Design and implementation of a FIPA compliant Agent Platform in .NET

Miguel Contreras Ernesto Germán Manuel Chi Leonid Sheremetov
Instituto Mexicano del Petróleo
Eje Central Lázaro Cárdenas #152
Col. San Bartolo Atepehuacán.
México, D.F. 07730, México
mcontrer{egerman, machi, sher}@imp.mx

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.

Keywords

Distributed Computing, Multi-Agent Systems , FIPA, Agent Platform, .NET Framework.

1. INTRODUCTION

Agent-based computing has the potential to significantly improve the theory and the practice of modeling, designing, and implementing complex distributed computer systems [Jen00, Woo99]. Autonomous agents are entities that can complete their objectives while situated in a dynamic and uncertain environment, that can engage in rich, high-level social interactions, and that can operate within flexible organizational structures and systems.

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 [Fip00].

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.

.NET Technologies'2004 *workshop proceedings,*
ISBN 80-903100-4-4

In the last few years, several APs have been developed, and special attention has been 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 [Jad00], FIPA-OS [Nor99] and Zeus [Zeu00] just to mention a few of them. One exception to this trend was the original CAP [She01] 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 a great number 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 on the .NET Framework.

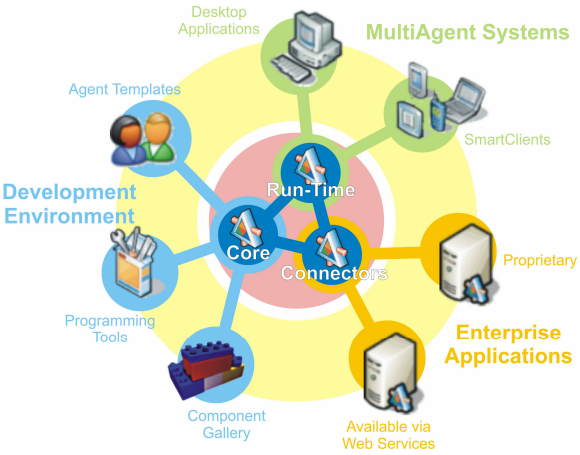

Figure 1. MAS development and deployment in CAPNET

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. In section 3, the CAPNET architecture is defined. In section 4, the most important implementation details are addressed and technically described. Finally, the main features and advantages of this work are discussed, along with some future work.

2. AUTONOMOUS AGENTS

An agent is a computational entity that interacts with one or more software counterparts or real-world systems [Fra96]. Unlike traditional computer programs, agents exhibit the following capabilities to various degrees: autonomy, reactiveness, proactive-ness, mobility, intelligent behavior and social abilities.

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.

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 [She03]. 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 [Fip00]. The reference model considers an AP as a set of four components: 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).

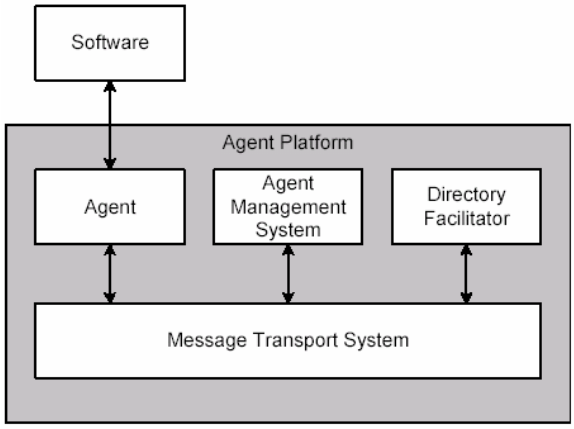

Figure 2. Agent Management Reference Model [Fip00].

The functionality of the main components of the FIPA spec are: the AMS provides white-page and life-cycle service, maintaining a directory of agent identifiers (AID) and agent state. The DF is the agent who provides the default yellow page service. The MTS is the software component controlling all the message exchange within the platform, including messages to/from remote platforms.

The 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 a great number of agent applications, 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.

3. CAPNET ARCHITECTURE

The main objectives of CAPNET design are 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 CAPNET is shown in fig. 3. In this architecture four main parts are shown: i) application agents, ii) agent management and directory facilitator services, iii) built-in security services, and iv) message transport system and connectivity techniques. The message transport mechanisms are the core of the 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.

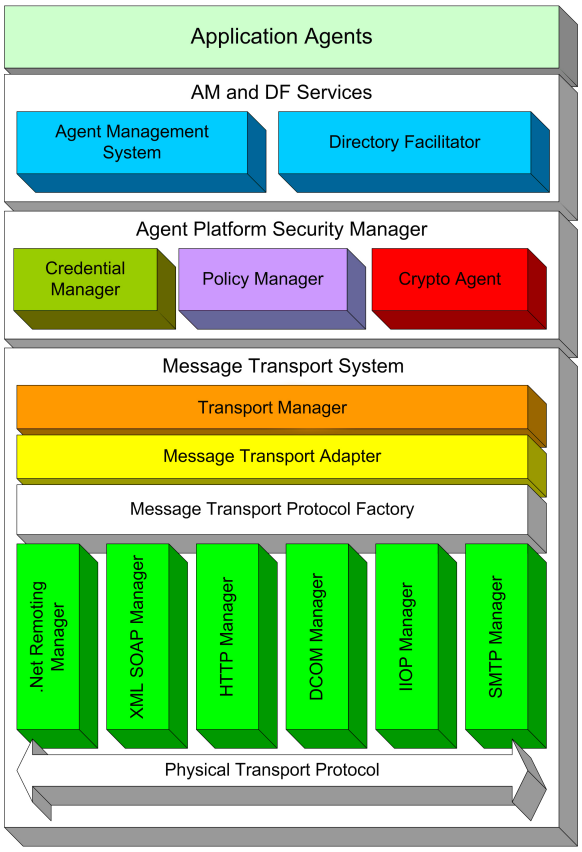

Figure 3. CAPNET architecture

The services of transport, delivery and reception of messages represent a central point within the CAPNET platform. In CAPNET, an agent communicates with others using a service provided by the 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 (MOM) [Ber96] or publish/subscribe architectures [Pal03].

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 or not. If some adressee 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 it waits 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 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 is out of the scope of this paper, the details on its architecture and implementation can be found in [San03].

4. IMPLEMENTATION DETAILS

4.1 The Message Transport Mechanism

The MTS was implemented as a singleton remoteable object [Ram01] 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.

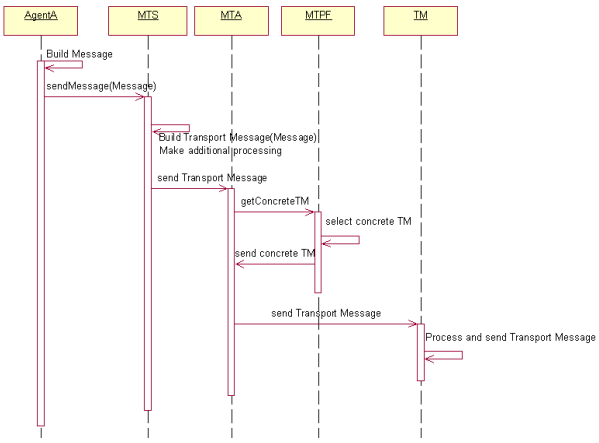

Figure 4. Sequence for message sending.

When a message is received by a TM, 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.

4.2 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 (Tier 1 in the figure) 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 ACL (Agent Communication Language) *request* message specifying the action and its parameters, and launching the corresponding services.

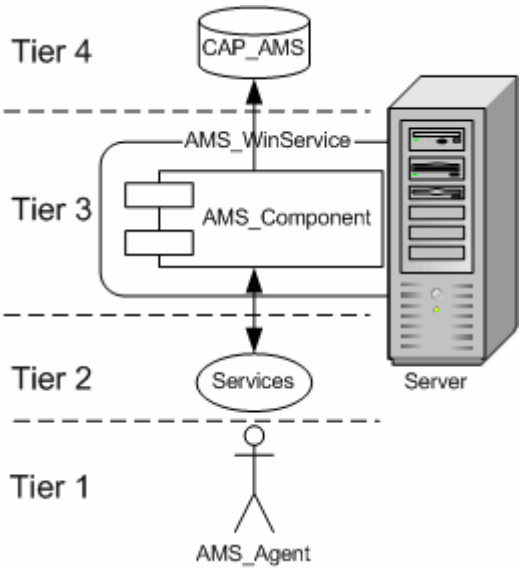

Figure 5. AMS Service implementation.

The application-logic tier (Tier 2) is implemented in the services (actions) that the AMS_Agent is able to perform. The actions that this agent is able to perform are: registration, deregistration and modification of agent descriptions in the white page directory, and search of agent descriptions.

The data access tier (Tier 3) 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 (Tier 4) is implemented in a relational database using Microsoft SQL Server as the DBMS that stores Agent Descriptions of all the registered agents.

4.3 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.

4.3.1 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 provided. This API includes classes to construct the unique agent identifier (AID), DF service descriptions (ServiceDescription), the local platform description (APDescription), AMS agent descriptions (AMSAgentDescription), DF agent descriptions (DFAgentDescription), etc. Figure 6 (in appendix) shows the diagram of CAPNET utility classes.

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).

4.3.2 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 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 their 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 in run time to an AIP's concrete implementation.

4.3.3 Agent Communication and Content Languages

In order to provide communication functionality, FIPA-ACL is implemented [Fip00]. 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).

4.3.4 Dynamic action invocation

Agents can carry out actions in favor of others. Because the action requests are codified in a sort of 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.

4.4 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.

One of the most important tools is the AMS administrator (fig. 7), because it allows monitoring of agents registered at the AP, their state and properties.

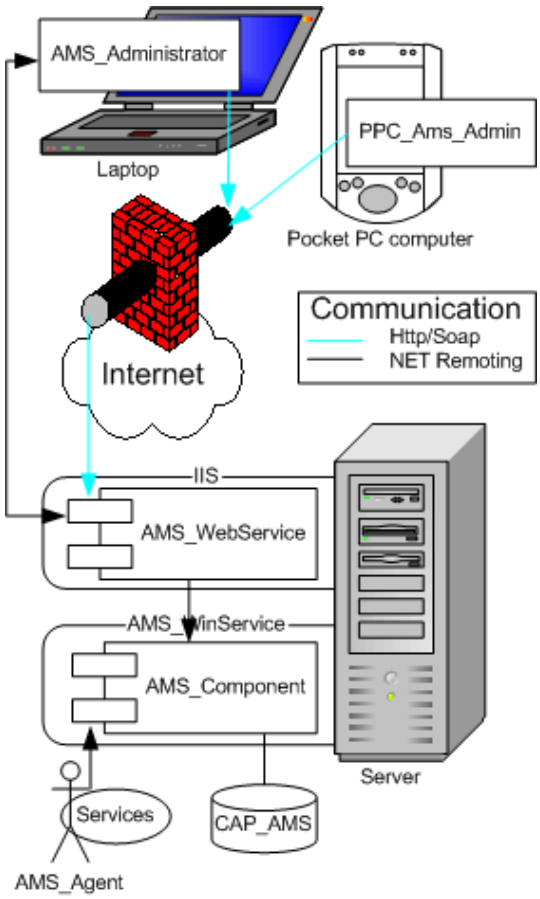

Figure 7. AMS Administration architecture.

This tool is designed to connect directly to the AMS_Componet using the IAMSAdmin interface via .NET Remoting. 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. 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 occuring in the AMS as a result of registration, deregistration or modification of agent descriptions in the AMS.

5. DISCUSSION AND 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. One particular feature of this platform is that its primary goal is to enable the developers to construct another kind of distributed, flexible and open systems (MAS) using it as the basic infrastructure for communication and administration of the elements that will be part of them.

The architecture proposed for the core of the CAPNET is divided into three main blocks: directory services, security and message transport services. Along with those elements a set of tools for the administration, monitoring and development of MAS for the platform have been constructed.

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 will eventually lead to the clusterization of these services to increase the reliability of the AP in future versions.

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

In order 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.

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.

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 LAN. 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).

A very important feature to be implemented in future versions of CAPNET, will be the support for agent mobility, that will enable the agents to traverse hosts to perform their tasks.

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, contingency management (fig. 8). Details can be found in [She04, Smi04].

6. ACKNOWLEDGMENTS

Support for this research work has been provided by the Mexican Petroleum Institute within the project D.00006 "Distributed Intelligent Computing".

The authors would like to thank Microsoft™ México, especially Luis Daniel Soto and Felipe Lemaitre for their support for the realization of this work and also to all their colleagues for the helpful advices on the CAPNET architecture and Ana Luisa Hernandez for her contribution to the development of the CAPNET message encoding mechanism.

7. REFERENCES

[Ber96] Bernstein, Philip A. "Middleware: A Model for Distributed Services." Communications of the ACM 39, 2 (February 1996): 86-97.

[Fip00] FIPA Specifications. http://www.fipa.org/specs/

[Fra96] Franklin, S. and Graesser, A., “Is it an Agent, or just a Program? A Taxonomy for Autonomous Agents”, Proc. ATAL’96, Springer-Verlag, Berlin, Germany, 1996.

[Jad00] JADE Programmer’s guide, Bellifemine F., Caire G., Trucco T., Rimassa G. JADE 2.5.

[Jen00] Jennings N. R. On agent-based software engineering. Artificial Intelligence, 117, 277–296.

[Nor99] Nortel Networks FIPA-OS http://www.nortelnetworks.com/products/cements/

[Pal03] 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.

[Ram01] Rammer I. Advanced .NET Remoting, 1st edition, Apress, ISBN: 1-59059-025-2

[San03] 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

[She01] 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.

[She03] 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.

[She04] Sheremetov L., Contreras M. and Valencia C. Intelligent Multi-Agent Support for the Contingency Management System. International Journal of Expert Systems with Applications (Special Issue), Pergamon Press, 26(1): 57-71, 2004.

[Smi04] 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.

[Woo99] Wooldridge M. "Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence", G. Weiss (Ed.), MIT Press, Cambridge, MA, 1999.

[Zeu00] ZEUS: Nader A., Thompson S. A Toolkit for Building Multi-Agent Systems Proceedings of fifth annual Embracing Complexity conference, Paris April 2000.

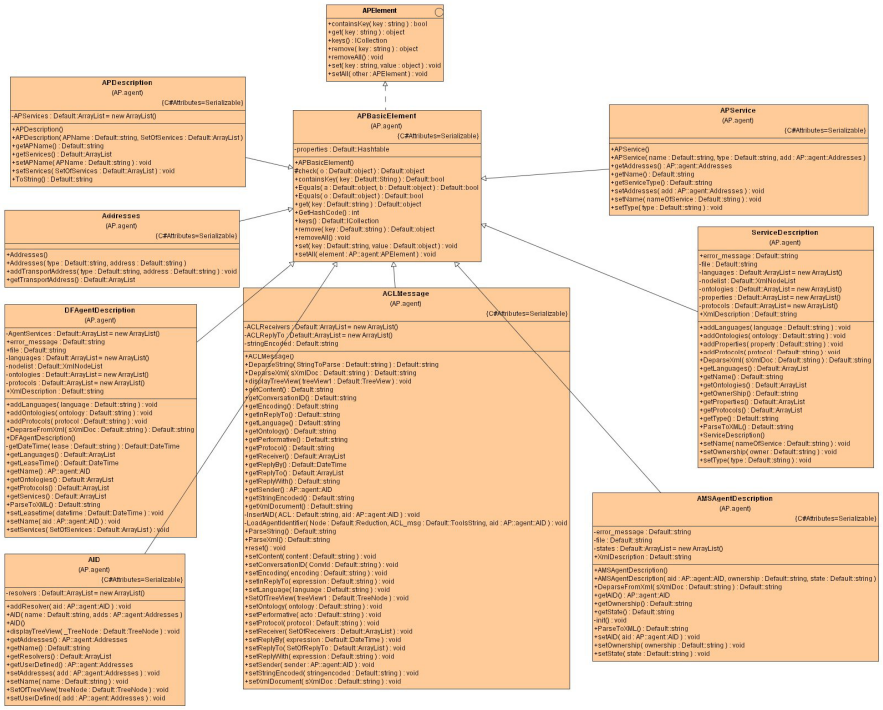

Figure 6. CAPNET Utility Class Diagram.

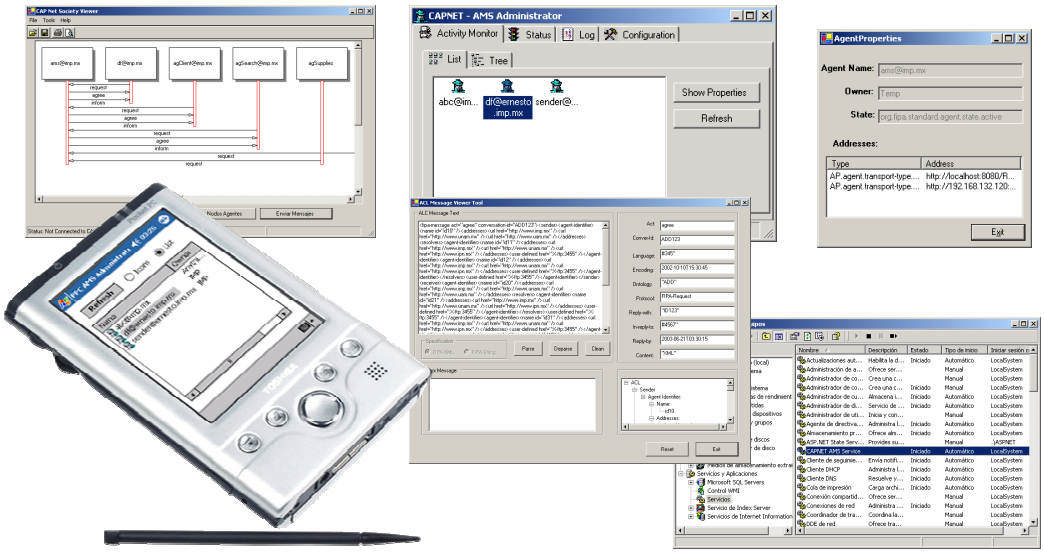

Figure 8. CAPNET Tools (Contingency Management System application).