Towards a Fault-tolerant and Decentralization-supporting .NET based Agent Framework

Mallah G. A., Shaikh N. A. & Shaikh Z. A.*

Abstract: Software Engineering and Agent-Oriented Programming (AOP) are being focused to design distributed software systems. More than 100 agent toolkits for commercial and research purpose are available. Most of these toolkits are based on Java, whereas the focus on .NET framework 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 is lack of the fault-tolerance and unavailability of decentralization.

This paper proposes and implements a distributed, decentralized, and fault-tolerant agent toolkit based on .NET to solve distributed and complex problems. The proposed agent toolkit, ACENET (Agent Collaborative Environment based on .NET), follows FIPA specifications and works under decentralized architecture.

Keywords: Agent toolkits, Agent-oriented programming, .NET framework, FIPA, Distributed Environment

1. INTRODUCTION

Agent-oriented programming paradigm is the appropriate practice to model, design and implement the complex and distributed software systems [1]. 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 [2]. It is viewed as abstraction 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. [3].

The justification of the Multiagent 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 [4].

The architecture that provides implementation environment to multi-agent 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 multi-agent systems implementation environment such as JADE, FIPA-OS, Zeus [5], etc. With the increasing number of these

* Shah Abdul Latif University, Khairpur, E-mail: ghulamali@salu.edu.pk
frameworks, two parallel agent development standards have evolved: FIPA and MASIF [6]. 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 [7].

In this paper we intend to focus on very recently emerging area, i.e. Agent Oriented Programming (AOP) on the top of the .NET framework so that the developers of this domain may develop supporting and execution environment for multi-agent systems. In the next phase of the research we will address the interoperability issues among different platforms, i.e. how agents executing on .NET platform communicate with the agents not running on the .NET platforms, etc.

2. FIPA SPECIFICATIONS

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 [8].

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.

![Diagram of FIPA Agent Platform Reference Model](image)

FIPA standard does not talk about mobility as a mandatory requirement for FIPA compliant system (as shown in figure 1). 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. RELATED WORK

A variety of agent toolkits, academically and commercially, is available in the market. According to the AgentLink [9] 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 [10]. Most of the FIPA and MASIF compliant Mobile Agent Toolkits and Multiagent System toolkits are developed in Java [11].
Towards a Fault-tolerant and Decentralization-supporting .NET based Agent Framework

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 [12]. 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 & 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 the requirement of the application. All machines are independent to connect or disconnect the framework.

4. ACENET ARCHITECTURE

The ACENET has been designed for the organization of the agents and integrate complete distributed agent applications. The architecture of ACENET consists of four major components shown in figure 2 whereas low level components are shown in figure 3. The description of the major components is given below.

(a) Application Agents
(b) Agent Management and Directory Facilitator Services
(c) Message Transport Service
(d) Agent Transport System

Figure 2: Major Components of ACENET
Application agents interact with AMS and DF, and use the services of Agent Transport Service (ATS) and Message Transport Service (MTS) through AMS. 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 searching agents and their services at runtime by specifying search criteria such as AgentID, Agent Name, and Service Type etc. MTS supports asynchronous messaging between agents within the same platform and between ACENET to ACENET. ATS offers message passing service to agents between ACENET to ACENET. In future, the functionality of ATS will be extended for mobility as well. ACENET has decentralized architecture, shown in figure 5.3(a). Because of that if one node does not work properly or fails to run and isolated; the other nodes remain intact and function without disturbance. 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. There was the extensive use of the .NET Remoting in the implementation of ACENET mainly for the communication and management purpose. Besides other implementations, the platform also provides configuration tools for Agent Management System, Directory Facilitator and Message Transportation. The detailed implementation of the components is given below.
5. IMPLEMENTATION DETAILS

There was the extensive use of the .NET Remoting in the implementation of ACENET mostly for the communication & the management purpose. Along with other implementations the platform also provides configuration tools for Agent Management System, Directory Facilitator & 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.

![Figure 4: ACENET Decentralization-supporting Architecture](image)

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.

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.

6. CONCLUSION AND FUTURE DIRECTIONS

The paper presents multi-agent system environment and implemented over .NET for Windows operating systems user. The developed agent framework, ACENET, follows FIPA specifications with decentralized fault-tolerant architecture supporting distributed environment. This framework 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 .NET platform.
Towards a Fault-tolerant and Decentralization-supporting .NET based Agent Framework

Figure 5: ACENET GUI (Application Agents Running Over ACENET)

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.

REFERENCES