JOURNAL OF OBJECT TECHNOLOGY

Online at http://www.jot.fm. Published by ETH Zurich, Chair of Software Engineering ©**JOT, 2004**

Vol. 3, No. 9
Special issue: .NET Technologies 2004 workshop, Pilsen

Design and implementation of a FIPA compliant agent platform in .NET

Miguel Contreras, Ernesto Germán, Manuel Chi and **Leonid Sheremetov,**
Mexican Petroleum Institute (MPI), Mexico.

Abstract

The aim of this paper is to describe the design and implementation of an agent platform called CAPNET (Component Agent Platform based on .NET) that is fully compliant with the specifications of the Foundation for Intelligent Physical Agents (FIPA) and implemented as 100% managed code in the .NET framework. The tools for the platform configuration and administration are presented and a case study is provided that shows the usability of the CAPNET in industrial and mission critical scenarios.

1 INTRODUCTION

Agents can be defined to be autonomous, problem-solving computational entities capable of effective operation in dynamic and open environments. Agents are often deployed in environments in which they interact, and maybe cooperate, with other agents (including both people and software) that have possibly conflicting goals [Luck03].

Agent-based software should be robust, scalable and secure. To achieve this, the development of open, stable, scalable and reliable architectures that allow compliant agents to discover each other, communicate and offer services to one another is required. These architectures go beyond the capabilities of the typical distributed object oriented programming techniques and tools. The FIPA's Agent Platform (AP) reference model seems to be an effective approach to address this problem [FIPA00023].

An AP is a software architecture that controls and manages an agent community allowing the survival and mobility of an agent in a distributed and heterogeneous environment.

In the last few years, several APs have been developed with a special attention paid to interoperability and compatibility issues. In this sense, the FIPA reference model has emerged as a standard for interoperability sustaining the development of APs. Most of the FIPA compliant APs were developed in Java language, such as JADE [JADE00], FIPA-

Cite this article as follows: M. Contreras, E. Germán, M. Chi, L. Sheremetov: "Design and implementation of a FIPA compliant agent platform in .NET", in *Journal of Object Technology*, vol. 3, no. 9, October 2004, Special issue: .NET Technologies 2004 workshop, pp. 5'28. http://www.jot.fm/issues/issue_2004_10/article1

OS [Nortel99] and Zeus [ZEUS00] just to mention a few of them. One exception to this trend was the original CAP [Sheremetov01] agent platform implemented using Microsoft DCOM and ActiveX technology. The purpose of CAP was to enable the construction and operation of multi-agent systems (MAS) using Windows programming languages and platform.

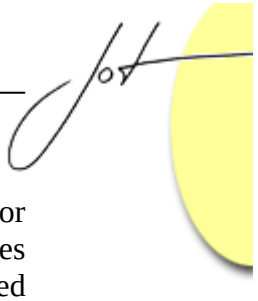
Our recent work has lead to the development of a completely new AP, named CAPNET under the novel Microsoft .NET Framework and Compact Framework in 100% managed code written entirely in the C# language. This new agent platform uses several of the technologies available in .NET and the Windows platform, such as Web Services (WS), remoting, asynchronous callbacks, delegates, XML, database connectivity, performance counters, event log, Winforms, Windows Services and network access, among others.

The main objective of CAPNET is to bring an integrated infrastructure that covers the programming, deployment, administration and integration with legacy applications of MAS (Fig. 1). It consists of a run-time environment that supports MAS deployment, development environment in the form of agent templates, programming tools and a component gallery and some connectors to enable the integration with enterprise applications.

The run-time environment is described in this paper focused on the design and technical details of its implementation targeting the .NET Framework and Compact Framework. Some detailed information describing agent templates implementation, are provided, focusing on internal structure and mechanisms for its integration with the rest of the platform. Inter-agent communication mechanisms are presented.

Figure 1. MAS development and deployment in CAPNET

This paper is structured as follows: in section 2, basic concepts of agents and multi-agent systems are presented along with a short description of the FIPA specifications that guided this work. In section 3, the CAPNET layered architecture is defined and its parts presented. In section 4, this paper describes the particular implementation of the FIPA reference model using .NET framework. Also this implementation is extended by inclusion of the security layer responsible for maintaining platform and infrastructure



security described in [Santana03]. In section 5 the most important administrative tools for the platform are described. Section 6 contains the implementation of the agent templates and its features. In section 7 some experimental results from benchmark test performed on the platform are analyzed, and a case study illustrating the use of the AP in an industrial scenario is presented. Finally, the main features and advantages of this work are discussed in section 8, comparing it with other implementations followed by conclusions and future work.

2 AUTONOMOUS AGENTS

Multi-agent systems offer a new and often more appropriate way of development of complex systems, especially in open and dynamic environments. Some key features of the agent technology support these capabilities. The autonomy and pro-activeness features of an agent allow it to plan and perform tasks defined to accomplish the design objectives. The social abilities enable an agent to interact in MAS and cooperate or compete to fulfill its goals. An agent may be static or mobile. In the latter case it is able to migrate along with its associated data, state, and logic to another host to interact with local resources and other agents to perform a given task.

Agents can be distinguished from objects (in the sense of object oriented software) in that they are autonomous entities capable of exercising choice over their actions and interactions. Agents cannot, therefore, be directly invoked like objects. However, they may be constructed using object technology [Luck03].

The open nature of the MAS is provided by the agent organization, similar to that of distributed enterprises, and supported in the agent platform's tools, which are responsible to provide flexibility both in component aggregation and interaction between them [Sheremetov03]. The AP reference model of the FIPA provides the framework of normative work, inside which the agents exist and operate; it also establishes the logical and temporal contexts for the creation, operation and destruction of agents [FIPA00023]. The reference model considers an AP as a set of four components where every one represents a set of logical capacities or services that can be combined in each AP concrete implementation: Agents, Directory Facilitator (DF), Agent Management System (AMS), and Message Transport System (MTS). The DF and AMS are special types of agents that support the management of other agents, while the MTS provides a message delivery service (Fig. 2).

The functionality of the main components of the FIPA spec is: the AMS offers white pages services and agent's life-cycle administration service, maintaining a directory of agent identifiers (AID contains the transport address by which an agent can be found) and the execution state of each agent. AMS is a mandatory component in an AP and every application agent must register itself with the AMS to permit the validation of its identifier in the multi-agent environment. The DF is the component that offers yellow pages service. Although it is an optional component, multiple DFs can register in the AP. DF allows agents register their services and be requested to find out the available

distributed services within agents community. The MTS is the software component controlling all the message exchange within the platform, including messages to/from remote platforms.

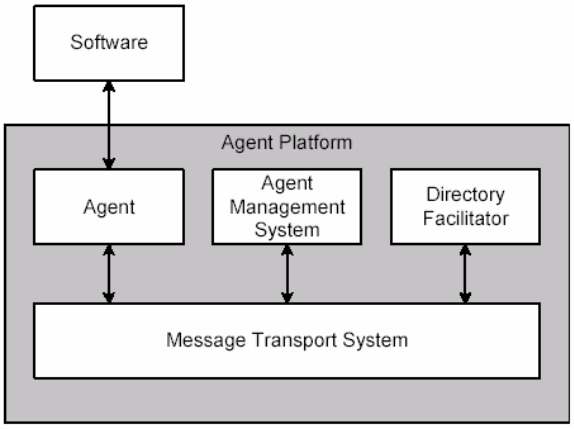


Figure 2. Agent Management Reference Model [FIPA00023].

Agents are the main parts of a platform. An agent encapsulates one or more services inside a unified and integrated execution model. FIPA maintains an open concept of what an agent is, to be able to include different agent architectures, and not limit the form in which they are implemented. The Software refers to all those systems that do not have characteristics of agents but that are used by agents to fulfill their tasks. Here, domain specific systems, as well as the Application Programming Interfaces (API) for handling communication protocols, databases, security algorithms, etc. are included. The AP internal design is an issue in the specification that is not subject of standardization, in the sense that agents can use any proprietary intercommunication mechanism using any standard middleware.

3 CAPNET ARCHITECTURE

The main purpose of CAPNET design is to construct a platform that enables developers to easily create and integrate distributed agent applications in a consistent and scalable way. The platform should be able to interoperate with applications developed in other APs as well.

The architecture of the CAPNET shown in Fig. 3 contains four main layers: i) application agents, ii) agent management and directory facilitator services, iii) built-in security services, and iv) message transport system and connectivity techniques.

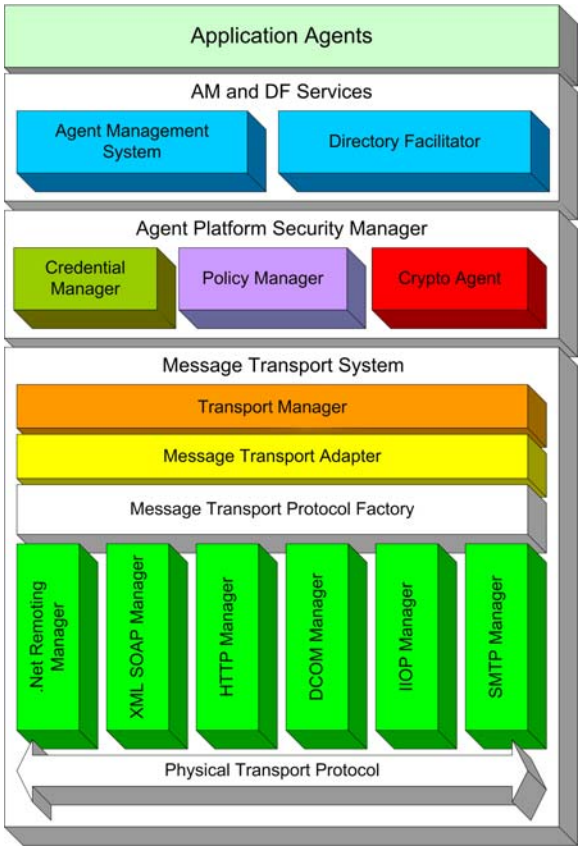


Figure 3. CAPNET architecture

In the Application Agents layer, the agents built from the agent templates interact between them and with the platform. This layer provides the functionality of the application MAS.

The administration services of the platform (AMS and DF) are located just below the agents in the architecture because they administer the life cycle and the directory of services provided by the agents.

In a heterogeneous multi-agent environment, security becomes an extremely sensitive issue. Security risks exist throughout the whole agent life-cycle: agent management, registration, execution, agent-to-agent communication and user-agent interaction. Agent Platform Security Manager (APSM) is responsible for maintaining platform and infrastructure security policies, authentication and run-time activities, such as, communications, providing transport-level security, and creating audit trails. The APSM is responsible (i) for negotiating the requested inter- and intra-domain security services with other APSM's on behalf of the agents and (ii) for enforcing the security policy of its domain. The description of the APSM is out of the scope of this paper; more details on its architecture and implementation can be found in [Santana03].

The services of transport, delivery and reception of messages represent a central point within the CAPNET platform, supporting a wide variety of transport types. Among the considered transport managers are all those required to ensure platform

interoperability with other widely available APs such as those mentioned in section 1. In CAPNET, an agent communicates with others using a service provided by the Message Transport System (MTS). This level of abstraction allows developers of multi-agent systems to design them based on schemes of loosely coupled messaging systems between components. With this type of asynchronous messaging the communication can be seen as a distributed messaging system similar to a Message Oriented Middleware [Bernstein96] or publish/subscribe architectures [Pallickara03].

To assure MTS reliability, some additional mechanisms are implemented. When an agent sends a message, it knows if the MTS has delivered it to the destination agents. If some addressee could not be contacted then the emitting agent receives a notification indicating that the message could not be delivered. It is the responsibility of the emitting agent to maintain its state or to block its execution while waiting for the answer.

Two cases for message delivery exist. The first one takes place when the addressee agent is accessible through the same MTS and the internal mechanism of the MTS is used. In the second case, the message addressee is registered in other MTS. In this case, a component called Message Transport Adapter (MTA) has been implemented to determine the type of message transport that is needed to complete the operation and to use the appropriate technological infrastructure for it. The MTA uses the services of Message Transport Protocol Factory (MTPF) to instantiate the component that really implements the access to the physical transport. This component in our architecture is called Transport Manager (TM).

If a new transport mechanism is required, a new TM component has to be created implementing the standard interface known by the MTPF and the MTA. If that interface is implemented, the interaction with the MTPF is assured and the particular implementation of the transport mechanism is completely open to the developer, and can include any form of communication both synchronous and asynchronous, such as raw sockets, MSMQ, FTP, SMTP, etc. The TM has to be registered in the platform in order to be used. At the time of writing the TMs for HTTP and .NET Remoting have been implemented, and several more are in process.

In the next section, technical details of the CAPNET implementation are described.

4 AP MESSAGING AND ADMINISTRATION IMPLEMENTATION

The Message Transport Mechanism

The MTS was implemented as a singleton remoteable object [Rammer01] that can be instantiated by the agents in order to be able to send and receive messages. This remote object is able to deliver messages to the agents based on delegated method and publish/subscribe mechanisms for events.

The mechanism for message delivery is a type of asynchronous callback allowing messages to be delivered as they are received by the MTS. The agents use a special class

working like a listener of incoming messages. This listener contains a delegate method, which is invoked when the MTS receives a message.

When agents are initialized they subscribe to the MTS sending a listener object. This object is registered in the events manager administered by the MTS. When a message has to be delivered by the MTS, it consults events manager to look for the listener of the destination agent of the message. This way it can invoke the delegated method that was registered for the listener of the agent.

If an agent wishes to send a message (Fig. 4), it contacts its MTS. When a message is received by the MTS, it is processed as specified by FIPA. When it is prepared for delivery to the destination agent(s), if any of them is located in a different CAPNET, MTS contacts the MTA to delegate the delivery. The MTA determines the required type of delivery based on the destination agent's address. Using this information the MTA obtains a concrete TM from the MTPF. That TM is the object capable of performing the real delivery of the message and once obtained, the MTA invokes the delivery functionality on it.

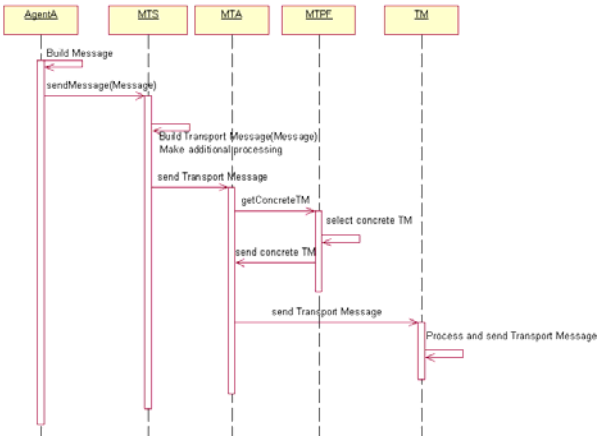


Figure 4. Sequence for message sending.

When a TM receives a message, it extracts the important fields and constructs a new platform specific Transport Message object to be sent to the known MTA. This MTA forwards the Transport Message to the MTS, which sends the message to the destination agent. Message receiving is similar.

The AMS and DF Services

The AMS and DF services of the platform have been implemented using a multi-tier architecture (Fig. 5). We shall only describe the AMS Service, since the implementation of the DF is almost identical.

The agent tier is implemented by the functionality provided by the BasicAgent class (described in the next section), and involves all the communication, interaction and conversation mechanisms that provide the social interaction capabilities. This layer is responsible for listening to other agents requests, implemented as an Agent

Communication Language (ACL) *request* message specifying the action and its parameters, and launching the corresponding services.

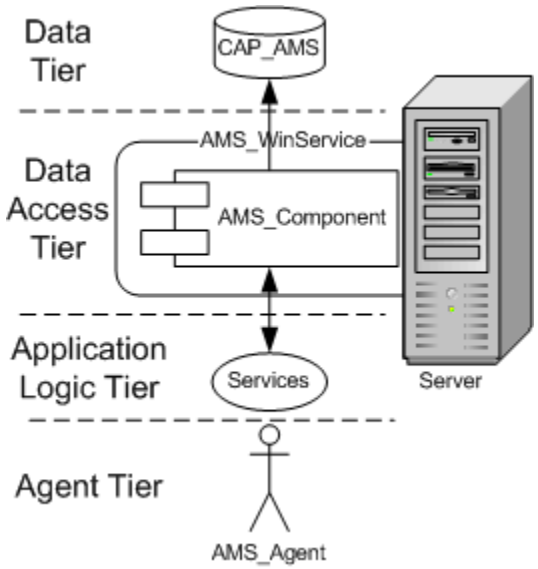


Figure 5. AMS Service implementation.

The application-logic tier is implemented in the services (actions) that the AMS_Agent is able to perform, and which are requested by other agents in conversations following predefined interaction protocols. The actions that this agent is able to perform are: registration, deregistration, modification of agent descriptions in the white page directory, and search of agent descriptions.

The data access tier is implemented in the AMS_Component which is a remoteable component hosted in a Windows Service. This component designed as a "singleton" exposes two interfaces: the IAMS interface for the AMS_Agent and the IAMSAdmin interface for the administration and monitoring applications, such as the Society Viewer (that shows the registered agents and their interactions) or the AMS_Administrator among others.

The AMS_Component registers and updates constantly a set of performance counters. These counters allow system administrators to monitor and record the state of the platform and to raise events or generate alarms under certain conditions they are interested in.

The data tier is implemented in a relational database using Microsoft SQL Server as the data base management system that stores agent descriptions of all the agents registered in the AP.

CAPNET interoperability extension

An interface allowing to expose the functionality of Agent Services (AS) to web based systems and vice versa has been implemented as a component of CAPNET allowing to consume the services of a MAS in a service oriented (non-agent like) way.

Today, Web Services are believed to be the crucial technology for web based business communities and can be seen as high-level interfaces through which partners can conduct business operations. A current trend is to WS-enable legacy applications allowing XML-based data exchange between a legacy application and a software client [Plummer00].

In the WS – AS integration scenario it is easy to see the need for some mechanism that allows agents to use the functionality provided by systems based on WS either inside or outside the organization and to allow those services to use the ones that agents in a MAS implement.

Integration of WS and AS requires the implementation of a bridge that is able to listen for web requests and, at the same time, to communicate with agents residing at the AP. Such a bridge has been built using standard technologies (such as SOAP, XML, UDDI, WSDL, etc.) in order to provide the functionality of MAS to the "outside world" and to consume resources external to the AP. The implementation details are discussed in [Sheremetov04b].

5 ADMINISTRATION TOOLS

Along with the platform, a set of tools for its configuration and administration have been developed. These tools include a society viewer, a ping agent, AMS and DF administrator, Pocket PC version of the AMS and DF administrator, platform's communication infrastructure configurator, etc. Only the three most representative tools are described in this section because of the space limits.

AMS administrator

One of the most important tools is the AMS administrator (Fig. 6), because it allows monitoring of agents registered at the AP, their state and properties. This tool keeps a log of all the events (registration, deregistration and modification of agent's properties) fired by the AMS service. This administrator is able to send life cycle related commands to the agents, such as destroy, quit, resume, suspend and wakeup. It is also useful to deregister agents from the platform to stop its execution or just to remove it in case it has terminated abnormally without deregistering first. The AMS Administrator also shows the current state of the AMS Windows Service and allows the user to start/stop it. Currently the platform has a desktop and a Pocket PC version of this tool with pretty much the same functionality.

Figure 6. AMS Administrator.

This tool is designed to connect directly to the AMS_Componet using the IAMSAdmin interface via .NET Remoting (Fig 7). When communication using remoting is not possible (because of network restrictions or using the Pocket PC version of the administrator) a Web Service is used to form a bridge to the AMS_Component.

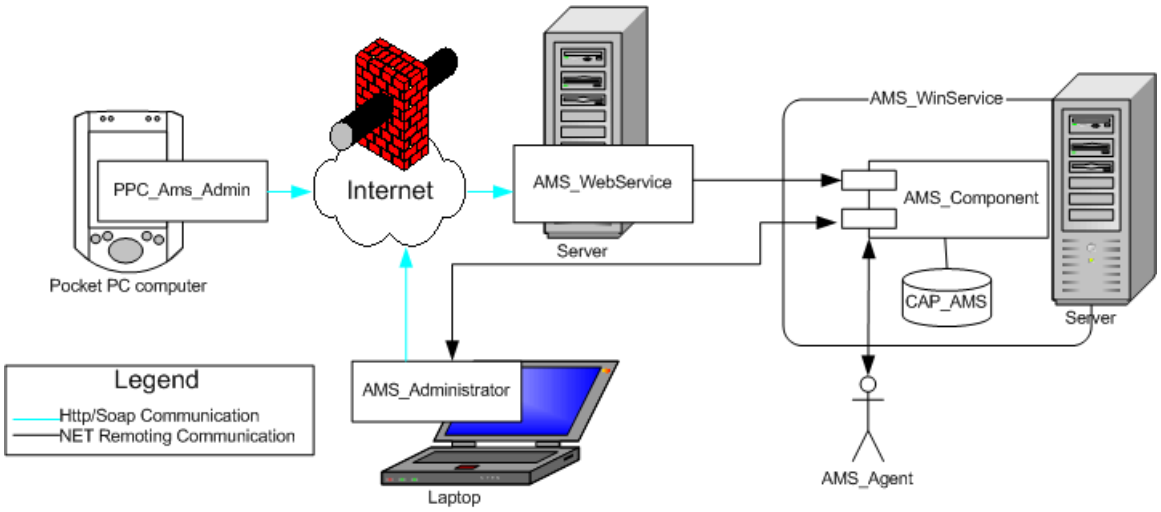


Figure 7. Connection of the AMS Administrator with the platform.

The latter approach however has a drawback: the administrator has to request constantly the last state of the AMS, whereas using remoting the Administrator is able to subscribe to the published events of the AMS_component, in order to receive instant notifications of any changes occurring in the AMS as a result of registration, deregistration or modification of agent descriptions in the AMS.

Society Viewer

The Society Viewer (Fig. 8) is another important tool. It displays CAPNET agents and messages they exchange. This tool is configured to be connected to the AMS and MTS services. The AMS notifies to Society Viewer about events related to the lifecycle of agents, so the tool displays agents living within the platform and any change of the agent's state. On the other hand, the tool shows the messages exchanged between agents when the MTS notifies it. When the Society Viewer shows the dynamic behavior of the multi-agent systems running on CAPNET, it also allows the developers to trace and debug the system's execution, making possible to detect errors in applications logic and resolve more complex tasks, like load balance in agents work.

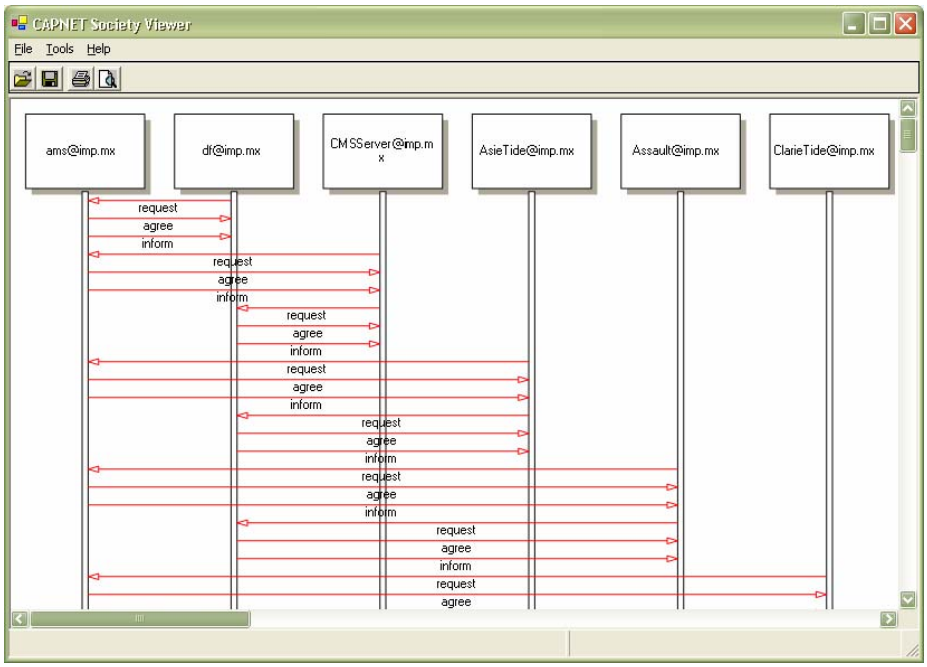


Figure 8. CAPNET Society Viewer

Society Viewer tool connects to AMS and MTS by subscribing to notification of remote events from those services (using .NET Remoting again), registering delegate methods which are invocated when those events are fired. Each event fired by AMS is presented in Society Viewer using the information related to the agent whose state is changing; specifically, the new state of its lifecycle and an object AMSAgentDescription with complete agent description. In the same way, the MTS fire an event as a result of message delivering request from a source agent; then the Society Viewer catches this event through its delegate method, receiving an object TransportMessage and showing graphically this message, drawing a link between source and target agents. By clicking this link, a Message Viewer tool is invoked displaying all the fields from the ACLMessage object (Fig. 9).

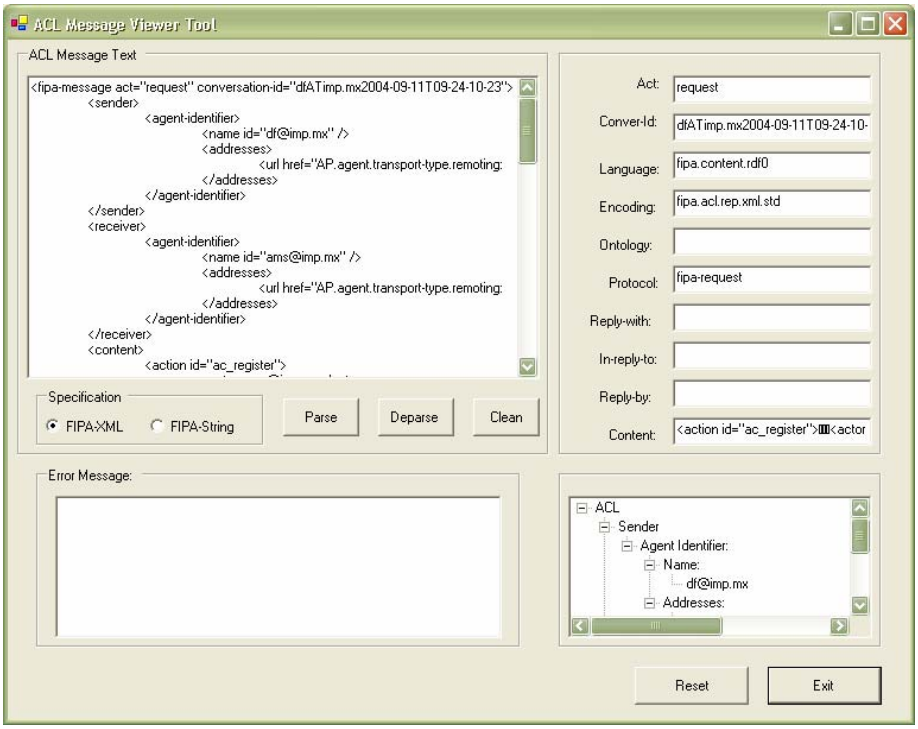


Figure 9. ACL Message Viewer

Communication Infrastructure Configurator

The Communication Infrastructure Configurator tool (Fig. 10) enables us to specify parameters for MTS, and administer the execution of this service. The MTS is hosted by a Windows service, which can be started and stopped by the configurator tool.

By using this tool we can read and change the URI address where the MTS listens from, in order to provide message exchange services for CAPNET agents. The tool also maintains a log for relevant events happening in MTS, particularly referred to successful and erroneous deliveries of messages. This log provides valuable information of the activity in the system and can be saved to a XML file to analyze it later.

In this tool CAPNET administrator are able to monitor system performance using the information provided in the statistics section, where performance graphs are generated in real-time to reflect the activity of the agents in the AP.

Finally, this configurator tool provides the functionality needed to send ping messages to any agent registered in the MTS; a CAPNET administrator can use this functionality to detect points of failure in the network where multi-agent systems are running and even to perform debugging processes by determining which agents are not reachable at any given time.

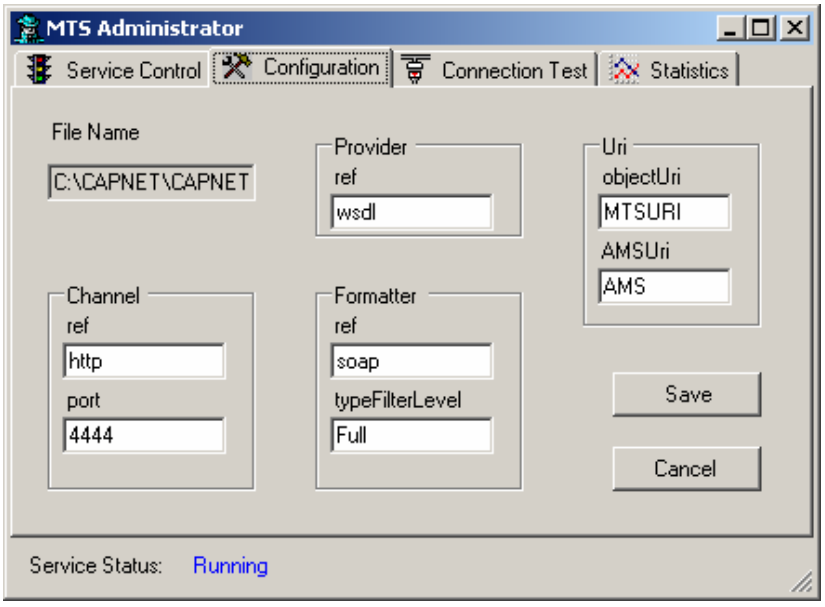


Figure 10. Communication infrastructure configurator tool

6 AGENT IMPLEMENTATION

The construction of a set of basic elements that constitute the internal architecture of the agents in a FIPA compliant environment is crucial for application development. These elements provide the ways to develop agents, their unique identifiers, registry information, ACL message construction, message reception and handling, content codification and representation with different content languages, the platform services description and application agents services. In the following sections these elements are described in more details.

Basic Agent

In order to be successful or, at least, easy to use, the AP has to provide some mechanism for agents creation. Being the central element of the applications, a basic agent class takes advantage of the entire infrastructure in a transparent way. This class allows carrying out several tasks like i) instantiating the local MTS and registering its listener in it for incoming messages reception, ii) registering of AMS and DF services, and iii) processing of received messages. AMS and DF registration have been implemented using the conversation mechanism designed to control the messages exchange. This mechanism is described in the following section.

In order to support agents programming for the AP, an API is included, providing a set of core classes that define ontological objects in the domain of FIPA-compliant platforms. The main objective is to facilitate agent and MAS development by means of AP services, operating under the same principles of design and implementation. In turn,

this would allow accomplishing more general goals of interoperability among heterogenous agent systems.

Agents must perform certain activities in order to access registration data from other agents in the AP. That is why they need to use the core classes extensively. In this sense, AID class represents the unique identifier for each agent, containing information like its name and the set of transport addresses where it can be located and contacted by other agents. AMS Agent Description class, on the other hand, contains information related to each agent registered in the AMS; for example, its AID, owner and current state. In the same way, DF Agent Description class represents the description of agent services that can be published by DF service, including the AID, a list of services and the maximum lifetime for this description. The list of services contains a set of objects of Service Description class, which uniquely describes each service an agent provides.

As we mentioned before, agent communication based on message exchange lays in the core of this reference model. In this sense, the ACL Message class provides the functionality required for agents to be able to construct and send new messages, according to established forms for agent communication language.

Every core class of CAPNET is defined using key-value tuples (KVTs). This is very important for CAPNET extensibility; the key is the name for an attribute defined in FIPA namespace and the value, which can be a FIPA attribute, a constant or a specific type [Jas02]. This implementation encapsulates different sets of attributes used in the components of the AP, and also some attributes required for agent systems development.

By using KVTs, extensibility gets improved because the keys contained inside a defined namespace have a well known programmatic interpretation for their values. In the same way, flexibility gets improved through the use of values, which are linked to well known keys. This is particularly crucial for implementation abstraction, when the elements of AP are integrated with each other. APElement interface is implemented by each class that can be represented by a KVT set. In the following source code fragment APElement is shown:

```
public interface APElement
{
   object get( string key );
   void set( string key, object value );
   void setAll ( APElement other );
   bool containsKey(string key);
   ICollection keys();
   object remove( string key );
   void removeAll();
}
```

APBasicElement abstract class, which implements APElement, is the base for every class using the KVT model in CAPNET. In Fig. 11 a diagram for CAPNET core classes is shown.

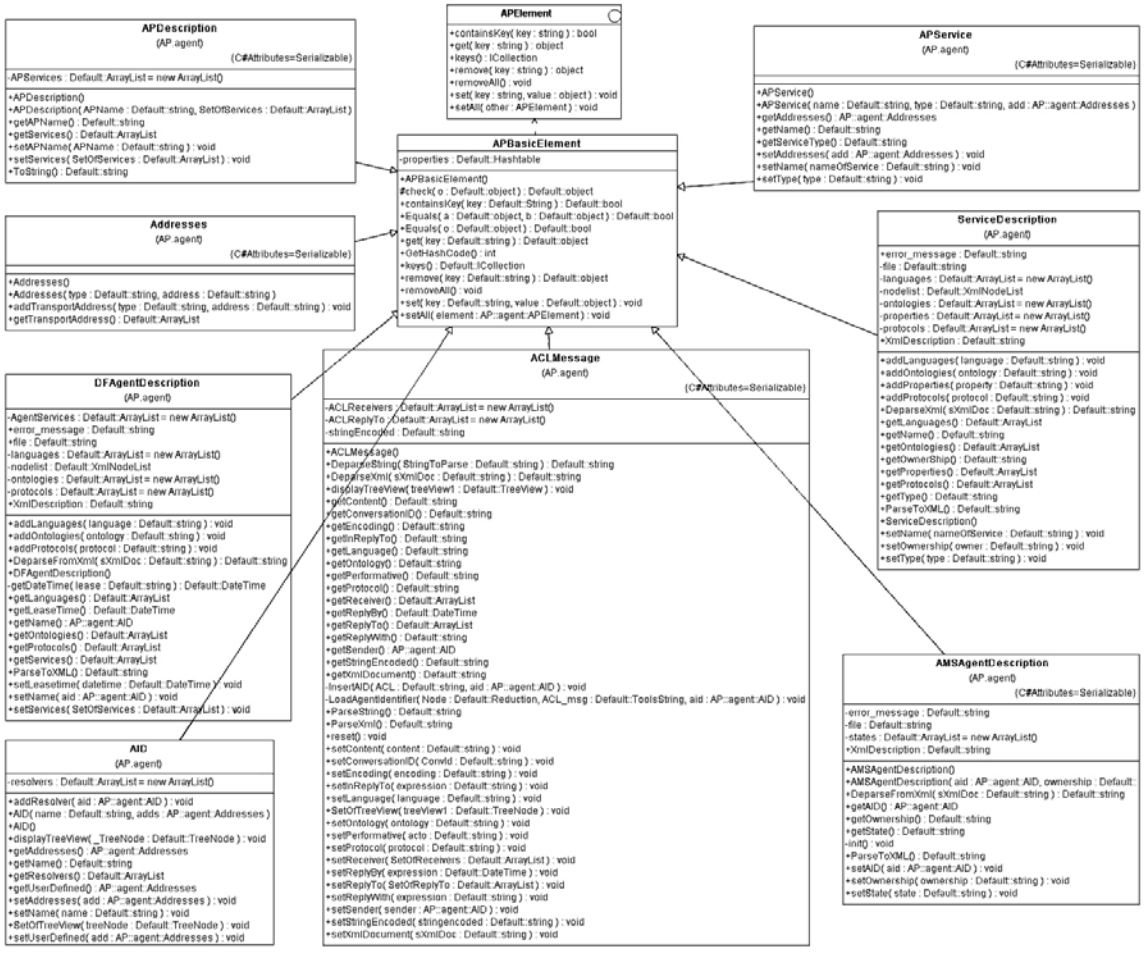


Figure 11. CAPNET Utility Classes Diagram.

Two mechanisms for message processing are developed: polling and callback. Polling is the mechanism that allows an agent to process messages in a synchronous way and is used by the conversations mechanism to control the predefined message sequence in an interaction protocol. Callback works similar to the MTS message delivery mechanism. Events are declared in each agent to process each one of the message types or a pre-established conversation. The particular mechanism to be used is dynamically determined according to the attributes of the message (conversation-id and protocol).

Conversation Manager

A conversation manager is an internal component of each agent linked to its communication capacity. Conversations are important to facilitate the interaction between different agents and to carry out some tasks within a multi-agent environment. Besides facilitating the interaction, a conversation manager allows an agent to control its course of action on the basis of the results that are obtained during conversations. A conversation manager allows an agent i) to add new conversations required during the communication process, ii) to add agent interaction protocols (AIP) that can be used to control the

message sequence in a conversation, and iii) in general, to offer access to the completion state and results of a conversation.

A conversation is added when a message establishes a conversation and AIP identifiers. A new conversation is dynamically created determining (by means of the .NET reflection mechanism) the AIP from the CAPNET library. It is important to mention that an AIP can have several implementations. Each new conversation is handled in a new execution thread in such a way that an agent can carry out parallel interaction through simultaneous conversations.

We have defined a set of classes and interfaces that helps to create conversations and AIPs in a standard way. At the moment, two interfaces for the synchronous and asynchronous conversations are implemented. When a new interface is created, it must inherit from a conversation class to establish its attributes (conversation-identifier, AIP class to be used in conversation control and delay time-out between messages) and must implement some of the interfaces of conversation type, that mainly serve to give access at run time to an AIP's concrete implementation.

Agent Communication and Content Languages

In order to provide communication functionality, FIPA-ACL is implemented [FIPA00061]. XML is used as the standard encoding for messages. The content of the ACL messages is represented in a content language allowing agents to obtain and handle objects from the agent's knowledge base. It enables knowledge interchange and handling between heterogeneous applications and guarantees high interoperability and autonomy degrees. For CAPNET message coding two languages are implemented: FIPA-RDF0 (using XML representation with validation through schemes) and FIPA-SL (using the string representation scheme, a grammar and parser for construction and validation of these content objects).

Dynamic action invocation

Agents can carry out actions on behalf of the others. Because the action requests are codified in a semantic language (a text format) and are not obtained directly through method invocation, agents extract the requested action and achieve dynamic invocation by means of .NET reflection mechanism. The content language allows expressing an action, its internal results and its arguments. The agent itself determines how to extract the action and its attributes to carry out the invocation.

Currently, CAPNET includes a mechanism for action representation in XML. By using it, agents can request, inside the message content, action execution by other agents.

Using the basic agent template, agents can obtain requested actions, and perform the search for these actions in their list of implemented services. Eventually an agent will perform the requested action, if it is possible.

Currently the agent determines if it is able to perform the requested action based on its current state, which is managed by a deterministic finite automaton.

7 EXPERIMENTAL RESULTS

CAPNET Testing

The communication capacities of the platform were stress tested with a custom made benchmark application that involved the creation of 100 agents distributed in 10 hosts in a single segment of a 10Mbps local area network. The benchmark measured the time it took for each agent to send one message to each other agent, and one to itself sending a total of 10,000 XML encoded messages with a length of 300 bytes/each. The results obtained showed that the full load of 10,000 messages took a variable time of 83 to 108 seconds to be delivered in 40 different simulations at different times (the network infrastructure was not exclusive for this purpose and had peek use hours).

Prototype development and testing

A prototype called Contingency Management Simulation System (CMSS) has been developed, which implements a multi-agent system for optimization of decision making in contingency situations requiring personnel evacuation from the petroleum platforms in the Gulf of Mexico [Sheremetov04a]. This marine zone has many weather disasters (like hurricanes), so a reliable system for simulation of personnel evacuation from the platforms is required.

The main goal of CMSS is to define and simulate an evacuation plan for personnel of petroleum platforms, taking into account the following parameters:

- Data about the predicted route for a hurricane, the weather and sea conditions.
- Number of petroleum platforms and the number of persons in each one.
- The number, type and capacity of transport vehicles available to realize the evacuation. Those vehicles include helicopters and boats.

The evacuation is usually performed in three phases:

1. First phase. The hurricane alert is announced, though the hurricane is far enough from the platforms. The evacuation of personnel is started using only boats with large capacity. It should be mentioned that on the petroleum platforms a minimum personnel is always required, even in an emergency situation. So, less mission critical workers for platform operation are evacuated first.
2. Second phase. The hurricane is closer to the zone of platforms, so helicopters and boats must be sent to evacuate the personnel.
3. Third phase. Because of the critical proximity of the hurricane, only helicopters could be sent to the platforms. Mission critical personnel are left on the platforms.

The prototype communicates with a set of databases that contain data about vehicles and platforms using a Wrapper Agent (WA). The WA handles all the logic to access the data sources and hides it from the MAS. For the vehicles, relevant information is their type (boats or helicopter), maximum capacity for transporting people, maximum velocity,

operation cost per kilometer, and range. For platforms, important data is the number of persons and the percentage values indicating the people who must be evacuated at each phase of the plan. Additionally, a route for the hurricane is obtained from the prediction Agent, through a simple pre-defined linear equation for each phase (this information could eventually be obtained from weather forecast sites and web services). The algorithms construct the optimal routes for the vehicles.

An agent in the CAPNET platform represents each mobile entity in this application, even the hurricane. At the beginning, a simulation administrator agent (SAA) uses the described algorithm in order to generate service requests and to send them to the agents. These requests are managed by means of a contracting conversation protocol. Each vehicle agent determines if is able to perform the task depending of its possibilities to carry out the number of persons specified by the task, or to reach the specified platforms according of the required distance and time. This way, the exact evacuation plan for each phase is generated. Once an agent has committed to perform a part of the evacuation plan, it will periodically inform the SAA of its progress. The simulator uses a Tactical Display software developed in the MPI to show graphically all the active elements in the scenario and their current situation in real-time. Fig. 12 shows the user interface for this prototype.

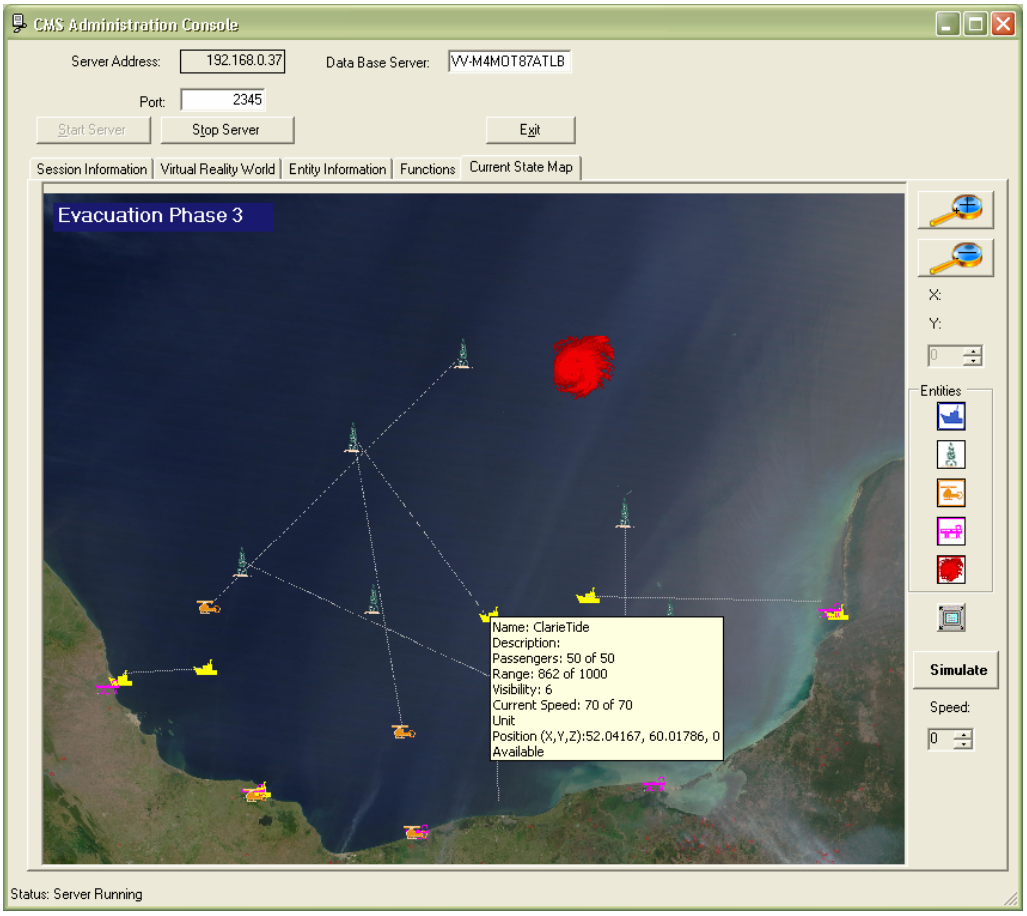
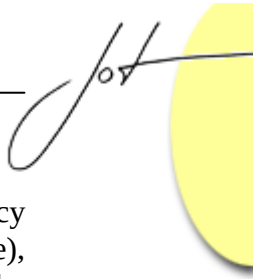


Figure 12. Contingency Management Simulation System interface and administration console



In this application, several key features of MAS were used: integration with legacy systems (to obtain information on available units and the personnel to evacuate), integration with tactical display software, where the SAA is responsible for updating the scenario with information it receives from the periodical reports from the agents in the system. By wrapping the service of hurricane trajectory prediction in an agent, the system can evolve towards the real life operation substituting the Prediction Agent only.

During the construction of the CMSS, CAPNET proved to be an appropriate infrastructure for the development of the system, the whole simulator was built by group of three developers in about a week. The first two days were used to implement the SAA (that contains the evacuation logistics algorithm) and the integration with the Tactical Display software, one day was used to program the agent that wrapped the legacy systems and databases that contained information about units and personnel and two more days were used to program the prediction agent and mobile units agents (vehicles and hurricane itself). The simulator has been deployed on several hosts in a local area network and behaved according to the specifications.

8 DISCUSSION

In this section we outline and discuss the main technical features of the CAPNET implementation over the CAPNET and stress the advantages of this implementation over the .NET framework.

As shown, the architecture proposed for the core of the CAPNET is divided into four main blocks: application agents, agent administration and directory services, security and message transport services. Since agent communication plays a key role in social interaction, a great effort was invested in order to make it very extensible and interoperable. To achieve this, the low level transport mechanisms (TM) were isolated from the core of the MTS and integrated into it using the “factory pattern” that enables the possibility to have several concrete implementations of TMs for different protocols or communication techniques. Another advantage of using this design pattern is that it will easily accept the implementation of load balancing techniques in the future.

The implementation of the CAPNET required the extensive use of remoting for different tasks such as agent communication and administration. The use of remote delegates allowed the agents to subscribe to the events published by the MTS, and also enabled the administrative tools to get instant notifications of the changes in the state of the platform services.

All the remoteable AP components involved in the message transport mechanisms (MTS and Remoting TM) and directory services (AMS and DF) were implemented as Server Activated Singleton Objects and hosted in Windows Services. This particular implementation has the advantage of high availability of these components and leads to the clusterization of these services to increase the reliability of the AP as proved by our latest experiments on the CAPNET clusterization using Microsoft™ Cluster Services and Network Load Balancing.

The use of XML as the standard encoding for messages has several advantages. Some of them are: native support provided by the .NET framework for managing XML documents, easy integration with the available commercial and industrial applications and the natural integration to the semantic languages.

To take advantage of the features of the development platform, the CAPNET services report their state to the operating system via performance counters and the event log. This enables CAPNET monitoring and logging the platform behavior using standard Windows utilities (such as "Performance Log and Alerts") and also to raise events and fire alerts if desired, based on the platform's state.

In order to make CAPNET compatible with the .NET Compact Framework (CF) and to be able to deploy administration tools and agent systems in mobile devices, alternative mechanisms to remoting for communications had to be implemented. These mechanisms included the use of Web Services, easily accessible for applications written for the CF and capable of bridging to the main CAPNET infrastructure.

A mechanism to provide agent services to non-agent clients is provided by means of a bridge to web-services. Using this bridge, external applications can request services performed by agents using web service invocations. This bridge also allows agents to consume web services without having to deal with direct invocation.

Comparison with other APs

The current development can be compared to some products already available, most of which are based on Java language and its associated technologies. One of these APs and probably the most widely used today is JADE. It has all the advantages of Java based systems, such as platform independence, which is not yet fully available for the .NET framework. This leads to the fact that CAPNET can only be used on Windows™ based platforms and devices compatible with them.

One of the main features of JADE is that it works as a container for the agents, that is, the agents live inside a container (that can be distributed across several hosts) and they have to operate inside it at all times, in CAPNET application agents are designed to be integrated with the application programs (hence distributed as well), and they do not require a container to live in. The only case where agent containers will be used in CAPNET is to support agent mobility (agent traversing hosts carrying along its state and code) between hosts, which is under development.

Several prototype MAS are under development using CAPNET, which will help to test its functionality for concrete applications. Most of these prototypes were earlier implemented using JADE or the first version of the CAP and include supply chain configuration, secure desktop and contingency management. Details can be found in [Sheremetov04a, Smirnov04].

While having prototype applications implemented in both platforms, extensive testing and performance comparison can be carried out. The main idea is to measure how easy is to learn and to use the agent programming paradigm specific for each platform and also to compare their scalability and performance capabilities.

9 CONCLUSIONS

In this paper we have presented an Agent Platform named CAPNET that constitutes an excellent example of a distributed system built on top of the .NET Framework. The primary goal of the CAPNET is to enable the developers to construct distributed, flexible and open systems in a multi-agent fashion using AP as the basic infrastructure for communication and administration of the components. As shown in the paper, the .NET framework provided an infrastructure that allowed the construction of the AP that is easily integrated with enterprise applications and server products.

A very important feature being implemented for the next version of CAPNET, is the support for agent mobility, that will enable agents to traverse hosts to perform their tasks. This feature will make extensive use of .NET remoting and serialization in order to move the code and execution state of agents between hosts. This leads to the implementation of agent containers that will transport agents and provide a secure environment for their execution.

ACKNOWLEDGMENTS

Support for this research work has been provided by the Mexican Petroleum Institute within the project D.00006 "Distributed Intelligent Computing" and the Post-Graduate Studies program. The authors would like to thank Microsoft™ México, especially Luis Daniel Soto and Felipe Lemaitre for their valuable support in the realization of this work.

Special thanks to all the MPI colleagues for the helpful discussions on the CAPNET architecture and to Ana Luisa Hernandez for her contribution to the development of the CAPNET message encoding mechanism.

REFERENCES

[Bernstein96] Bernstein, Philip A. “Middleware: A Model for Distributed Services”. *Communications of the ACM* 39, 2 (February 1996): 86-97.

[FIPA00023] FIPA Agent Management Specification. Foundation for Intelligent Physical Agents, 2002. http://www.fipa.org/specs/fipa00023/

[FIPA00061] FIPA FIPA ACL Message Structure Specification. Foundation for Intelligent Physical Agents, 2002. http://www.fipa.org/specs/fipa00061

[JADE00] JADE Programmer’s guide, Bellifemine F., Caire G., Trucco T., Rimassa G. JADE 2.5.

[Jas02] Java agent Services, JAS http://www.java-agent.org/, http://www.java-agent.org/JAS_Intro.htm, JAS Specification PR 1.13 05032002.pdf

[Luck03] Luck, M., McBurney, P. and Preist, C. *Agent Technology: Enabling Next Generation Computing*. AgentLink, 2003. ISBN 0854 327886

[Nortel99] Nortel Networks FIPA-OS http://www.nortelnetworks.com/products/announcements/fipa/

[Pallickara03] Pallickara, S., Fox J., Yin J., Gunduz G., Liu H., Uyar A., Varank M., "A Transport Framework for Distributed Brokering Systems". *Proc. PDPTA'03*.Volume II pp 772-778.

[Plummer00] Plummer D., Smith D., Web Services and Software E-Services: "What's in a Name? Application Integration and Middleware Strategies", *Research Note COM-12-0101*, Gartner Group, 2000.

[Rammer01] Rammer I. *Advanced .NET Remoting*, 1st edition, Apress, ISBN: 1-59059-025-2

[Santana03] Santana G., Sheremetov B. L., and Contreras M. "Agent Platform Security Architecture, Computer Network Security", *Proceedings of the MMM ACNS 2003, St. Petersburg, Russia, September 2003*, V. Gorodetsky, L. Popyack V. Skormin (Eds.), LNCS 2776, Springer Verlag, 2003, pp.457-460

[Sheremetov01] Sheremetov, L. & Contreras, M., "Component Agent Platform", in *Proc of the Second International Workshop of Central and Eastern Europe on Multi-Agent Systems, CEEMAS'01*, Cracow, Poland, 26-29 of September, 2001, pp. 395-402.

[Sheremetov03] Sheremetov L., Contreras M., Chi M., Germán E., Alvarado M., "Towards the Enterprises Information Infrastructure Based on Components and Agent", in *Proc. of the 5th International Conference On Enterprise Information Systems*, Angers, France, 23-26 April, O. Camp, J. Filipe, S. Hammoudi and M. Piattini (Eds.) Escola Superior de Tecnologia do Instituto Politécnico de Setúbal, Portugal, 2003. pp. 340-347.

[Sheremetov04a] Sheremetov L., Contreras M. and Valencia C. "Intelligent Multi-Agent Support for the Contingency Management System", *Int. Journal of Expert Systems with Applications (Special Issue)*, Pergamon Press, 26(1): 57-71, 2004.

[Sheremetov04b] Sheremetov L., Contreras M. and Smirnov A. "Implementation of an Ontology Sharing Mechanism for Multiagent Systems based on Web Services". J. Favela et al. (Eds.): *AWIC 2004*, LNAI 3034, pp. 54-63, 2004. Springer-Verlag Berlin Heidelberg 2004.

[Smirnov04] Smirnov, A. V. Sheremetov, L. B. Chilov, N. Romero-Cortes, J., "Soft-computing Technologies for Configuration of Cooperative Supply Chain".

Int. Journal Applied Soft Computing. Elsevier Science, Vol. 4(1):87-107, 2004.

[ZEUS00] ZEUS: Nader A., Thompson S. "A Toolkit for Building Multi-Agent Systems", in *Proceedings of fifth annual Embracing Complexity conference*, Paris April 2000

GLOSSARY

ACL	Agent Communication Language
AID	Agent Identifiers
AIP	Agent Interaction Protocols
AMS	Agent Management System
AP	Agent Platform
API	Application Programming Interfaces
APSM	Agent Platform Security Manager
AS	Agent Services
CF	Compact Framework
CMSS	Contingency Management Simulation System
DF	Directory Facilitator
FIPA	Foundation for Intelligent Physical Agents
MPI	Mexican Petroleum Institute
KVT	Key-Value Tuple
MAS	Multi-Agent Systems
MTA	Message Transport Adapter
MTPF	Message Transport Protocol Factory
MTS	Message Transport System
SAA	Simulation Administrator Agent
TM	Transport Manager
WA	Wrapper Agent
WS	Web Services
XML	eXtensible Markup Language

About the authors

Miguel Contreras obtained in 2002 his MS degree in Computer Science at the National Technical University of Mexico developing as a Masters Thesis the original CAP agent Platform. Now he is a PhD student of the Post-Graduate Studies program at the Mexican Petroleum Institute. His current research includes Multi-Agent Systems, Interaction Protocols, Service Oriented Architectures and Distributed Systems. E-Mail: mcontrer@imp.mx

Ernesto Germán obtained in 2002 his MS degree in Computer Science at the National Technical University of Mexico. Now he is a PhD student of the Post-Graduate Studies program at the Mexican Petroleum Institute. His current research includes Multi-Agent Systems, Knowledge Representation, Expert Systems, Social Behaviors, Artificial Life and Distributed Systems. E-Mail: egerman@imp.mx

Manuel Chi obtained in 2003 his MS degree in Computer Science at the National Technical University of Mexico. Now he is a research assistant of the Distributed Intelligent Systems Group and a part-time professor of the Post-Graduate Studies program of the Mexican Petroleum Institute. His current research includes Service Oriented Architectures, Distributed Systems, Software Security and Multi-Agent Systems. E-Mail: machi@imp.mx

Leonid Sheremetov obtained his Ph.D. in computer science from St. Petersburg Institute for Informatics and Automation of the Russian Academy of Sciences (SPIIRAS). His research interests include Multi-Agent Systems, Decision Support Systems, Expert Systems, and Web-Based Learning. He is a Principal Investigator of the Research Program on Applied Mathematics and Computing of the Mexican Petroleum Institute (PIMAyC MPI) where he leads the Distributed Intelligent Systems Group, a part-time professor of the Artificial Intelligence Laboratory of the Centre for Computing Research of the National Technical University (CIC-IPN), Mexico, and member of the National System of Researchers of Mexico. He is also member of the Technical Committee "Artificial Intelligence and Expert Systems" of the IASTED (The International Association of Science and Technology for Development). E-Mail: sher@imp.mx