International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:1, No:1, 2007

A Flexible and Scalable Agent Platform for Multi-Agent Systems

Ae Hee Park, So Hyun Park, and Hee Yong Youn

Abstract**—**Multi-agent system is composed by several agents capable of reaching the goal cooperatively. The system needs an agent platform for efficient and stable interaction between intelligent agents. In this paper we propose a flexible and scalable agent platform by composing the containers with multiple hierarchical agent groups. It also allows efficient implementation of multiple domain presentations of the agents unlike JADE**.** The proposed platform provides both group management and individual management of agents for efficiency. The platform has been implemented and tested, and it can be used as a flexible foundation of the dynamic multi-agent system targeting seamless delivery of ubiquitous services.

Keywords**—**Agent platform, container, multi-agent system, services, ubiquitous computing.

I. INTRODUCTION

MULTI-AGENT system achieves the goal through co-work of the agents showing autonomous behaviors. The multi-agent system needs a platform for efficient and stable interaction between intelligent agents [1]. Each domain and agent existing in the area is registered in the agent platform. Several platforms can be connected together through the interface offered for convenient implementation, and thereby high scalability is supported.

The agent platform uses standard agent communication language (ACL) for the communication between the agents. It needs to have the agent administration and management system (AMS) module for providing white page service. It, also, requires the modules supporting additional agent services such as yellow page service.

The Java Agent Development Environment (JADE) [8] is a framework used for implementing multi-agent system, which conforms to the FIPA standard. It is available to implement only the Java-based system and hard to implement multiple domain presentations of agent system. SEAGENT [9] is a platform used to develop semantic web-based multi-agent system. All agents in the platform follow semantic web standards to represent their internal knowledge and provide semantic service, directory service, and ontology service. However, it must provide a means to define the mappings between platform anthologies and external anthologies, and the translation process is based on the mappings.

In this paper, thus, we propose a new agent platform architecture which uses containers to manage the agents like JADE. However, it can manage the agents more efficiently by composing the containers with multiple hierarchical agent groups. Also, it has the classes and functions allowing communication between the agents implemented in various languages such as java and c++. The proposed agent platform uses the ACL of FIPA [5], and conforms to the standard agent technologies. As a result, it is compatible with the system developed using JADE.

The proposed agent platform is composed by the system developed based on the standard agent technology of FIPA, and it supports agent services operating as agents. The multi-agent system developed with the proposed agent platform can be flexible since it supports both group management and individual management of the agent at the same time. It has essential functions for the communication of the agents and includes agent monitoring functions. Moreover, it allows reflection for self-configuration. As a result, the proposed agent platform provides a flexible foundation for dynamic multi-agent system which can be used for seamless delivery of ubiquitous service [2] of u-home, u-health, etc. In this paper we introduce the proposed agent platform architecture, while focusing on the implementation. We also provide an explanation on the operation sequence between the modules. The proposed platform has been implemented in the CALM (Component-based Autonomic Layered Middleware) [1] developed by the authors, which is reflective to efficiently support dynamic operation in ubiquitous environment.

The remainder of the paper is organized as follows. Section II deals with the related work and Section III introduces the proposed agent platform. The implementation of the proposed agent platform core is presented in Section IV. Finally, Section V summarizes the proposed approach and ongoing work.

Manuscript received November 30, 2006. This work was supported in part by the Ministry of Information and Communication, 21C Frontier Project on Ubiquitous Computing and Networking and the Ministry of Education, Brain Korea 21 Project.

A. H. Park is with the School of Information and Communication Engineering, Sungkyunkwan University, Suwon, Korea, 440-736 (e-mail: ahpark@skku.edu).

S. H. Park is with the School of Information and Communication Engineering, Sungkyunkwan University, Suwon, Korea, 440-736 (e-mail: sohyun1027@skku.edu).

H. Y. Youn is with the School of Information and Communication Engineering, Sungkyunkwan University, Suwon, Korea, 440-736 (corresponding author to provide phone: 82-31-290-7147; fax: 82-31-290-7231; e-mail: youn@ece.skku.ac.kr).

II. The Related Work

A. Agent-Based Computing

An agent is a software component capable of flexible autonomous operation in a dynamic, unpredictable and open environment. Agent technology is a natural extension of current component-based approaches, and includes distributed planning, decision-making, automated auction mechanisms and learning mechanisms [3].

Multi-Agent Planning: A multi-agent system consists of several agents capable of reaching the goal collectively. Planning the interdependent systems includes ensuring the interoperability of different agents, attempting to optimize the overall plan schedule and the distribution of local planning tasks by a central agent. For example, there are models of team or group activity in which the agents collaborate towards specific objectives.

Agent Communication Language: Two common languages for inter-agent communication are KQML and FIPA ACL. KQML was developed as a part of the ARPA Knowledge Sharing Effort, and it includes the message format and message-handling protocol to support run-time knowledge sharing among the agents [4]. FIPA is an international organization dedicated to promoting the industry of intelligent agents by openly developing specifications supporting interoperability among the agents and agent-based applications [5]. FIPA ACL message contains a set of one or more message parameters that vary according to the situation for effective agent communication.

Agent Architectures: Agent architectures are the fundamental engines underlying the autonomous components that support effective behaviors in real-world, dynamic and open environments.

B. Autonomic Computing

Autonomic computing is a systematic approach to achieving computer-based systems which manage themselves without human interventions [6]. The autonomic computing capabilities are based on four characteristics of self-managing systems. **Self-configuring** allows the system to dynamically adapt to the deployment of new components or changes with minimal human intervention. **Self-optimizing** can efficiently tailor resource allocation and utilization to meet the user needs and ensure optimal quality of service. **Self-healing** automatically detects, diagnoses, and repairs localized software and hardware problems. **Self-protecting** automatically defends against malicious attacks or cascading failures. Huang et al. [7] presents a reflection-based approach for autonomic computing middleware. It shows the philosophy that autonomic computing should focus on how to reason while reflective computing supports how to monitor and control the system. Here, reflective computation collects the states and behaviors of basic computation. Then, autonomic computation measures and analyzes the data and decides when and what to change. Finally, reflective computation decides how to change and enforces the changes.

C. JADE Platform

JADE is a software framework simplifying the implementation of multi-agent systems. JADE platform has containers to hold the agents, and a main container resides on the host which runs the RMI server of the platform. The agents are implemented as Java threads and live within the agent containers providing runtime support to the execution of the agents. JADE platform includes the following three components automatically activated at the agent platform start-up time [8].

AMS (Agent Management System): AMS provides white-page and life-cycle service, maintaining a directory of agent identifiers (AID) and agent state. Each agent must register with an AMS in order to get a valid AID.

DF (Directory Facilitator): DF is the agent providing the default yellow page service in the platform.

ACC (Agent Communication Channel): ACC is the software component controlling the exchange of messages within and between the platforms.

III. The Proposed Scheme

We first present the overview of the proposed agent platform architecture. We then show the sequence flow between the main modules in the architecture core. Also, we show group management and personalization administration for efficient agent management.

A. Overview of the Agent Platform Architecture

The agent platform includes important information and functions for agent management internally. The AMS and DF are registered in the platform as an agent, respectively. Fig. 1 shows that the agent platform includes the Message Transport Protocol (MTP) module for reliable message processing using the ACL of FIPA with HTTP. The agent platform contains the core module responsible for the management of the agent platform and agents. The MTP module is constructed as follows.

- HTTP Communication: This is a module built for the communication between the agents, and embodied according to the HTTP. An agent has a HTTP server and client module in itself. When a message is received, the HTTP server processes it. When a message is sent, the HTTP client is used for the communication.
- ACL Encoder/Decoder: This is a parser module encoding and decoding the ACL message. An agent can send a message simultaneously to several agents through the process of ACL.

- Messaging Service: This is a message queue module that supports ordering and queuing of messages in case of receiving/sending messages from/to several agents at the same time.

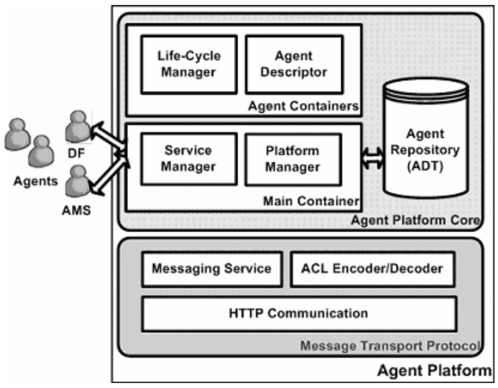


Fig. 1 The structure of the proposed agent platform

The main modules of the agent platform core are as follows.

- Main Container: This is the main module that manages the agent platform.
  - Platform Manager: This is a submodule that handles the initialization of the agent platform and delivery of messages from the MTP level to the agent processing module.
  - Service Manager: This is a submodule that registers and manages additional services such as reflective module.
- Agent Container: This is a module that manages groups of agents having similar roles or functions hierarchically.
  - Life-Cycle Manager: This is a module that manages and monitors the activity of an agent as a thread. The methods used are start, suspend, resume, stop etc.
  - Agent Descriptor: This is a module managing the information of agent description such as id, address, roles, etc.
- ADT (Agent Description Table): This is a hash table managing the reference information of the description of the agent using the hash key of the agent id.

The agent platform core uses the MTP library, and it hierarchically consists of one main container and several agent containers. An agent container manages the groups of agent descriptions, where the agent description contains the information required for connection and management of the agent and status information for life cycle. The agent platform monitors and manages the agents as threads. The threads take actions such as start, stop, and resume, running dynamically with the present state of the agent.

B. *Sequence Flow of Agent Platform Core*

The agent platform core has important parts that allow interactions among the agents and manage the execution of agent platform. It governs the relationships between the modules and services through the main action flow. Fig. 2 shows the sequence among the principal modules of the agent platform core in case of initializing the agent platform, registering and releasing an agent, and delivery of message.

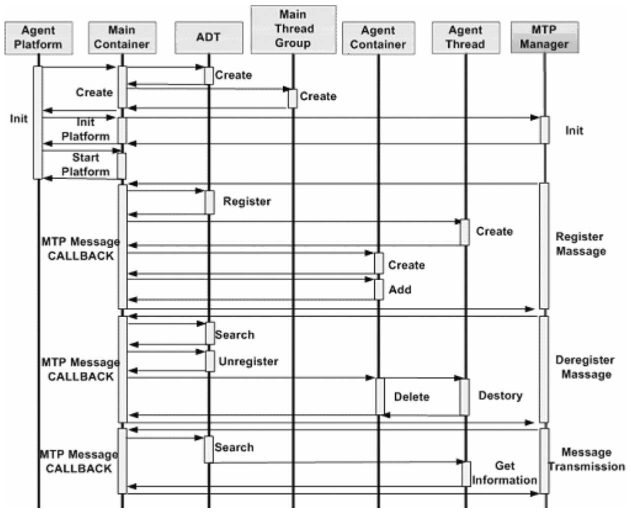


Fig. 2 The main operational sequence of agent platform core

When the agent platform begins to operate, it creates Main Container first. The Main Container reads the setting file of XML format for initialization, which has the version information of the platform along with the name, address, port, and resource information. It also creates the ADT to directly access the agent description information and Main Thread Group Table that manages the Agent Container's reference values. Then, it finishes the initialization to receive the ACL messages by initializing the communication server module of the MTP.

Each agent sends an ACL message to register at the agent platform. Then, the agent platform core receives a callback message from the MTP. It then confirms whether it is a message sent by the agent platform through the receiver part of the message. After checking the content of the message if it is a registration message, it extracts the agent’s name, address, port, description information, container information from the content. Then, it examines the Agent ID from the ADT using the hash key whether it has been already registered or not. If it has already been registered, the platform sends a failure message indicating ‘already-registered’ to the agent. After the agent is registered at the ADT, the agent platform executes an agent thread. The agent thread has the agent description information, and it dynamically monitors the agent according to its state. It then checks if there already exists a relevant agent container. If not, it makes a one. Otherwise, it adds it to the existing agent container. When the agent registration process is completed, it sends the agent the result of registration process.

When an agent sends a destroy message, the agent platform core receives an ACL message through callback. Then it confirms that the receiver part of it is agent platform and the content is for deletion. It searches the ADT using the hash key of the agent id, and deletes the reference value of the relevant agent information. Also, it destroys the agent thread and deletes the reference of it from the table of the agent container.

The agent platform core delivers messages to relevant agent in case the receiver of the ACL message entering through the message callback function is not the agent platform. The sender agent does not know the IP Address and Port information of the receiver agent but only the name. Therefore, it sends the request for transmission to the agent platform. The agent platform core searches the ADT to find the address equivalent to the name of the receiver agent. The agent platform core receives necessary information from the agent thread managing the agent description information. It also talks to the relevant agent after changing the receiver's name and attribute value of the address, and so on. The principal operational sequence is supported by the MTP. Also, the platform core collects the information of management obtained by the thread monitor.

C. *Management of Agents in the Agent Platform Core*

The agent platform core supports both group management and individual management of the agents for efficiency. Also, it uses the ADT for fast access on individual information of the agents. The agent platform includes multiple group management of the agents, where the groups are hierarchically formed. The management of agent group supports the administration of the agents belonging to the group – establishment of priority order, start, stop, resume, kill, etc.

For example, refer to Fig. 3. There exists "Sports" agent belonging to the Entertainment's Outdoor group, and it is referred by News group. When it registers at the agent platform "A" as in the figure, it sends an ACL message that includes the agent's address information, name, agent description information, container information, etc. The container information consists of the primary domain name, "Entertainment:Outdoor", and the referred domain name, "News". The agent platform searches the ADT whether there already exists the agent description information. Then it makes a key with the name and address of the agent if it accepts the "Sports" agent registration information. It registers it at the ADT after making an instance inherited Agent Thread class in case of no existence in the ADT. Then, it examines the Agent Group Management Table for managing the agent group and checks whether "Entertainment Agent Container" exists. The agent platform then creates "Outdoor Agent Container" in case the management table of "Entertainment Agent Container" does not include that container. The management table of "Outdoor Agent Container" includes the reference of "Sports" agent instance made for ADT registration. Also, "News Agent Container" gets the reference of "Sports" agent instance. Then "Sport" agent is accessed through several groups such as News Agent Container and Entertainment Agent Container. The management of agent group includes the message priority, enumerating, adding, removing, getting the number of agent, and so on. It operates differently according to the situation such as "Broadcast" agent uses "Music" agent and "Sports" agent. Therefore, the agent container can apply same or different policy to the agents in a group of agents.

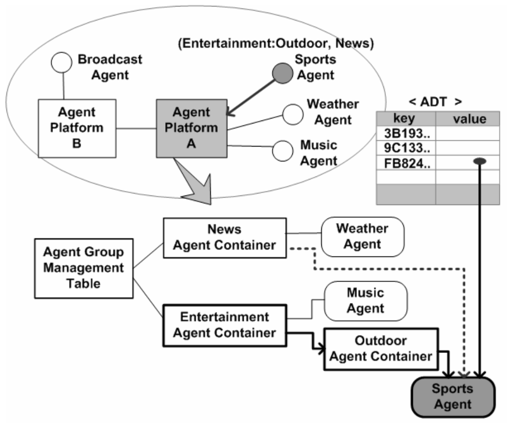


Fig. 3 Registration of "Sports" agent to the agent platform

Unlike JADE [8], the proposed agent platform core provides the presentation of multi-domain by supporting efficient group management internally. However, it might take long time to find the information of the agent through the hierarchical group management process. Therefore, we use the ADT at the same time the group is managed hierarchically, and the agent description information can be accessed directly.

The agent platform core finds and manages the information of description of Agent ID by monitoring the state of the agent using a thread. Also, the agent thread becomes a virtual agent having a state since it acts differently according to the situation of the agent. The agent thread has a structure with which adjustment is possible by applying the policy decided according to the property of the agent. It operates independently from the process of the platform manager of Main Container module.

The agent thread monitoring the agent gets information of the agent from the message delivered among the agents. When a set of agents in a multi-agent system is $A = \{a_1, a_2, \ldots a_i \ldots a_n\}$ and agent a_i is receiver of message, m_k, the monitored information is follows.

- Current message size: $m_{c\_size} = sizeof(m_k)$
- Accumulated message size: $m_{h\_size=} \sum_{j=1}^{k-1} sizeof(m_j)$
- The total number of delivered message: $m_{cnt} = k - 1$
- The ratio of current message: $m_{ratio} = \frac{m_{c\_size}}{m_{h\_size}} \times m_{cnt}$
- Monitored data from the messages sent to a_i by major agents: $S = \{S_1, S_2, \ldots S_i \ldots S_n\}$ is a set of major senders, and the agent platform includes the monitored

information relative to each element. That is $s_j = \{m_{j\_c\_size}, m_{j\_h\_size}, m_{j\_cnt}, m_{j\_ratio}\}$

The monitored information is used to measure message convergence, specification of agent's importance, and other information relevant to the agents.

IV. Implementation of Agent Platform Core

The agent platform core imports the MTP library, and the main modules consist of classes. The MTP library consists of DllQueuing library for processing the message queues, NetworkLib for HTTP communication, and ACL_Parser_DLL for ACL message processing. Fig. 4 shows the class diagram of the agent platform core. The principal classes are CMainContainer, ThreadGroup, CLADT, CAP_Thread, and CAgentInterface.

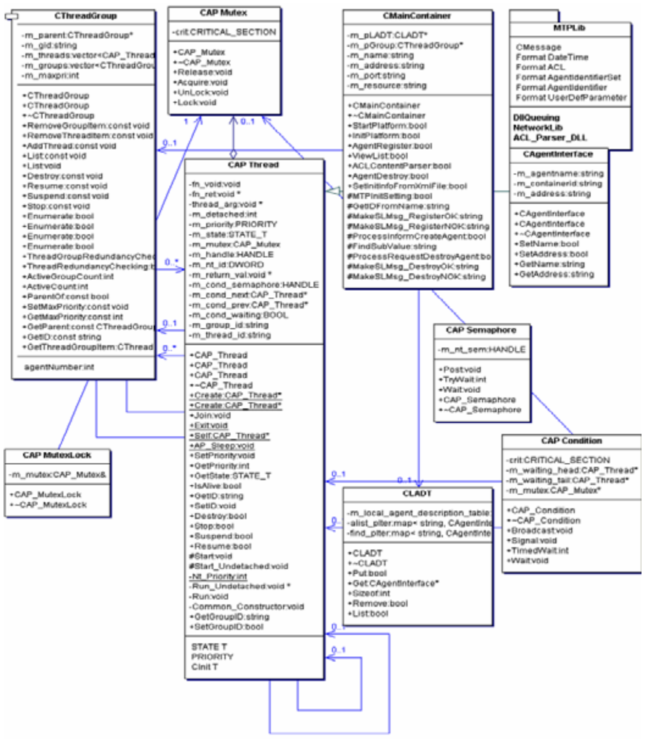


Fig. 4 The class diagram of the agent platform core

CMainContainer is the class managing the Main Container and its main methods are as follows.

- bool StartPlatform (): Starting the execution of agent platform
- bool InitPlatform (): Initializing the agent platform
- bool MTPInitSetting (void): Initializing the MTP
- bool AgentRegister: (string cid, string name, string address): Processing registration of the agent at the agent platform
- bool AgentDestroy (string AgentName): Destroying the agent from the agent platform
- bool ProcessMsgTransmission (CMessage msg): Processing delivery of message to one or more receivers
- bool ACLContentParser (CMessage msg): Processing the ACL message received from the MTP Library

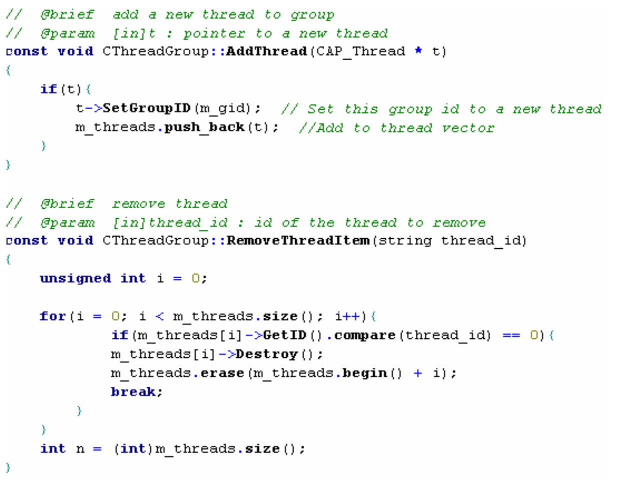
CThreadGroup is a class for the administration of agent container and management of sub-container references and agent threads using a vector. Fig. 5 shows the simplified declaration of the CThreadGroup class. The class constructor receives the group id and pointer from the parent group as parameters.

```
class THREADAPI CThreadGroup
{
public:
    CThreadGroup(CThreadGroup* parent, string gid);
    virtual ~CThreadGroup();

private :
    CThreadGroup* m_parent;           // The parent thread group
    string m_gid;                     // Thread group ID
    vector<CAP_Thread *> m_threads;   // The agent threads in the group.
    vector<CThreadGroup *>  m_groups; // Child agent thread groups
    int m_maxpri;

public:
    const void RemoveGroupItem(string group_id);    // Remove agent group
    const void RemoveThreadItem(string thread_id);  // Remove agent thread
    const void AddThread(CAP_Thread * t);           // Add agent thread
    const void Destroy();                           // Destroy child threads
                                                    //      and child groups
    const void Resume();                            // Resume child threads
    const void Suspend();                           // Suspend child threads
    const void Stop();                              // Stop child threads

    // Show list of active threads
    bool Enumerate(vector<CThreadGroup *> * list);
    bool Enumerate(vector<CThreadGroup *> * list, bool recurse);
    bool Enumerate(vector<CAP_Thread *> * list);
    bool Enumerate(vector<CAP_Thread *> *list, bool recurse);

    int ActiveGroupCount();       // Get the number of all active threads
    int ActiveCount();            // Get the number of active threads in current group
    int getAgentNumber();         // Get the number of threads

    const void SetMaxPriority(int maxpri);          // Set maximum priority
    const int GetMaxPriority();                     // Get maximum priority
    const CThreadGroup* GetParent();                // Get the parent of this group
    const string GetID();                           // Get thread group id
    CThreadGroup * GetThreadGroupItem(string gid);  // Get an agent thread
};
```

Fig. 5 The CThreadGroup class

The AddThread() mothod adds a new agent thread to the thread vector. The RemoveThreadItem() method deletes an agent thread from the thread vector. The RemoveGroupItem() method removes a sub-container from the current group vector. The Destroy() method removes all agent threads and child group vectors, and then destroys the instance of the group class. The Enumerate() method shows the list of active threads in the group. These methods are used for group management. Fig. 6 illustrates the processes of the AddThread() and RemoveThreadItem() methods.

The main method which manages the agents belonging to the group are Resume(), Suspend(), Stop(), ActiveCount(), and so on. Moreover, the getAgentNumber() method gets the number of agents belonging to the group. The SetMaxPriority () method sets a highest priority and the GetMaxPrioriy method gets the priority of the group. The GetParent() method gets a Spointer to the parent group of the group. The GetID() method gets the id of the group and the GetThreadGroupItem() method gets a pointer to the group which has the group id of input parameter. The format of a group id has the form of GUID (Globally Unique Identifier).



```
//  @brief  add a new thread to group
//  @param  [in]t : pointer to a new thread
const void CThreadGroup::AddThread(CAP_Thread * t)
{
    if(t){
        t->SetGroupID(m_gid);  // Set this group id to a new thread
        m_threads.push_back(t);  //Add to thread vector
    }
}

//  @brief  remove thread
//  @param  [in]thread_id : id of the thread to remove
const void CThreadGroup::RemoveThreadItem(string thread_id)
{
    unsigned int i = 0;

    for(i = 0; i < m_threads.size(); i++){
            if(m_threads[i]->GetID().compare(thread_id) == 0){
            m_threads[i]->Destroy();
            m_threads.erase(m_threads.begin() + i);
            break;
        }
    }
    int n = (int)m_threads.size();
}
```

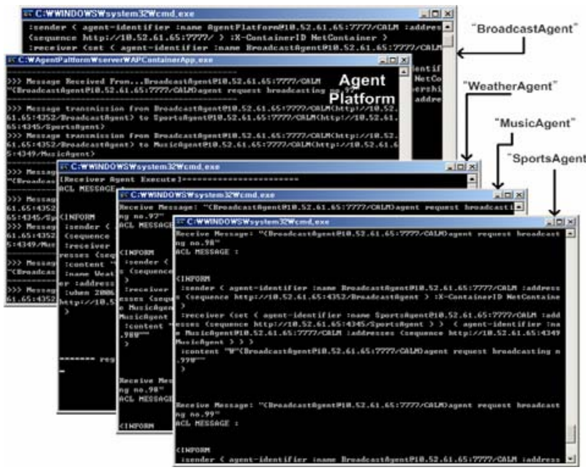
Fig. 6 The AddThread and RemoveThreadItem methods

CAP_Thread is a class for agent life cycle management and acts according to the state of the agent. It uses classes connected with a semaphore, condition, mutex for the creation and management of threads. The main methods of CAP_Thread are as follows.

- STATE_T GetState(): Getting the current state
- string GetID(void): Getting the agent id
- void SetID(string tid): Setting the agent id
- bool Destroy(void): Destroying the thread
- bool Stop(void): Stopping the thread
- bool Suspend(void): Suspending the thread
- bool Resume(void): Resuming the thread
- void Start(): Stating the thread
- void SetPriority(PRIORITY pri): Setting the priority
- PRIORITY GetPriority(): Getting the thread's priority

CAgentInterface is a class managing the information of agent description and it is inherited CAP_Thread class. It can get agent description information and manage the life cycle according to the situation of the agent using a thread. CLADT is the class managing the ADT, and its main methods are Put(), Get(), Sizeof(), Remove(), and so on.

Fig. 7 shows the snapshots of the screens of communication among the agents in the proposed platform – BroadcastAgent, WeatherAgent, MusicAgent, and SportsAgent. The BroadcastAgent sends the agent platform the ACL message whose receiver part includes the names of the MusicAgent and SportsAgent. Then, the agent platform delivers the message to them after searching the address information.


C:WWINDOWSWsystem32Wcmd.exe
C:WAgentPlatformWserverWAPContainerApp.exe
Agent Platform
"BroadcastAgent"
"WeatherAgent"
"MusicAgent"
"SportsAgent"

```
:sender ( agent-identifier :name AgentPlatform@10.52.61.65:7777/CALM :addres
(sequence http://10.52.61.65:7777/ ) :X-ContainerID NetContainer )
:receiver (set ( agent-identifier :name BroadcastAgent@10.52.61.65:7777/CALM
```

```
>>> Message Received From...BroadcastAgent@10.52.61.65:7777/CALM
"(BroadcastAgent@10.52.61.65:7777/CALM)agent request broadcasting no.97"
>>> Message transmission from BroadcastAgent@10.52.61.65:7777/CALM(http://10.52.
61.65:4352/BroadcastAgent) to SportsAgent@10.52.61.65:7777/CALM(http://10.52.61.
65:4345/SportsAgent)
>>> Message transmission from BroadcastAgent@10.52.61.65:7777/CALM(http://10.52.
61.65:4352/BroadcastAgent) to MusicAgent@10.52.61.65:7777/CALM(http://10.52.61.6
5:4349/MusicAgent)
```

```
[Receiver Agent Execute]========================
ACL MESSAGE :
```

```
Receive Message: "(BroadcastAgent@10.52.61.65:7777/CALM)agent request broadcasti
ng no.97"
ACL MESSAGE
```

```
Receive Message: "(BroadcastAgent@10.52.61.65:7777/CALM)agent request broadcast
ng no.98"
ACL MESSAGE :


(INFORM
 :sender ( agent-identifier :name BroadcastAgent@10.52.61.65:7777/CALM :address
s (sequence http://10.52.61.65:4352/BroadcastAgent ) :X-ContainerID NetContaine
 )
 :receiver (set ( agent-identifier :name SportsAgent@10.52.61.65:7777/CALM :add
resses (sequence http://10.52.61.65:4345/SportsAgent ) )  ( agent-identifier :na
e MusicAgent@10.52.61.65:7777/CALM :addresses (sequence http://10.52.61.65:4349
MusicAgent ) ) )
 :content "W"(BroadcastAgent@10.52.61.65:7777/CALM)agent request broadcasting n
.97W""
 )



Receive Message: "(BroadcastAgent@10.52.61.65:7777/CALM)agent request broadcast
ng no.99"
ACL MESSAGE :


(INFORM
 :sender ( agent-identifier :name BroadcastAgent@10.52.61.65:7777/CALM :address
```

Fig. 7 The communication among the agents

V. Conclusion

In this paper we have introduced a new architecture of agent platform that provides an interactive environment between intelligent agents. We have also showed the sequence flows of registration, removal, and transmission of agents in the proposed agent platform core. It allows both the group management and individual management of the agents for efficiency. Moreover, we showed the implementation of the overall agent platform core. It includes various service agents such as AMS and DF which are required to support multi-agent system dynamically and flexibly. The proposed agent platform has the mechanisms for managing several agents efficiently at the same time. It monitors the messages transmitted between the agents, and extracts information from them. The proposed agent platform conforms to the standard agent technology and allows scalability. Therefore, it can effectively support interactions between the agents used for various ubiquitous computing applications. As the future work, we will further optimize the agent platform and implement dynamically configurable and fault tolerant agent platform.

References

[1] Youn, H.Y. et al. "CALM: An Intelligent Agent-based Middleware Architecture for Community Computing", Proceedings of the third Workshop on Software Technologies for Future Embedded & Ubiquitous Systems (SEUS 2006).
[2] Kindberg et al, "System software for ubiquitous computing", *IEEE, Pervasive Computing*, pp 70-81, Jan.-March 2002.
[3] Michael Luck et al. "Agent Technology: Enabling Next Generation Computing" AgentLink community (2003).
[4] KQML-Knowledge Query and Manipulation Language, http://www.cs.umbc.edu/kqml/
[5] FIPA-Foundation for Intelligent Physical Agents, http://www.fipa.org
[6] IBM. Autonomic Computing: IBM's Perspective on the State of Information Technology, http://www.ibm.com/research/autonomic, 2001.
[7] G HUANG et al. "Towards Autonomic Computing Middleware via Reflection", *IEEE* , Computer Software and Applications Conference (COMPSAC'04), vol.1, pp. 135 - 140, 2004.
[8] JADE, Java Agent Development framework, http://jade.cselt.it
[9] Oguz Dikenelli et al. "SEAGENT: A Platform for Developing Semantic Web based Multi-Agent Systems", AAMAS'05, 2005.