from small microcontrollers to low-end FPGA. This is possible thanks to the picoObject approach [4].

Object Oriented. If the programmer wishes, it is possible to see the server side as a remote object. For stream oriented services or when the invocation semantics is unspecified, it is possible to use an alternative socket-like API.

C. About prototypes

In section VII, several initial prototypes are described in detail. They are implemented using Ice (Internet Communications Engine), an object oriented communications middleware by ZeroC, Inc. Throughout this article we will refer to some of its features and details because we consider them important to get a better understanding of the concepts explained. The ideas presented in this paper are not specific to Ice. Any middleware based on heterogeneous distributed objects may also be used.

II. Related work

Multiprotocol Label Switching (MPLS, RFC3031) or Layer Two Tunneling Protocol (L2TPv3, RFC3931) are similar to our proposal in intent, although they differ widely in the approach. They allow seamless end-to-end communications with any number of intermediate heterogeneous networks, but they rely on IP as the lowest level common protocol. This is less than ideal in sensor networks.

CORBA3 Messaging [5] implements a message routing strategy as an implementation detail of scalable asynchronous interaction. Routing information is statically embedded into the client-side proxies and there are no provisions for general purpose dynamic routing. On the other hand the CORBA interoperability architecture [6] defines a CORBA bridge to communicate different CORBA domains transparently. Unfortunately CORBA bridging is also static and it does not support routing or transparent migration of objects across domains. Besides, the overhead required to implement CORBA message routing makes it impractical for sensor and actuator networks.

III. Addressing

Traditional network addressing schemes use a logical address to identify the host, delegating destination process identification (server side) to the transport layer protocols. In TCP and UDP the destination processes are identified by *port* numbers. IDM uses a hierarchical addressing scheme and destination processes are identified with suffixes. These addresses are actually object identifiers. If global addressing is required, it may use any identification system that guarantees uniqueness. These are called GUIDs (Globally Unique IDentifiers). It is possible but not necessary to include in the address an identification of the host holding the object.

Therefore, clients only need to know the object identifier. To successfully deliver data an *endpoint* is required. The endpoint encapsulates the data transport for some specific technology: TCP, UDP, IP, Ethernet, XBow, Bluetooth, etc. The endpoint selection is automatic given the set of QoS constraints specified by the client and the set of currently available interfaces in the node. If these constraints are not satisfied then the invocation may fail.

IDM provides mechanisms to find the address associated to any endpoint of an object from its object identity. There is no need to know the object location in advance. Indeed it is used to find the endpoint of the adapter the object is registered in. An adapter works as an object demultiplexer. This allows a single object to be backed by several servants in different hosts, even on different physical networks.

A. Object Identity

The object identity is a globally unique byte sequence composed by two fields:

- **Category** Identifies the *application*, i.e. the common goal for all the objects connected to the same virtual network. Valid examples could be a P2P network, a set of streaming service subscribers, etc. It should be understood as a filter and thus, it is not mandatory.
- **Name** Identifies the object independently of the application. An object will have the same name in any application it is involved in. However, a single object implementation (servant) may have several names.

As mentioned above, the *name* field should conform to a hierarchical naming scheme, setting up the basis of the routing table functionality. One of the distinguishing features is that the address hierarchy may be based on functional criteria instead of geographic criteria (see figure 2). That is a significant contribution to overall flexibility allowing transparent node and router migration. In any case, in order to maintain the inter-network cohesion, at least a router is required in the border between any pair of networks.

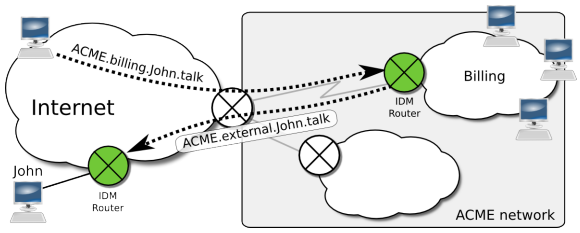

Fig. 2. Functional routing sample.

IV. Endpoints

An *endpoint* is a logical stand-alone structure that basically encapsulates data transport details and distributes them to the underlying protocol or technology. Furthermore, in regard to IDM, differences among endpoints lie in the provided features and not in the way they are managed. Therefore, there is no difference between an endpoint handling a TCP connection over VPN or a different one that sends SLIP frames over a RS-232 line.

The endpoint is in charge of the following tasks:

- Builds the IDM frames, marshals the user data, and generates the checksum when required.
- Hides the protocol or network interface details.
- Enumerates and checks the available capabilities of the transport protocol.

Off-the-shelf Ice provides only three endpoint types: TCP, UDP and SSL, although we have implemented a few more, such as UNIX sockets, message queues or Crossbow. Figure 3 depicts how the different endpoints relate IDM with the traditional network layer model. It also emphasizes the fact that the endpoint selection does not depend on the underlying protocol or network layer but on the provided features.

Each endpoint has a textual representation of the underlying protocol, the required hardware and software addresses, and additional information such as timeouts, coding certificates, etc.

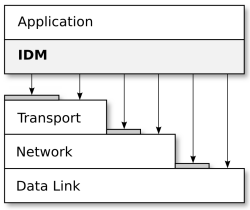

Fig. 3. Relations between IDM and traditional network layers.

IDM has a special type of endpoint to prepend the IDM header to the IceP[2] conventional messages. This fact simplifies the use of IDM with the standard Ice distribution by avoiding any modifications in the original code.

The aforementioned header fields are the following:

- **Magic number** "IDM".
- **Destination OID** The remote object identity.
- **Hop Limit** Maximum number of hops. Similar to TTL on IP.
- **QoS Constraints** A list of transport QoS preferences requested by the client.

A. Quality of service management

The required quality of service is specified when requesting access to the remote object. Some of the requirements such as encryption or authentication may be provided as end-to-end services. But some others such as the required bandwidth must be guaranteed along the whole route. The following parameters were taken into consideration although prototypes are currently missing some of them:

Oneway/twoway invocation, Delivery acknowledge/guarantee, Message ordering and multicast message ordering, Authentication, Authorization, Maximum latency, Required data rate, Source routing, etc.

Client must be able to specify what to do when some of the requirements cannot be satisfied. A strict policy may lead to the impossibility to provide the requested service.

V. Message Delivery

Simplest message delivery just involves the client and the object (server); usually this is designated as *direct delivery*. The client node requires the endpoints of the remote object adapter to perform message delivery, then the runtime must match a compatible endpoint available in the client side.

An **IDM network** is defined as a set of directly reachable objects, via some endpoint. There are no relations between the IDM network and the underlying physical networks the client is connected to. It may happen that an object adapter holds several endpoints despite being connected to a single physical network. Usually, IDM routers are placed in the border nodes, the nodes with interfaces in two or more incompatible networks either because of technology or due to protocol reasons (see figure 4).

[2]IceP is the protocol used to code Ice object method invocations. It's similar in many aspects to the CORBA GIOP protocol.

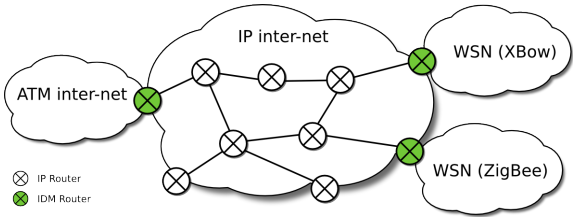

Fig. 4. An IDM inter-network made up of 4 different IDM networks.

In order to know the object adapter endpoints, the clients and routers must use the ALP (*Adapter Location Protocol*). The next listing shows the interface specification for this protocol, using the SLICE[3] language:

```
module IDM {
  module ALP {

    exception ObjectNotFoundException {};

    interface Locator {
      void add(Object* prx);
      void remove(Ice::Identity id);

      Object* resolve(Ice::Identity oid)
        throws ObjectNotFoundException;
    };

    interface Lookup {
      idempotent void request(Ice::Identity oid,
                              Lookup* callback);
      idempotent void reply(Object* prx);
    };
  };
};
```

Each node has a local service, the *ALP Service* that implements the *ALP::Locator* interface. Using the `lookup()` method, the client node gets a remote object proxy[4] which lets it invoke remote methods.

There are several ways to implement the ALP service. In our initial prototypes, we considered two alternatives:

- A distributed service that allows any host to ask for the existence of a specific object. This service uses the multicast or broadcast mechanisms provided by the available endpoints. Each node must listen to `request()` messages and it must answer by means of `reply()` messages. These messages are part of the interface *ALP::Lookup* and actually that interface is implemented as an *ALP Service facet* (see figure 5).
- The Ice Locator, which implements *indirect binding* in Ice may be a centralized (although it may be replicated) alternative to the ALP service. IDM has a wrapper that allows using the *IceGrid::Locator* service with minimal overhead. This is useful in big virtual network domains and when the hosts have enough resources to run as an IceGrid node. However it has a drawback: hosts must know in advance the endpoints of any *Locator* replica and at least one of them must be compatible.

In both cases, hosts serving objects must register them to make them available, either in the `Locator` or in the local ALP Service. Both may co-exists at the same time in the same network.

[3]Specification Language for Ice.

[4]A proxy is a local intermediary which acts on behalf the remote object.

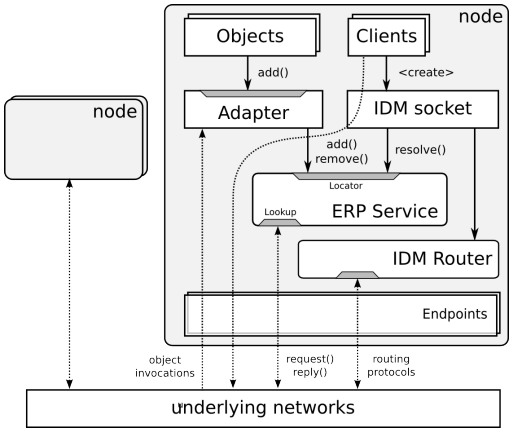

Fig. 5. IDM block diagram.

The *ALP* service has a cache that temporally stores resolved endpoints. The goal of that cache is to minimize the performance penalty imposed by the ALP messages. When the ALP is centralized, the IDM runtime incorporates a local cache due to the same reason.

Indirect delivery is used for those objects that are not reachable with any of the client endpoints. In these cases, at least a router is needed: Note it may happen a particular situation, the client QoS requirements may prevent endpoint usage even if the target is reachable.

VI. Routing

Hosts belonging an IDM network require a routing table that determines when is the direct delivery possible and when an IDM router is needed and which one. A router is also a standard distributed object. The main difference with respect to the objects is they may accept and forward messages whose destination is another object.

The next listing shows a routing table sample. It uses simplified object identifiers due to readability reasons.

```
Router.Row.NetA.Subnet1.* = tcp, *
Router.Row.NetA.Subnet2.* = tcp, *
Router.Row.NetB. *        = udp tcp, NetA.Subnet1.R2
Router.Row.*              = tcp, NetA.Subnet2.R1
```

That routing table contains the next information:

- All the objects prefixed with `NetA.Subnet1` or `NetA.Subnet2` are directly reachable (direct delivery).
- All the objects prefixed with `NetB` may be reachable by means of the router `NetA.Subnet1.R2`.
- For everything else, it must use `NetA.Subnet2.R3` (the default router).

Each row in the routing table includes an endpoint type list which may be used to contact objects connected to that network. Because an IDM network is formed one or "physical" networks using the same compatible protocol set. To use the IDM ALP mechanism at least a compatible type of endpoint must be known.

A. Dynamic Routing

It results relatively easy to implement dynamic routing algorithms. Each routing protocol message is defined as a object method in the specific interface. To support a new routing protocol is enough that the router needs to implement the respective interface as a new facet. As example, the next listing shows the interface for the RIP2 [7] protocol:

```
module Routing {
   sequence <Ice::Identity> IdentitySeq;

   interface RIP {
      void request(IdentitySeq oids, Ice::Identity src);
      void response(IdentitySeq oids, Ice::Identity src);
   };
};
```

In both methods, the parameter `oids` is an router identifier list (that may be empty in the `request()` message) and the `src` parameter is the identifier of the router that makes the invocation.

This approach allows to implement several dynamic routing protocols in the same router, although in that case, it must exists an administratively defined criterion to avoid conflicts in the routing table updates.

B. Network Management

Because routers and servers are conventional distributed objects, it is possible to use all the standard services available in the middleware such as application deployment or remote management (IceGrid in the case of Ice). This allows very interesting features, some of them found in a current network management platforms, but with no significant overhead.

a) *Planning* The topology design, the user and network requirements analysis, the maintenance and operation costs may be calculated before the system deployment. b) *Deployment* It allows distribute configuration files, libraries and customized binaries for each host or routing device. c) *Monitoring/Accounting* The system performance, available bandwidth, mean latency and many other variables may be measured easily in each router or server. d) *Configuration Management* New or improved software version may be deployed to routers, including replacements for routing protocols or administrative priorities.

VII. Prototypes and results

We developed some early prototypes to illustrate the possibilities and to evaluate the overhead of IDM. IDM is compared with conventional Ice invocations and traditional TCP/UDP sockets. The following results are obtained in a compatible IBM-PC computer with Intel Core2 Duo 2.33GHz processor, 100Mbps Ethernet NIC and Debian GNU/Linux OS. The programs are built with ZeroC Ice version 3.3 with GNU GCC 4.3 compiler.

These latency and throughput tests are inspired by the benchmarks made by ZeroC that are available in their webpage [9] as well as the source code used to make the tests.

The intention of these tests is to prove that the IDM overhead does not imply significant performance degradation.

TABLE I
Latency benchmarks.

Latency	**socket**	**ICE**	**IDM**
(Twoway)			
Local	23	31	32
Remote (direct)	268	274	278
Remote (1 IP hop)	312	360	384
Remote (1 IDM hop)	N/A	N/A	2 537
(Oneway)			
Local (oneway)	8	13	15
Remote (direct)	76	98	112
Remote (1 IDM hop)	N/A	N/A	1049

TABLE II
Throughput benchmarks.

Throghput	**socket**	**ICE**	**IDM**
Local	292	365	485
Remote (direct)	42 394	43 051	43 276
Remote (1 IDM hop)	N/A	N/A	98 291
Remote (2 IDM hops)	N/A	N/A	101 265

Moreover, note that we are using TCP endpoints for all the tests. When more suitable endpoints are available it will be possible to obtain better results than Ice and even better than raw TCP/IP sockets in many cases. For example, with the adequate application requirements we may send a multicast multimedia stream in a LAN encapsulating messages directly on Ethernet frames. This is an important overhead reduction.

The full source code used to make these tests, the current reference implementation of the prototype and other relevant information is available in the IDM webpage [10].

A. Latency

The latency tests measure the time needed to complete an invocation to a remote object sending a null or very small payload. The table I shows the average times for a set of 100 000 `ice_ping()` method invocations. This is the simplest message that may be implemented in Ice, it does not have any argument nor it returns a value. For the sake of comparison with BSD sockets, the raw socket test sends single byte requests and replies. In order to prevent batching of several messages in a single segment each message is forced to be sent immediately with a `flush()` call. Twoway invocations are synchronous, they will not be sent until the previous response arrives. Oneway invocations may be sent without a pause.

B. Throughput

Our performance tests measure the efficiency of the protocol when transmitting raw data. Table II shows average times per invocation where each invocation sends 500KB of raw data in a loop of 1000 iterations. All the invocations are twoway.

C. API

We intend to deploy IDM as a library for advanced end-to-end services in a way as transparent to the user as possible.

In this section we will see a brief overview of the library interface.

From the point of view of the application programmer, server implementation is almost identical to the standard Ice implementation. A special object adapter must be used, it is called `IDM::Adapter`. This is a simple decorator of the standard Ice ObjectAdapter. When this adapter is created, it will automatically be registered in the available *ALP* service. The following C++ listing shows the implementation of a minimal IDM accessible object.

```
class HelloI : virtual public Demo::Hello {
public:
  void sayHello(const string& name,
                const Ice::Current& current) {
    cout << "Hello from '" << name << "'" << endl;
  }
};

class SimpleServer : public Ice::Application {
public:
  int run(int argc, char* argv[]) {

    Ice::ObjectPtr serv = new HelloI;
    IDM::AdapterPtr adapter =
      new IDM::Adapter(communicator(), "testAdapter");
    adapter->add(serv,
                 communicator()->stringToIdentity(argv[1]));
    adapter->activate();

    communicator()->waitForShutdown();
    return EXIT_SUCCESS;
  }
};

int main(int argc, char* argv[]) {
  SimpleServer app;
  return app.main(argc, argv);
}
```

The IDM client requires certain modifications in relation to a conventional Ice client implementation. It needs to create an `IDM::Socket` instance and then invoke the `plug()` method, that takes an object identifier and returns a proxy to make invocations to the specified remote object. That class is responsible for looking up the node routing table and uses the ALP Service to get the reference to contact the remote object (either for direct or indirect delivery). The listing bellow shows the client implementation to access the server shown above.

```
class SimpleClient : public Ice::Application {
public:
  int run(int argc, char* argv[]) {
    IDM::SocketPtr sock = new IDM::Socket(communicator());
    Ice::ObjectPrx obj = sock->plug(argv[1], IDM::QoS());

    Demo::HelloPrx prx = Demo::HelloPrx::uncheckedCast(obj);
    prx->sayHello("John");

    return EXIT_SUCCESS;
  }
};

int main(int argc, char* argv[]) {
  SimpleClient app;
  return app.main(argc, argv);
}
```

VIII. Conclusions and future research

At first sight the design and implementation of a network protocol on top of the transport layer may seem contradictory or useless. We tried to show in this paper that the features provided by IDM may become very valuable in many different scenarios. Some of them are:

- Technology-independent universal logical addressing.
- Transparent end-to-end transport.
- Better efficiency due to the fact that the application designer may easily select the most appropriate transport.
- Additional features may be added transparently and they may be decoupled from the transport, such as ciphering, accounting, authentication, authorization, etc.
- Functional routing.
- Router and node migration.

The potential of IDM is currently being studied in the following areas:

- MANET Networks (*Mobile Ad-hoc Networks*). Each node in the ad-hoc network may include an IDM router, achieving an homogeneous access to the main network.
- Implementation of the IDM router in embedded systems based on microcontrollers and FPGAs for the development of highly efficient gateways for SAN networks and heterogeneous clusters.
- Metrics and evaluation of network performance and quality of service.
- Virtual network providing QoS services. An early proposal was already published by some of the authors of this paper.

Regarding the implementation of new endpoints, we must highlight that a wide variety of protocols such as Zigbee, BlueTooth and Ethernet are being developed in our prototypes. We are also prototyping *endpoint decorators*. These are a set of stackable endpoints which may be used to provide additional features to an existing endpoint, such as ciphering, delivery guarantees, accounting, etc. These decorators allow easy composition of endpoint functionalities leading to interesting combinations. For example, to transmit a private audio stream over an Ethernet network we may use an Ethernet endpoint which is able to transport data directly on Ethernet datagrams. This endpoint would be decorated with a ciphering endpoint to guarantee privacy. We would get the minimum protocol overhead for the requirements of the application, avoiding in this example the UDP and IP headers.

References

[1] A. Dunkels, J. Alonso, T. Voigt, "Making TCP/IP Viable for Wireless Sensor Networks", The First European Workshop on Wireless Sensor Networks. *EWSN 2004*.
The First European Workshop on Wireless Sensor Networks

[2] A. Dunkels, "Full TCP/IP for 8-Bit Architectures. In Proceedings of the first international conference on mobile applications, systems and services" *MOBISYS 2003*, San Francisco, May 2003.

[3] G. Holland, N. Vaidya, "Analysis of TCP Performance over Mobile Ad Hoc Networks" *MOBICOM'99*, August 1999.

[4] F. Moya, D. Villa, F.J. Villanueva, J. Barba, F. Rincón, J.C. López, J. Dondo "Embedding Standard Distributed Object-Oriented Middlewares in Wireless Sensor Networks," *Wireless Communications and Mobile Computing Journal*, Mar 2007.

[5] Object Management Group, "CORBA Messaging", in Common Object Request Broker Architecture (CORBA) Specification, Version 3.1, Part 1: CORBA Interfaces, ch. 17, pp. 419-494, Jan 2008.

[6] Object Management Group, "ORB Interoperability Architecture", in Common Object Request Broker Architecture (CORBA) Specification, Version 3.1, Part 2: CORBA Interoperability, ch. 7, pp. 15-64, Jan 2008.

[7] G. Malkin, "RIP Version 2", RFC 2453, Nov 1998.

[8] S. Deering, R. Hinden, “Internet Protocol, Version 6 (IPv6)”, RFC 2480, Dec 1998.
[9] Ice Perfomance Tests. Available online at http://www.zeroc.com/performance/index.html.
[10] IDM Webpage. Available online at http://arco.esi.uclm.es/idm.