Simulating SMEPP Middleware

Javier Barbarán, Carlos Bonilla, Jose Ángel Dianes, Manuel Díaz, Ana Reyna

Dpt. Lenguajes y Ciencias de la Computación
University of Málaga
Campus de Teatinos, 29017 Málaga. SPAIN
Tel: +34 95 2131394
Email: (barbaran, cba, jdianes, mdr, reyna@lcc.uma.es)

ABSTRACT

Embedded Peer-to-Peer Systems (EP2P) represent a new challenge in the development of software for distributed systems. The main objective of the SMEPP (Secure Middleware for Embedded Peer-to-Peer Systems) project is to develop a new middleware, based on a new network centric abstract model, specially designed for the above described systems, and trying to overcome the main problems of the currently existing domain specific middleware proposals. This paper presents a SMEPP Middleware component-based simulation tool. The main objective of developing this simulator is to provide a tool to enable the testing of the service model proposed for the middleware and to provide a framework to test different middleware design choices. Simulations will help us to make future decisions. Simulating SMEPP applications, that is, applications running on the SMEPP middleware and based on the API that it offers, help us to make decisions about the most requirement-satisfactory way of constructing the middleware. The simulated middleware API component represents a first approach to middleware design, and introduces some of the architectural issues that must to be solved in the near future.

Categories and Subject Descriptors

EP2P systems, service oriented middleware, component based simulators, routing protocols.

General Terms

Algorithms, Measurement, Performance, Design, Languages, Theory.

Keywords

Middleware, EP2P, Simulation

1. INTRODUCTION

Embedded Peer-to-Peer Systems (EP2P) represent a new challenge in the development of software for distributed systems. These systems have brought about an important revolution in distributed computing paradigms, now that the roles of client and server, which are the basis of the most widely used distributed computation models, are disappearing. The new scenario consists of systems in which all the elements of the network are symmetrical and, in most cases, the mechanisms of communication are not based on pre-existing infrastructures, but rather on dynamic ad-hoc networks among peers. At the same time, the recent technological advances in short distance wireless communications have opened up new areas of application which represent an important technological challenge. One of the key factors in the success of these systems is the possibility of abstracting all these problems by means of appropriate middleware. The main objective of the SMEPP (Secure Middleware for Embedded Peer-to-Peer Systems) project is to develop a new middleware, based on a new network centric abstract model, specially designed for the above described scenario, and trying to overcome the main problems of the currently existing domain specific middleware proposals. The SMEPP[1] Middleware platform will have to comply with the following general EP2P objectives: adaptability, scalability, high availability, and ubiquity, as the model will be based on the possibility of incorporating and removing resources in a dynamic and adaptive way, and users can access those resources offered by the network anytime, anywhere.

One of the key factors in the success of the definition of the SMEPP abstract model is to provide suitable support tools that can help in the design and testing of the application from the first steps of the development process.

As we said before, the development of EP2P applications is a complex task and simulation and validation activities at the initial stages of the development process can help to minimize possible design errors that would be difficult to discover during the deployment phase. In this sense, the SMEPP simulation tool can be very useful to test the early designs of applications, in a single node, with different configurations for the different nodes of the network and with a high-level description of the network behavior. In the case of application developers and in order to simplify the development of the application prototypes, the behavior of the peers can be described as a multithreaded program in C# or as a SMoL program [1,3].

2. SMEPP MIDDLEWARE OVERVIEW

2.1 SMEPP Features

From our point of view, the main characteristic of Peer-to-Peer networks is that the elements of these networks communicate in a bidirectional and symmetric way with each other. If this type of connection is not provided directly by the underlying network, a virtual network will be set up on top on the existing overlay network. In this sense, the term P2P can also be applied in a more generic context to name the set of communication models that provides this type of end-to-end communication, independently of the application and the network protocols used to construct this end-to-end communication on top of the overlay network.

SMEPP will support infrastructure and infrastructure less networks (ad-hoc networks) by re-using state of the art protocols

[1] This work is supported by the SMEPP Project (Secure Middleware for Embedded Peer-to-Peer systems). Information Society Technologies (IST) Programe. (FP6-IST-033563)

SIMUTOOLS 2008, March 03-07, Marseille, France

DOI 10.4108/ICST.SIMUTOOLS2008.3016

and implementations in order to allow the SMEPP peers to communicate and reach each-other. In terms of physical media, again SMEPP is agnostic, wired and wireless communications are relevant for SMEPP applications. It is true that sensor networks are recently being developed using wireless communication technologies such as Zigbee or even WiFi, therefore the SMEPP applications will be most probably using this type of implementation when relevant. It is especially important to highlight the differences between Ad-Hoc Networks and P2P file sharing platforms. In both cases we are dealing with self organizing networks, where P2P communications exist, but the objective and the technologies used to build the virtual P2P communication channels (mainly the routing algorithms) are very different.

The SMEPP Middleware platform will have to comply with the following general EP2P objectives: adaptability, scalability, high availability, and ubiquity, as the model will be based on the possibility of incorporating and removing resources in a dynamic and adaptive way, and users can access those resources offered by the network anytime, anywhere. SMEPP applications will be covering and spanning over a very heterogeneous terminal, gateway, sensor and device ecosystem (applications may run on different devices, from PCs, laptops, mobile phones and PDAs to sensor network nodes, with quite different network bandwidths, memory capacities and computing power). The networking and routing protocols used will have to support the important dynamicity of the network topology (the elements come into the system and go out in an independent way, involving frequent reorganization of the systems). The latter objectives represent important technological challenges to tackle in the project such as networking decentralization, network paths with transitory communications (connections and disconnections happen in an unpredictable and frequent manner) and a constantly changing topology.

The SMEPP middleware -and in particular its API- is service-oriented. Services are first-class citizens in SMEPP, and are described by contracts. Contracts will be used for matching (which will be in turn used within discovery) and also for verification and analysis.

A file containing the description of the service is called “Service Contract” or briefly “Contract”. A contract must contain all the information that any client (human or software) of the service may need to discover, to instantiate, and to interact with the service. A contract is no other than metadata describing the signature and behavior of a SMEPP Service.

The service oriented model is defined by an API including the following services:

- newPeer
- createGroup
- getGroups
- joinGroup
- leaveGroup
- publish
- unPublish
- getPeerID
- getGroupIDs
- getServiceContract
- invoke
- receiveMessage
- receiveEvent
- reply
- smepp_event
- subscribe
- unSubscribe

For a detailed syntax and description of these services see the SMEPP Service Model Description [3].

2.2 SMEPP workflow and workgroup

Together with the middleware infrastructure, a customizable component framework will be developed. This framework will provide the tools necessary to adapt the middleware infrastructure to different embedded devices and networks.

Two validation applications will be developed: one in the field of environmental monitoring of industrial plants and in mobile telephony and the other in context aware computing. These applications will be used to obtain the main requisites of the middleware and will help to demonstrate the suitability of the new middleware for the development of applications for these types type of environments. The application domains are different enough to study the flexibility and adaptability of the middleware and its associated tools.

The SMEPP consortium is composed of a mixture of research institutions and industries that bring together the complementary skills and the expertise necessary for the project, with experience in middleware development, security and different aspects of software development for embedded systems, including Telefonica I+D, Tecnatom S.A., Universidad de Málaga, Technische Universitaet Graz, Siemens Aktiengesellschaft, Valtion Teknillien Tutkimuskeskus of Oulu in Finland, Universita de Pisa and the Institute for Infocomm Research at Singapore.

3. SIMULATION TOOL

3.1 Objectives

The main objective is to provide a tool to enable the testing of the service model proposed for the SMEPP Middleware and to provide a framework to test different middleware design choices.

Why simulate a middleware?

Simulations help us to make future decisions. Simulate SMEPP applications, that is, applications running on the SMEPP middleware and based on the API that it offers, help us to make decisions about the most requirement-satisfactory way of constructing the middleware. Specifically, we believe that these decisions may affect different development areas and for different reasons.

First of all, the simulator development, that can be seen as a first approach to the final middleware development with the proper design and implementation process of the simulated API that simulated applications have to use, has generated some questions about the architecture of this API, about its structure in different functional units and abstraction levels. The particular SMEPP features, a middleware ideated to support multiple hardware platforms and operative systems, with different programming paradigms, has produced a very rich API, but a very complex and heterogeneous one, which needs some kind of structure in order to be easily understandable and usable. These and other conclusions, mainly related with the middleware architecture, will be described in the following sections.

Related to the different interaction and service models, using the simulation tool, with its different options of executing programs and interacting with them by the users, allows them to decide upon the suitability of these models to specific problems and applications.

Finally, always when working with P2P distributed systems, routing protocols have to be taken into account. Discussing about them is a very important advantage that a simulation tool can offer, testing the best options and algorithms and defining the most appropriate interface with the rest of the middleware. The SMEPP Simulator tool allows users to configure and interchange different routing algorithms for the same set of simulated applications, allowing them to reach important conclusions. Also security issues may be introduced as a part of the simulation tool and experimented in the execution of programs. For this purpose, the simulation environment will offer mechanisms for the definition of specific security policies for SMEPP groups.

3.2 Functionality

SMEPP Simulation Tool accepts as input different SMEPP programs, a network definition and some SMEPP Peers definition, in order to execute over the simulated middleware rather than distributed in real nodes.

The SMEPP Simulation Tool is a 'black-box simulator'. The simulated application is an executable, generated from a SMEPP application program together with different components described below in section 3.3. Simulation interaction is performed in two ways:

- *Output events*, generated by the simulator components, like SMEPP primitive call events, network events, etc. These events can be displayed in the simulator GUI.

- *Input user events*, defined by users and generated by the simulator component. These mainly include network dynamics events. Real world conditions also may be defined using interaction mechanisms like events or variable value assignment.

The main functionalities to be achieved by the simulator are: Monitoring the simulation (Monitoring the SMEPP middleware), Control of the simulation (Start simulation., Stop simulation.), Configuration of the simulation (Definition of nodes and connections (network topology), Definition of peers, its behavior and its services, Definition of events, Configuration of the routing protocol.)

As mentioned, there are different kinds of input files for the simulator tool, specifically five. These files contain the information that the simulator needs to configure a specific application environment (to generate the simulator configuration file). The simulator tool does not exclusively simulate the execution of a single SMEPP program but it must be able to simulate the whole application environment. This means, that the nodes that take place in the simulation must be defined, this is, all the devices that are running the SMEPP middleware have to be configured. Additionally, the peers that are running at each device, the services provided by each peer, and, finally, the SMEPP programs associated with each peer and service, must be configured by the user. To ease the configuration, the simulator tool is based on these input files, and support tools to create and modify this input files, are provided.

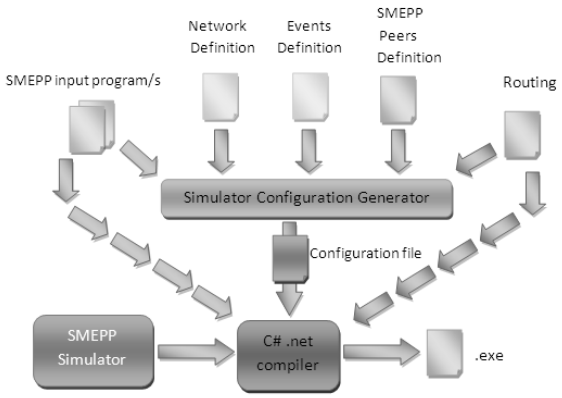

Figure 1 Functionality

3.2.1 SMEPP Program File

A SMEPP program describes the behavior of a peer or a service; this is achieved through invocation to the SMEPP API primitives. The SMEPP Simulator needs a modified version of the version provided by the SMEPP application developer; because this SMEPP program must interact with the developed simulation tool, which simulates the middleware, but the execution in the simulator is guided by the user. This is the main difference with respect to the execution of the program in the real world. The modifications of the original SMEPP files are minor changes, they do not burden the developer, and nevertheless the modified version needed can be automatically generated by a parser tool.

3.2.2 Network Definition File

This input file must provide the description of the topology of the overlay network, over which the SMEPP programs are going to be simulated. This means that all the devices (network nodes) taking part in the simulation must be specified, with its properties (such as bandwidth capabilities, maximum running peers…) and its connected neighbors. Additionally the routing protocol must be specified.

3.2.3 Peers Definition File

As important as the definition of the network, is the definition of the peers. Peers are processes that run over the devices (network nodes) and their behavior is modeled through a SMEPP program. A single device can execute more than one peer (process), but a peer runs only in a single node at each time, but not necessarily the same node during the whole simulation in order to model mobility.

This file must provide the relationship between the nodes defined, and the SMEPP programs provided. Associating a peer with a specific network node means that the peer will be initially running in that node. In addition, for each peer, the services that it provides must also be specified with the reference to the SMEPP program which defines the service behavior.

3.2.4 Network Events Files

Users can specify predefined network events such as location changes or node failures. Events could also be defined using a probability or be randomly generated by the simulator. This is useful to test the middleware and the application in special scenarios.

3.2.5 Routing

Finally the user has to provide the routing protocol component. Different components can implement different routing behaviors, creating different simulation scenarios.

3.3 Architecture

Using a component based Architecture makes applications very flexible because of their component "plug and play" nature. This is really useful in the case of the simulation tool because it will enable us to test the same applications under different middleware implementations, routing protocols or network overlays.

The simulator architecture is divided into four components that interact: Simulation Framework Component, SMEPP Middleware Component, Network Overlay Component and Routing Component. Their relationship is shown in the following figure:

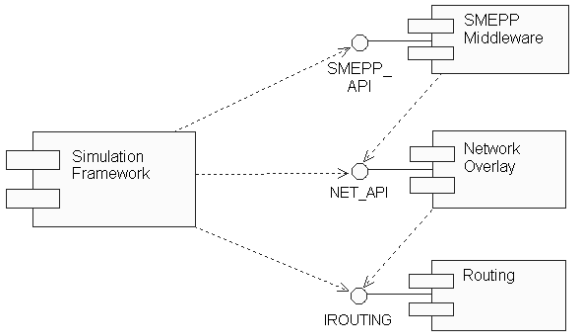

Figure 2 Simulator Component Based Architechture

3.3.1 Simulator Framework Component

This component is responsible for the configuration of the whole simulation and its monitoring. Interactions between the user and the simulations are also carried out here, using definitions of the configuration file and using run-time interactions through the Graphic User Interface of the simulation tool. This GUI is composed of different tab panels. Figure 3 shows the appearance of the simulator graphics.

Tab panels include Configured Peers, Configured Network, SMEPP API Primitives, Middleware Groups, Middleware Peers, Middleware Services and Routing. All will be described later.

3.3.2 SMEPP Middleware Component

This is one of the key components; it provides a preliminary implementation of the SMEPP middleware, providing the same API that the SMEPP middleware will provide. Changing this component to new versions of the middleware will be useful to test and compare different approaches during the development. In addition, this fist implementation is useful to evaluate the service model proposed.

Many architectural issues have been approached during the development of this component. Data structures for storage and management of SMEPP entities, like peers, groups, services and events, are included here. Interaction mechanisms between all these entities are also implemented into this component, including raising and listening of events, synchronization of service invoke-response sequences and data integrity when being accessed concurrently.

Doing so, we have approached a solution to some of the problems of the real middleware development.

3.3.3 Network Overlay Component

The NetOverlay component abstracts the operating system network interface underlying the middleware. For this version of the simulator, we have chosen a simple unidirectional point-to-point channel-based API. This API includes the following operations:

```
NetChannel createChannel(int nodeId);

void destroyChannel(NetChannel channel);

NetChannel connectChannel(int nodeId, int source);

NetChannel getInputChannel(int nodeId);

NetChannel[] getOutputChannels(int nodeId);

void send(int nodeId, NetChannel channel, Object[]
data);

Object[] receive(int nodeId, NetChannel channel);
```

Network entities include channels and nodes, the first represented by NetChannel instances, the second by an int id.

Routing is performed each time data is sent or received, and not when creating or connecting channels. We do this in order to emulate an ad-hoc network.

3.3.4 Routing Component

The SMEPP Simulator tool has been designed to enable the implementation of different routing protocols. This is achieved through the definition of the routing component. This component interacts with the SMEPP API to enable peers to find services or others peers connected within the overlay network. This enables the simulator to test different protocols over different networks easily.

Distributed Hash Tables (DHT) are the most common approach to routing P2P networks, and they have had a revolutionary effect in the decentralization of this kind of networks. We have selected an implementation of this type of protocol in order to study the requirements of the routing component interface in the context of the simulator. Our final objective is to be able to change this component with the different security routing protocols which will be defined in the context of the other SMEPP workpackages in order to test their suitability to SMEPP objectives.

A DHT basically has the functionalities of a hash table and its purpose is to distribute the storage and search of the hash table between several distributed nodes. Thanks to the structured topology, data lookup becomes a routing process with low routing table size and maximum path length. DHT also offers high data location guarantees.

In this first implementation, we have implemented an approximation of Chord [7], a distributed lookup protocol based on Distributed Hash Tables.

3.4 User interface

The Graphical User Interface of the simulation tool (figure 3) is composed by two panels, the first one, on the left side, is a simple control panel, and the other one contains seven tab panels with different information:

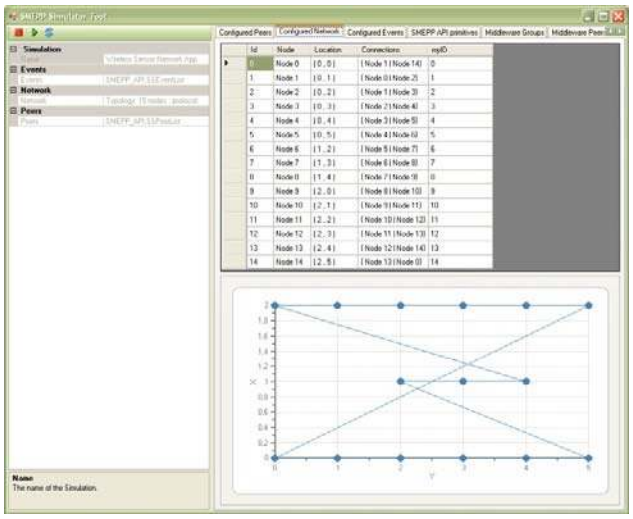

Figure 3 Simulation Tool GUI

- Configured Peers: contains information of each defined peer, the node where is deployed, the program that is executing and the services that will provide.
- Configured network: shows as a table and as a graph, the topology of the network (interconnection of nodes, independently of the peers that contains).
- SMEPP API Primitives: a log with a trace of the different SMEPP API primitives that are being used by the simulated application. Normally, the trace contains information about the invoker of the primitive and other arguments.
- Middleware Groups: information about the created SMEPP groups, including members and services of these groups.
- Middleware Peers: the difference with respect the firs tab panel is that that one contains the information predefined by the user, and this one contains the current state of the middleware. For example, if a peer fails to create, still will appears in the 'Configured Peers' panel, but do not appear in the 'Middleware Peers' panel.
- Middleware Services: lists all the published services, including information about the publishers (peers and groups) and the service contracts.
- Routing: it shows traces of routing each time that a node sends information to another.

3.5 Some implementation details

The SMEPP Simulation Tool has been developed in C# using Visual Studio 2005 to run over Windows. Concretely, all testing has been made over Windows XP. So, applications that are going to be simulated have to be written in C# too.

The GUI has been made using Component One framework for the development of visual components. The simulation kernel uses different events to alert the GUI of the different changes.

4. SAMPLE APPLICATION

A large number of scenarios can be simulated using this tool. Due to the combination of embedded and peer-to-peer characteristics for Wireless Sensor Networks (WSNs), this kind of application seems to be the most interesting one that can be simulated using the SMEPP Simulator Tool. For this reason in this paper we propose an application for monitoring environmental conditions in buildings using WSNs. This application must perform different activities, such as: Indoor environmental monitoring (heating, ventilation and air conditioning). On the other hand it is also necessary to respond to extreme events such as fire.

Each aforementioned goal involves only a specific part of the system. In WSNs keeping the processing close to where the data is sensed has been long recognized as an effective approach to save energy, achieve more efficient implementations and support real-time requirements.

In Figure 4 is shown the application schema that is going to be simulated. A building will have several sensors deployed, measuring different conditions, such as humidity, temperature and smoke. But there will also be other devices that are able to react in response to sensor measurements, such as air conditioners, sprinklers and fire alarms. Additionally, the application could send as well some information to external devices (for example, firefighter PDAs).

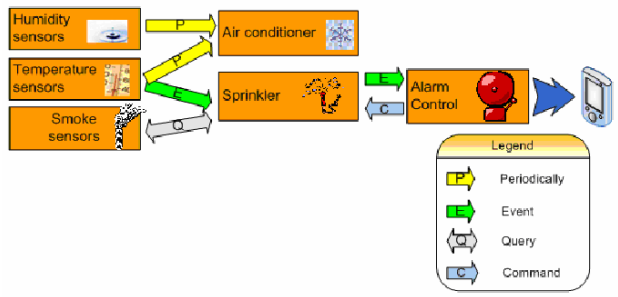

Figure 4. Sample application schema

Specifically, we have considered a room in a building that is equipped, as shown in Figure 5, with an air conditioner, water sprinklers and a fire control alarm, but is equipped also with a set of sensors measuring temperature, humidity and smoke conditions.

Following the operational setting described above we have designed a SMEPP application, using the C# programming language, which provides a solution to for the proposed scenario. Conceptually, the application is composed of the following peers: Sprinkler, Air Conditioner, Fire Control and sensors measuring different conditions.

SMEPP abstractions provide an elegant way to solve complex applications where different peers work together in order to find a global objective. The following groups have been created for the sample application: Temperature, Humidity, Smoke, Fire Control groups. In temperature are involved temperature sensors, but also air conditioner and water sprinkler peers. In the humidity group an air conditioner and a humidity sensor are needed. The smoke group is composed of smoke sensors and sprinklers. Finally, in the fire control group, a sprinkler and a fire alarm control peer are involved.

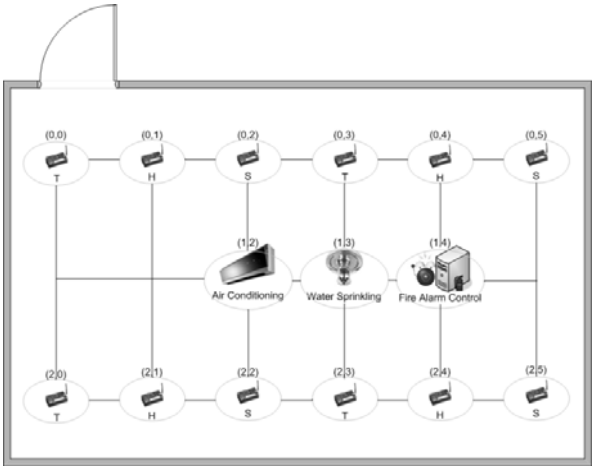

Figure 5 Sample Application

Peers publish their services in their groups, this way other peers can invoke them. For example, temperature sensors publish a service that is composed of a high temperature event and a periodic event that gives a temperature value each 5 seconds. On the other hand sprinkler peers subscribe to temperature room services in order to receive temperature values and high temperature events.

This kind of system due to it distributed characteristic are no predictable, for this reason the use of SMEPP simulator helps to the application developer to check that the code is free of deadlocks, but also free of abnormal behaviors. In the sample application, two different versions where simulated in order to check the suitability of the simulator: one with a deadlock and the other one without deadlocks. The sample application sketched before is designed to send continuously events from sensors to actors. The behavior of the simulator in the first case with the deadlock it is an abnormal stopped in the flow of primitives without any reason, then the user is announced that a possible deadlock is reached. In Figure 6 the execution of the application without deadlock is shown. The execution after the services are published and the peers are joined to the right groups, is composed by events received and sent that will not stop until the user stops the simulation.

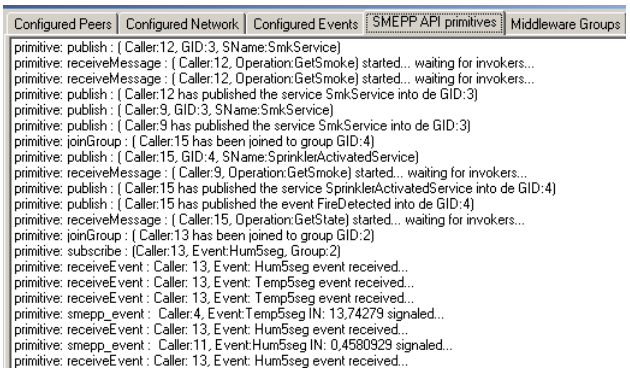

Figure 6 Flow primitives on free-deadlock execution

5. CONCLUSIONS

From the application programmer point of view it is clear that the tool will help to accomplish the objectives discussed in the introduction. During its development we have found some details in the definition of the API that can be improved. We think that the simulator can really help in the design of even simple application such as the one shown in the previous section.

Middleware developers have therefore taken some advantages from the simulator development so. The simulated middleware API component represents a first approach to middleware design, and introduces some of the architectural issues that must to be solved in the next future. These issues include concurrency models, suitable network abstractions, architectural design on its own, data structures and their management for entities storage (like peers, groups, services etc), service model lapses, etc.

Taking these objectives into account, we think that starting to develop our own simulation environment instead of using a general purpose simulation tool, was the best way to proceed.

The component-based approach of this simulator has many advantages. The most important is the possibility of representing different scenarios for the same application program. This will be done using different interchangeable components, like the routing component for testing different P2P routing algorithms, the network overlay component for testing different operating system network APIs, or even the own SMEPP API Component for testing different architectural design or concurrency models.

Finally, in the future, when de SMEPP middleware has been completely developed, the simulator will represent the first step in real application development. These kinds of massive distributed applications are very difficult to test and validate in real world conditions, so a simulator is a very useful tool, saving a lot of time and effort (and money) to application developers.

6. REFERENCES

[1] Antonio Brogi, Razvan Popescu, Francisco Gutiérrez, Pablo López, Ernesto Pimentel. A Service-Oriented Model for Embedded Peer-to-Peer Systems. In proceedings of the 6th International Workshop on the Foundations of Coordination Languages and Software Architectures, Lisbon, Portugal, September 8, 2007.

[2] M.Albano, A.Brogi, R.Popescu, M.Diaz, J.A.Dianes. Towards Secure Middleware for Embedded Peer-to-Peer Systems: Objectives & Requirements. Second Workshop on Requirements and Solutions for Pervasive Software Infrastructures, Innsbruck, Austria, September 16, 2007.

[3] SMEPP Consortium D.2.1 SMEPP Service Model Description.

[4] SMEPP Consortium D.2.2 Tool Support for the service Model

[5] Sameh El-Ansary. *Designs and Analyses in Structured Peer-to-Peer Systems.* June 2005

[6] Rüdiger Schollmeier, Ingo Gruber and Michael Finkenzeller. *Routing in Mobile Ad Hoc and Peer-to-Peer Networks. A Comparision.*

[7] Implementation of Chord. http://www.seas.upenn.edu/~cis505/spring2004/project2/chord505-0.3.tar.gz