Australian Journal of Basic and Applied Sciences, 4(5): 844-850, 2010
ISSN 1991-8178

Decentralized and Fault-tolerant FIPA-compliant Agent Framework Based on .Net

[1]Ghulam Ali, [1]Noor Ahmed Shaikh, [2]Madad Ali Shah, [3]Zubair A Shaikh

[1]Department of Computer Science, Shah Abdul Latif University, Khairpur
[2]Sukkur Institute of Business Administration
[3]National University of Computer & Emerging Sciences, Karachi

Abstract: Agent Oriented Programming & Software Engineering is being focused to design distributed software systems. More than 100 Agent toolkits for commercial & research purpose are available. Most of the toolkits are based on Java, whereas the focus on .NET platform has recently been started to develop environment to run multi-agent systems. CAPNET was the first .NET based agent platform developed in 2004 where agents could be developed on the application layer to solve distributed problems. The most and the common problem in agent platforms including CAPNET lack the fault tolerance and unavailability of decentralization. In this paper we propose and implement a distributed, decentralized, and fault tolerant agent framework based on the .NET to solve distributed and complex problems. The proposed Agent Platform ACENET (Agent Collaborative Environment based on .NET) follows FIPA specification and works under decentralized architecture.

Key words: Agent Toolkits, Agent-oriented programming, .NET platform, FIPA, Distributed Environment

INTRODUCTION

Agent-oriented programming paradigm is the appropriate practice to model, design and implement the complex and distributed software systems (Jennings, N.R., 2001).

Agent is an encapsulated computer system that is situated in some environment, and that is capable of flexible, autonomous action in that environment in order to meet its design objectives (Wooldridge, M., 1997). It is viewed as abstractions with intelligent behavior to cope up with complexity of system design. Agent Communication Languages, Interaction Protocols, Agent Markup Languages are designed to provide high-level abstraction.

When a software agent as alone is unable to solve a problem because of their limited capability or knowledge then multiple software agents form a distributed loosely coupled network and work together to solve the problem, such a community is known as Multi-Agent System (MAS). In MAS, task is decomposed, subtasks are divided among agents, and agents interact with each other. Because agents are of the different capabilities, therefore there is need of cooperation, coordination, negotiation, mediation by a neutral agent, etc (Zhong Zhang and V. Honavar, 2004).

The justification of the Multi-agent System is that (a) each agent has incomplete information or capabilities for solving the problem; (b) there is no system global control; (c) data are decentralized; and (d) computation is asynchronous (Katia, P. Sycara, 2000).

The architecture that provides implementation environment to Multiagent Systems for management of the agents, coordination, communication, negotiation, etc is called Agent Platform or toolkit. The agent toolkits provide agent builders with a sufficient level of abstraction to allow them to implement intelligent agents with desired attributes, features and rules.

In last few years there has been rapid development and deployment of Multiagent Systems implementation environment such as JADE, FIPA-OS, Zeus (AgentBuilder, 2000), etc. With the increasing number of these frameworks, two parallel agent development standards have evolved: FIPA and MASIF (Christos, 2002). The interoperability becomes easy due to evolve of the standards. FIPA's Agent Framework Reference Model is really an effective approach to develop agent framework, interoperable with other FIPA-compliant agent frameworks. MASIF intends to support interoperability among heterogeneous agent systems. It normally focuses on the migrations of the agents on different hosts (Frank Manola, 1998).

Corresponding Author: Ghulam Ali, Department of Computer Science, Shah Abdul Latif University, Khairpur
E-mail: ghulam.ali@salu.edu.pk,

In this paper we intend to focus on very recently emerging area of Agent Oriented Programming (AOP) on the top of the .NET framework. Agent Oriented Programming is a new evolving programming paradigm that touches programming abstraction concept of object-oriented patterns and latest research on Artificial Intelligence. It provides implementation methodology and defines relation between software agents and their execution environment. With the proposed framework the developers of this domain may design and implement execution environment for multiagent systems. In the next phase of the research the interoperability issues among different platforms will be addressed to discuss and implement agents executing on .NET platform and communication with the agents not running on the .NET platforms and other related issues (Shoham, Y., 2009). Agent oriented programming also supports distributed programming as the needs of distributed software systems are becoming complex day by day. It increases the demand of adaptive software to address complexity and distributed environment (Ricci, A., D. Weyns, 2008).

2 Foundation for Intelligent Physical Agents Specifications:

Foundation for Intelligent Physical Agents (FIPA) is standard to promote interoperable agent applications and agent systems. FIPA specifications only talk about the interfaces through which agents may communicate each other. It does not describe the implementation details. FIPA specifications are divided into five categories: Applications, Abstract architecture, Agent Communication, Agent Management and Agent Message Transport, are the five categories in which FIPA specifications are divided (FIPA Specifications, 2002).

The FIPA Reference Model considers an Agent Platform as a set of four components: Agents, Directory Facilitator (DF), Agent Management System (AMS), and Message Transport System (MTS). The DF and AMS support the management of the agents, while the MTS provides a message delivery service.

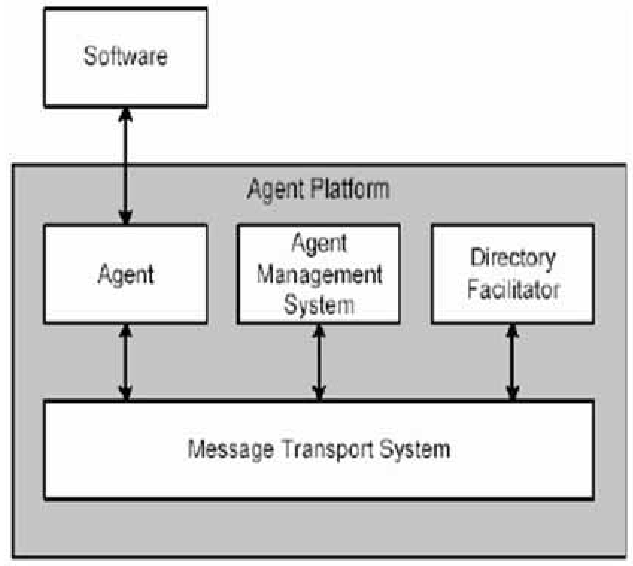


Fig. 1: FIPA Agent Platform Reference Model

FIPA standard does not talk about mobility as a mandatory requirement for FIPA compliant system (as shown in figure1). FIPA provides some guidelines for how implementation provider can provide support for mobility on their own. In addition the specification that talks about this is at preliminary stages.

3 ACENET Justification and Related Work:

A variety of Agent toolkits, academically and commercially, is available in the market. According to the AgentLink (European Coordination 2006) more than 100 toolkits have been developed. Four major categories of agent toolkits are identified: mobile agent toolkits, multi-agent toolkits, general-purpose toolkits and Internet agent toolkits (Henri Avancini1, 2000). Most of the FIPA and MASIF compliant Mobile Agent Toolkits and Multiagent System toolkits are developed in Java (Alexander Serenko, 2002).

Agent oriented programming has just started to focus on .NET framework to develop agent platforms. CAPNET (Component Agent Platform based on .NET) was the first .NET based FIPA compliant agent toolkit proposed in 2004 (Miguel Contreras, 2004). As already mentioned that most of the agent platforms are architecturally centralized and do not provide fault-tolerant model. We propose an agent platform, ACENET (Agent Collaborative Environment based on .NET), that is FIPA compliant that supports fault-tolerant and decentralized architecture. As .NET is the emerging technology therefore the proposed ACENET platform exploits the advanced features of the .NET. It also supports many programming languages. The Common Language Runtime (CLR) is an integral part of the .NET Framework that promises to let developers employ their cross-language skills in a single master architecture.

On the basis of the features that we have included in the framework we can say that the proposed platform is first .NET based fault-tolerant agent platform provides a decentralized architecture by implementing separate communication layer among different machines. The proposed platform is reliable and it can also be used as Client-Server architecture, if there is requirement of the application. All machines are independent to connect or disconnect the framework.

4 ACENET Architecture:

The ACENET has been designed so that the developers may create and integrate complete distributed agent applications. The architecture of ACENET (shown in figure 2) consists of four major components: a) Application Agents, b) Agent Management and Directory Facilitator Services, c) Message Transport Service and d) Agent Transport System.

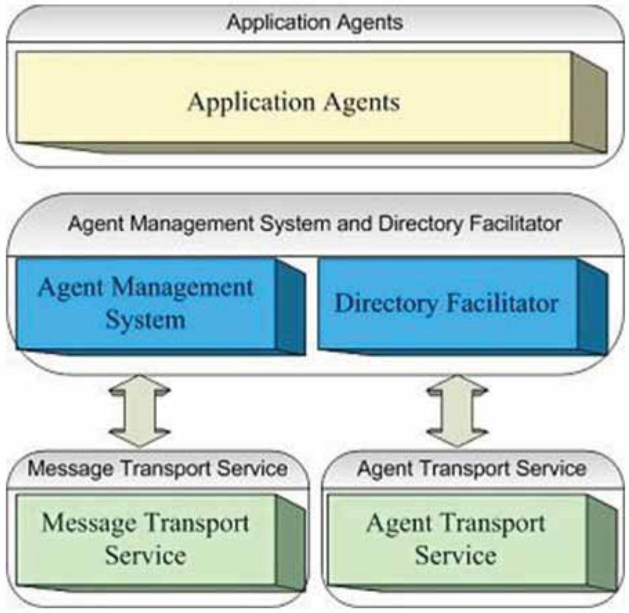


Fig. 2: ACENET Components

Application Agents interact with the Agent Management System (AMS) and Directory Facilitator (DF) through Agent Communication Language (ACL) that has been implemented according to the specification of FIPA standard. Application Agents utilize the services of Agent Transport Service (ATS) and Message Transport Service (MTS) through AMS. The interaction at any level, either AA to AMS or AMS to MTS, takes place through ACL. AMS coordinates with all the modules of the ACENET. It acts as a mediator to all the components and performs administrator level tasks. DF provides directory services to agents for finding other agents at runtime by specifying the search criteria such as AgentID, Agent Name, Service Type etc. MTS supports asynchronous messaging between agents within the same platform and between ACENET to ACENET. ATS provides message passing service to agents between ACENET to ACENET. In future the functionality of ATS will be extended for mobility as well.

Because of decentralized architecture (shown in figure3) if one node goes down all other nodes remain intact and function without failure of the framework and Applications Agents and their services will not be available on the framework. Instead of shut-downing of the entire system only agents of that machine will not be available to provide services. Therefore the architecture ensures high assurance using peer to peer architecture which brings scalability, fault tolerance and load balancing among distributed peers. In multiagent systems the property of Fault tolerance is implemented to avoid the failure of the agent platform. The ACENET multiagent system is based on peer to peer architecture therefore growth in numbers of hosts or software agents does not lead to the failure of framework. As discussed in related work, CAPNET is very simple work and does not follow FIPA specifications whereas our proposed framework, ACENET, is complete FIPA compliant agent platform and works on the property of Ignore-if-Fail strategy (Faci, N., O. Marin, 2008; Dragone, M., 2009).

5. Implementation Details:

One of the powerful tool available with .NET technology is .NET REMOTING. Most of the communication and management components have been implemented through this tool. The .NET technology also provides configuration tools for Agent Management System, Directory Facilitator and Message Transportation. The detailed implementation of the components is given below.

5.1 Application Agents:

Application agents are the main component of the framework. They are autonomous and have their own thread of execution. They can utilize the services provided by the platform such as directory service, message transport service, agent transport service etc. ACENET agents have the capability to interact with each other using FIPA interaction protocols. Messages are encoded in FIPA Agent Communication Language. Agents can choose either FIPA Semantic Language or any language that is embedded and encoded by the developer into the agent at the design time.

Agents can register/unregister/modify their services to Directory Facilitator. An agent can find other agents, within the same ACENET or remote ACENET, by querying the Directory Facilitator with the AgentID, AgentName, or ServiceType. That makes agents enable to create emerging virtual society of where they are able to find and interact each other for cooperation and collaboration to solve the problems.

An abstract agent class is provided with all the necessary functionality. The developer should derive its agent's class from the abstract agent class. Abstract functions are provided in order to notify the derived agent classes for essential events such as message notification, agent notification – before, during, and after migration notifications etc.

AMS takes agent assemblies and creates the instance of the agents by using the services of the reflection API provided by the .NET platform. Each agent is loaded in separate application domain in order to isolate agent code and data from other agents. The great advantage associated with this approach is that once agent assembly is loaded in an application domain it can be unloaded. The agents are accessed via proxy that is created by the .NET framework, hence providing a balance between isolation and security of code and data (Lavinal, E., 2009).

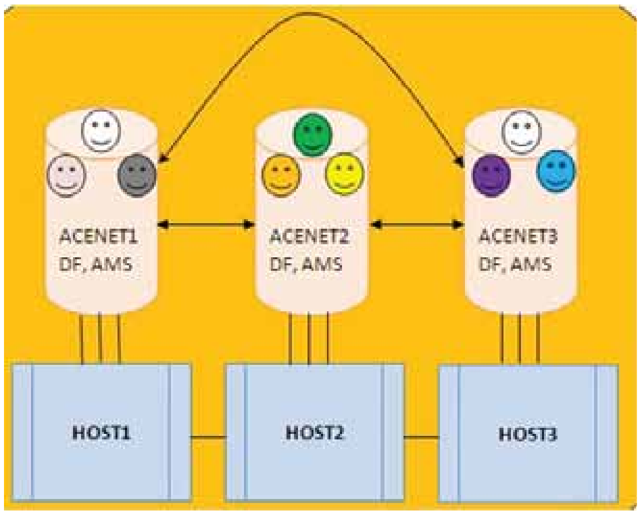


Fig. 3: ACENET Architecture

5.2 Life Cycle and Registration Management:

AMS handles all the communication, interaction and conversation mechanisms at the platform and agent level. AMS is able to handle tasks like: Creating Agent, Suspending Agent, Resuming Agent and Killing

Agent. When an agent is started, the AMS assigns a globally unique identifier, named as AgentID, to the agent and stores its instance into the local agent container. When an agent is instantiated, AMS initializes its state to START state. Agent can send a message to a remote host using the services of SendMessage of AMS.

DF provides the yellow page service to agents. Any agent can search any other agent by AgentID, Agent Name or Agent Service type. Only those agents could be searched that have registered to DF. An agent can register/unregister itself to DF and can also modify its registration details. DF can also be used to query for current and remote platform addresses respectively.

5.3 Interactions and Communications Among Agents:

To implement FIPA Interaction Protocols two sets of classes have been provided: InitiatorHandler and ResponderHandler. An agent, which initiates an interaction with other agent, should use InitiatorHandler class of that particular interaction protocol for initiating the interaction and to further handle the sending and receiving of the messages. If an agent takes part in an interaction with other agent as a responder, it should use ResponderHandler class of that interaction protocol of which the type of receiving message is, to respond the other agent. Each interaction can be distinguished from any other interaction because every interaction is assigned a globally unique identifier. This has an advantage in referring past interactions, if occurred, while interacting with other agents.

FIPA-ACL has been implemented using the string representation enabling the agents to communicate different speech acts with other agents. FIPA ACL messages are encoded in standard XML representation. 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. Using the string representation scheme, a grammar and a parser for FIPA SL has been implemented for construction and validation of the content of FIPA ACL messages (Lavinal, E., 2009).

5.4 Transport Mechanism for Messages and Agents:

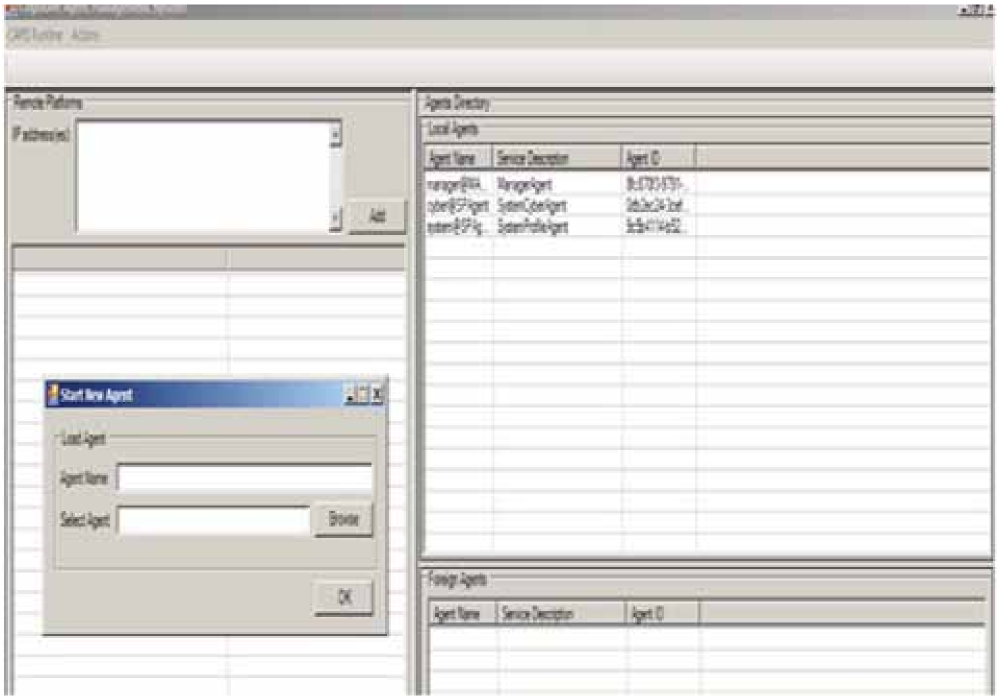
A socket based implementation has been done to control the transportation of messages. Both client and server sockets have been implemented enabling the platform to send and receive messages to and from the remote ACENET. RemoteReceiver and RemoteSender are the classes made for this purpose. RemoteReceiver and RemoteSender follow publish-subscribe mechanism. If an agent needs to get notified of a message destined for it, it should subscribe to the RemoteReceiver service of the platform. This service asynchronously calls the subscribers' MsgProc() method with the message in the argument list of that function. Only those subscribers' MsgProc() will be called along with the message for which the message is destined.

RemoteSender service when receives a message to send to a remote ACENET, it first checks the availability of the remote ACENET. If the ACENET is available, the message would be sent to that ACENET otherwise it waits for a certain period of time in order to re-connect to that ACENET. This repeats a predefined number of times and if that remote ACENET is still out of reach, the RemoteSender sends that message back to the sender, specifying the error of unreachable host.

RESULTS AND DISCUSSION

ACENET's support to the distributed environment has been tested on two different applications. In first application grid was targeted while in the other application task decomposition application was developed to test the proposed framework. It runs in both, client-server and peer-peer, environments where every platform component can be attached to the other component with the permission to ensure that the requesting platform component is authorized to join and access resources. If attached platform does not follow the trust levels and security criteria then it can be detached from the platform. Every platform component has its own container.

The master/slave design paradigm has been proposed for the development of the agent based grid computing environment. Server is responsible to manage the grid, while nodes are dynamic and active. The Master agent has the logic for the distribution of processes into tasks to nodes with idle CPU cycles in order to utilize the free CPU cycles. The programming model is a derivation of master/slave paradigm. Master class has all the controlling logic. It divides the problem into small computable tasks which are called slaves. One master class and more slave files can exist in the model. During the execution, the master thread will create slave threads, which will be executing the slave code. These threads will be mapped to the agents. When a call to create a slave thread is made, a new task will be created and assigned to any available agent, which will execute it. The slave code will report back to the master after finishing the assigned task. The messaging is supported through agent framework. ACENET has been deployed where two main agents are created: Manager Agent and Task agent. Following are the snapshots, taken from different network machines, of the running framework (Figure 4).



Agent Name	Service Description	Agent ID
manager@BIA...	ManagerAgent	[illegible]
cyber@SFAgent	SystemCyberAgent	[illegible]
system@SFAg...	SystemProfileAgent	[illegible]

Fig. 4: Application Agents running over ACENET

7. Conclusion:

The paper presented a multiagent system environment called ACENET that follows FIPA specifications on top of the .NET platform with decentralized fault-tolerant architecture supporting distributed environment. This platform provides a foundation to work on agent-oriented programming in a different domain. It provides a framework to develop next generation of distributed applications enriched with the robust features provided by the .NET platform.

Four architectural layers: AMS, DF, MTS and ATS have been constructed to implement the framework. FIPA compliant Interaction protocols and communications have been implemented to build a social community where agents may coordinate, cooperate and collaborate each other to solve complex problems. In order to avoid multicasting of messages observer pattern has been implemented. Agents subscribe to services published by the ACENET runtime environment. The work has been started to evaluate existing ontology implementations and to employ a suitable ontology scheme into the ACENET. The ACENET will be used as a test bed for evaluating different coordination algorithms. Additional services will be added to ACENET if required for the implementation of distributed problem-solving framework. A very important feature to be implemented and tested on the ACENET will be Monitoring User Behavior to mitigate Insider Threat, where a community of agents will be created to monitor, analyze and decide the behavior of the trusted user. In near future the MASIF compliant mobility will be integrated to the framework.

ACKNOWLEDGMENT

This work has been done in Center of Research for Ubiquitous Computing (CRUC) at FAST Karachi and with the financial assistance of Higher Education Commission Islamabad through Indigenous PhD scholarship. The work was completed in the guidance of Dr Zubair A Shaikh, Director FAST – National University, Karachi campus and Patron of CRUC.

REFERENCES

Alexander Serenko and Brian Detlor, 2002. "Agent Toolkits: A general overview of the market", Working pp: 455.

AgentBuilder, 2000. "*The agentbuilder user guide",* San Diego, California: Reticular Systems.

Christos Georgousopoulos and Omer F. Rana, 2002. "An approach to conforming a MAS into a FIPA-compliant system", AAMAS'02, Bologna, Italy, 15-19.

Dragone, M., D. Lillis, R. Collier, 2009. "SoSAA: a framework for integrating components & agents", Proceedings of the 2009 ACM symposium on Applied Computing, pp: 722-728, ACM New York, NY, USA publishers.

European Coordination Action for Agent based Computing, 2006. available at www.agentLink.com, updated in 2006.

Faci, N., O. Marin, 2008. "DimaX: a fault-tolerant multi-agent platform", Proceedings of the 2008 International workshop on Software engineering for large-scale multi-agent systems, Shanghai, China, pp: 13-20, ACM New York, NY, USA publishers.

Frank Manola, 1998. "*Agent Standards Overview*", Object Services and Consulting, OBJS Technical Note.

FIPA Specifications, 2002. "Foundation for Physical Intelligent Agents", FIPA's official website www.fipa.org, updated in 2002.

Holloway, E., Lamont, Gary, 2009. "Self organized multi-agent entangled hierarchies for network security", Proceedings of the 11th Annual Conference Companion on Genetic and Evolutionary Computation Conference, Montreal, Québec, Canada, pp: 2589-2596, ACM New York, NY, USA publishers.

Henri Avancini1, 2000. "A Java Framework for Multi-agent Systems", SADIO Electronic Journal of Informatics and Operations Research, 3(1): 1-12.

Jennings, N.R., 2001. "An Agent based approach for building complex software systems", Communication of the ACM, 44(4): 35-41.

Katia, P. Sycara, 2000. "Multiagent Systems", AI magazine, 19(2) Intelligent Agents Summer.

Lavinal, E., T. Desprats, Y. Raynaud, 2009. "A multi-agent self-adaptive management framework", 19(3): 217-235, John Wiley & Sons, Inc. New York, NY, USA publishers.

Miguel Contreras, Ernesto German and Leonid Sheremetov, 2004. "Design and implementation of a FIPA compliant Agent Platform in .NET", .NET Technologies'04 workshop proceedings published.

Ricci, A., D. Weyns, 2008. "Agent-oriented Programming, Systems, Languages and Applications (APSLA)", Proceedings of the 2008 ACM symposium on Applied computing, Fortaleza, Ceara, Brazil, pp: 50-51. ACM New York, NY, USA.

Shoham, Y., 2009. "Agent-oriented programming", 60(1): 51-92, Elsevier Science Publishers Ltd. Essex, UK.

Wooldridge, M., 1997. "Agent based Software Engineering", IEE Proceedings of Software Engineering, 144: 26-37.

Zhong Zhang and V. Honavar, 2004. "Multiagent System solutions for distributed computing, communications and data integration needs in the power industry", Power Engineering Society, 2004. IEEE, 1(6-10): 45 - 49.