Agent Communication and Migration for Distributed Applications in Mobile-C

Khaoula Addakiri
Department of Mathematics and Computer Science,
Université Hassan 1er, FSTS, LABO LITEN
Settat, Morocco

Mohamed Bahaj
Department of Mathematics and Computer Science,
Université Hassan 1er, FSTS, LABO LITEN
Settat, Morocco

ABSTRACT

In this paper we present the design and implementation of Mobile-C. This research is based on distributed application and the goal is to access an XML file in order to find some information and guaranteed that the data of mobile agent is only accessed by one agent on time by using the synchronization.

Keywords*:* Mobile agent; Agent communication; Distributed applications;synchronization.

1.INTRODUCTION

A distributed system is a set of autonomous machines connected through a network and composed of distinct software dedicated to the coordination of system activities, and leverage the availability of several distributed resources to provide better scalability.

Mobile agent is an autonomous software entity responsible for executing a programmatic process, which is able to migrate across a network. An agent migrates in a distributed environment between agencies. When an agent migrates, its execution is suspended at the original agency, the agent is transported to another agency in the distributed environment, and, after being re-instantiated at the new agency, the agent resumes execution.

The majority of mobile agent platforms in use are Java-oriented. Multiple mobile agent platforms supporting Java mobile agent code include Mole [1], Aglets [2], Concordia [3], JADE [4], and Agents [5]. Using a standard language like the mobile agent code language that provides both high-level and low-level functionalities is a good choice to treat with the large number of distributed applications. The choice of C/C++ is a proper for a mobile agent code language because it's provides powerful functions in terms of memory access. A several number of C/C++programs can be used as mobile agent code. Furthermore, C is a language which can easily interface with a variety of low-level hardware devices. Ara [6, 7] and TACOMA [8] are two mobile agent platforms supporting C mobile agent code, while Ara also supports C++. Mobile agent code is compiled as byte code [9] and machine code [10] for execution in both Ara and TACOMA, respectively.

Mobile-C [11–14] was developed as a standalone mobile agent platform in order to support C/C++ mobile agent code. Mobile-C chose an embeddable C/C++ interpreter—Ch [15–17] to run C/C++ mobile agent source code. This approach can stave off some potential problems, such as the implementation of the system which could be issued by the compiling approach, the execution security and the portability of the platform. Mobile agent migration in Mobile-C is achieved through the Foundation for Intelligent Physical Agents (FIPA) agent communication language (ACL) messages. Using FIPA ACL messages for agent migration in FIPA compliant agent systems simplifies agent platform, reduces development effort and easily achieves inter-platform migration through well-designed communication mechanisms provided in the agent platform. Messages for agent communication and migration are expressed in FIPA ACL and encoded in XML.

This article presents an exploration of using XML to represent different types of information in mobile agent system information and ensuring that the data of mobile agent is only accessed by one agent on time by using the mutex as synchronization. The system not only uses XML to represent agent communication messages and mobile agent messages, and processes these XML messages in binary agent system space, but also allows mobile agents to process XML data interpretively to avoid the need of an interface layer between script mobile agents and system data represented in XML Mobile-C uses IEEE FIPA ACL messages for inter-agent communication and inter-platform mobile agent migration. The remainder of the article is structured as follows. Section 2 introduces the architecture of Mobile-C. Section 3 shows two types of messages in Mobile-C, agent communication messages and mobile agent messages and presents how mobile agent migrate from multiple hosts. Section 4 gives an example of mobile agent that access to XML data and illustrates the synchronization support in Mobile-C.

2.THE ARCHITECTURE OF MOBILE-C

The system of mobile-C is shown in Fig.1. Agencies are the major building blocks of the system and abode in each node of a network system in order to support Stationary Agents (SA) and Mobile Agents (MA) at runtime. They serve for locating and messaging agents, moving mobile agents, collecting knowledge about other agents and providing several places where the agent can be run. The core of an agency provides local service for agents and proxies remote agencies. The principle of an agency and their functionalities can be summarized as follows [18]:

- Agent Management system (AMS): The AMS manages the life cycle of agents in the system. It relates the creation, authentication, registration,

deletion, execution, migration and persistence of agents. AMS is also responsible for receiving and dispatching mobile agents .Each agent must register with an AMS in order to get a valid AID.

- Agent Communication Channel (ACC): The ACC forwards messages between local and remote entities. The interaction and coordination of mobile agents and host systems can be performed through agent communication language (ACL).
- Agent Security Manager (ASM): The ASM is responsible for protection of access for platform and infrastructure.
- Directory Facilitator: DF serves yellow page services. Agents in the system can register their services with DF for providing to the community. They can also look up required services with DF.
- Agent Execution Engine (AEE): AEE serves as the workhorse for mobile agents. Mobile agents must reside inside an engine to execute. AEE has to be platform independent in order to support a mobile agent executing in a heterogeneous network.

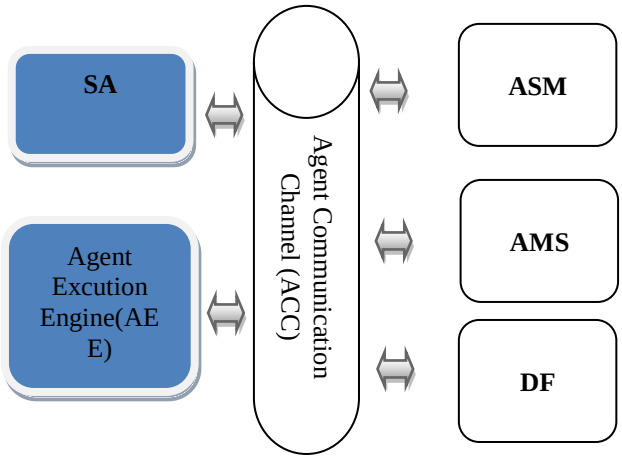

Figure1.The architecture of agencies in Mobile-C.

3.MESSAGES AND MIGRATION OF MOBILE AGENT IN MOBILE-C

A. Messages in Mobile-C

In Mobile-C there are two types of messages, agent communication messages, and mobile agent messages. A sample agent communication message is from agent-a to agent-b as shown in Fig.2. The message is encoded in XML. In Fig.2, the sender and recipient of the message are identified by their identifiers. For the sample message, the sender and receiver agent names are X and Y, respectively. The sender and receiver agent addresses are http://1.fsts.ac.ma:5120 and http://2.fsts.ac.ma: 5120, respectively.

```
<?xml version="1.0" ?>
<sender>
  < identifier>
    <name>X </name>
    <adresse>
    <url> http://1.fsts.ac.ma:5120</url>
    </adresse>
  </ identifier>
</sender>
<receiver>
   < identifier>
     <name>Y</name>
     <adresse>
     <url> http://2.fsts.ac.ma:5120</url>
     </adresse>
   </ identifier>
</receiver>
```

Figure2.An ACL message represented in XML

A mobile agent message contains general information about the mobile agent and the tasks performed by the agent in a remote host. The general information of mobile agent includes the name, the owner and the home agent where the mobile agent is created. The task information contains number of tasks, description of tasks and code of each task as shown in figure3. During the migration, the task performed by the mobile agent will be encapsulated into agent messages. At the end of the migration, all results of tasks must be sending back to the home agency.

```
Message type: Mobile_Agent
Sender: Agent ID
Receiver: Agent ID
Content
  name
  owner
  home
  tasks:
      task1
      task2
        .
      task n
```

Figure 3.A mobile message

B. The migration of mobile agent in Mobile-C

Mobile agent is a software agent who is able to migrate from one host to another in a network and resume the execution in the new host. The migration and the execution of mobile agents are supported by a mobile agent system. According to whether an execution state is transferred, an agent can migrate in two different ways: strong migration, and weak migration. Strong migration means that all of the agent execution state and data state are captured and transferred to a target host. After a strong migration, the agent continues to execute its program exactly at the point at which it has been interrupted before the migration. Weak migration means that only the data state is transferred from one host to another.

Since the current version of Ch does not offer the capability to capture the execution state of a process or thread, Mobile-C supports weak migration. The task of a mobile agent can be divided into several subtasks which can be executed in different hosts and listed in a list of tasks as shown in figure4. The task list can be modified by adding new subtasks and new conditions.

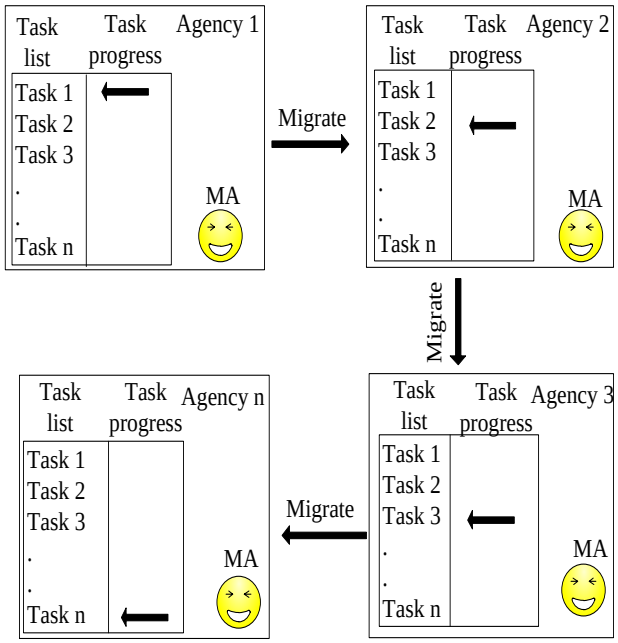

Figure4. Agent migration based on a task list and a task progress pointer.

Changing dynamically the task list improves the flexibility of a mobile agent. Thus, once we start the execution of a subtask in a host, the mobile agent cannot move until the end of execution.

Mobile agent migration is achieved through ACL mobile agent messages encoded to XML, which convey mobile agents as the content of a message. Mobile agent message contains the data state and the code of an agent. The data state of mobile agent include general information about mobile agent , also the tasks that mobile agent will performed in certain host. The data state and code will be wrapping up into an ACL message and transmitted to a remote host trough Agent Communication Channel. Mobile agent migration based on ACL messages is simple and effective for agent migration in FIPA compliant systems because these systems have mandatory mechanisms for message communication, transmission and procession.

4. SYNCHRONIZATION SUPPORT IN MOBILE-C

In a Mobile-C agency, mobile agents are executed by independent AEEs. A user might also need to design a multi-threaded application where a Mobile-C agency itself is one of the many threads that handle different tasks. The Mobile-C library supports synchronization among mobile agents and threads. The synchronization API functions are used to protect shared resources as well as to provide a method of deterministically timing the execution of mobile agents and threads.

The API functions can be called from the binary or mobile agent space to initialize the synchronization variables and access them by their unique identification numbers in the list. As opposed to traditional synchronization variables, a Mobile-C synchronization variable is an abstract variable. Once it has been initialized, it may be used as a mutex, condition variable, or semaphore. No further function calls are necessary to change a generic synchronization variable to one of the types. However, once a synchronization variable is used as a mutex, condition variable, or semaphore, it should not be used again as a different type. The example below demonstrates the ability of a Mobile-C mutex to protect a resource that may be shared between two agents.

Any real or imaginary resource that should not be accessed simultaneously by more than one entity at a time should be guarded by a mutex. The resource may be a shared variable, or something more abstract such as control of a robot arm. If there is only one robot arm, then only one entity, an agent in this case, should be able to control it at a time. In the following example, the tasks of agents include exploring an XML file and finding some information. The XML data files store information about the book, the borrower of this book and time of keeping it. A mobile agent dispatched by an agency in the host fsts1 visits remote host fsts2 and fsts3. Figure6 shows part of the mobile agent message sent from host fsts1 to host fsts2and host fsts3. The agent transports three kinds of information. First, information about the mobile agent including the name, the owner and the home address. Second, global information about the task it has to do. The statement <TASK task="2" num="0"> shows that this mobile agent has two tasks to perform and no task has been finished yet. Third, overall information about the task including the name of the task's return variable, the persistence of the agent, the completeness of the task, the host to perform the task, and most importantly, the mobile agent code is C/C++ source code that implements the task. Since the persistent is set to 1, the agent will not be removed from an agency once its code has been executed.

```
<NAME> mobileagent </NAME>
<OWNER> fsts </OWNER>
<HOME> fsts1fsts.ac.ma:5125 </HOME>
<TASKS task= "2" num= "0">
   <DATA num ="0"
            name = "results_fsts2"
            persistent="1"
            complete = "0"
            server = "fsts2.fsts.ac.ma.5138">
<AGENT_CODE>
Mobile agent code on fsts2
</AGENT_CODE>
</DATA>
    <DATA num ="0"
```

```
            name = "results_fsts3"
            persistent="1"
            complete = "0"
            server = " fsts3.fsts.ac.ma.5135">
  <AGENT_CODE>
  Mobile agent code on fsts3
  </AGENT_CODE>
   </DATA>
  </TASK>
```

Figure 6: The template of the mobile agent message sent from the host fsts1 to fsts2 and fsts3.

The task of the mobile agent"mobileagent1" and "mobileagent2" in both host fsts2 and host fsts3 is to access an XML system book information file listed in figure 7, and find the date of sortie and return the book. Function parseNode () is a typical C XML processing program. This function searches each node of the XML file and retrieves the date of sortie and return of the book. Also using a variable synchronization in order to guarantee that data of mobile agent is only accessed by one agent on time.

```
<?xml version="1.0" ?>
<!DOCTYPE  SYSTEM BOOK "Book.dtd">
<Book>
<Title>Les réseaux </Title>
<Author> A.Tanebaum </Author>
<Price>250 </Price>
<LoanList>
<Loan>
<borrower>Tarek Amine </borrower>
<Output>25/09/2011</Output>
<Return>02/11/2011</ Return >
</Loan>
</LoanList>
</Book>
```

Figure 7. The content of an XML system book file.

As shown in Program 1, the mobile agent mobileagent1 performs a ParseNode operation in the host fsts2, it's includes locking the mutex via the function mc_MutexLock() to guaranteed that the desired service and the agent providing the service are protected and the access of the desired service cannot be simultaneously. After that, finding the date of sortie the book by calling parseNode () through the function mc_CallAgentFunc(), and unlocking the mutex via the function mc_MutexUnlock(). After visiting the host fsts2, the mobile agent "mobileagent1" visit the host fsts3 which perform the same task as on the host fsts2. The result obtained from the host fsts2 is sent to the host fsts3 and the return data will be included in the sub-element DATA_ELEMENT.

```
<?xml version="1.0"?>
<MESSAGE message="MOBILE_AGENT">
 <MOBILE_AGENT>
  <AGENT_DATA>
   <NAME>mobileagent1</NAME>
   <OWNER>fsts</OWNER>
   <HOME> fsts1.fsts.ac.ma:5125 </HOME>
   <TASK task= "2" num= "0">
   <DATA number_of_elements ="0" name =
"results_fsts2"
     complete = "0" server = "fsts2.fsts.ac.ma:5138">
   <DATA_ELEMENT> </ DATA_ELEMENT >
    <AGENT_CODE>
     <![CDATA[
#include <stdlib.h>
#include <string.h>
int main() {
 int i, numService = 1, mutex_id = 55, *agentID,
numResult;
 char *funcname = "ParseNode", **service,
**agentName, **serviceName;
 MCAgent_t agent;
 service = (char **)malloc(sizeof(char *)*numService);
 for(i=0; i<numService; i++) {
  service[i] = (char
*)malloc(sizeof(char)*(strlen(funcname)+1));
 }
 strcpy(service[0], funcname);
 mc_SearchForService(service[0], &agentName,
&serviceName, &agentID, &numResult);
 if(numResults < 1) {
  /* No agent is found to have provided such a service. */
  mc_RegisterService(mc_current_agent, service,
numService);
 }
 else {
  /* An existing agent is found to have provided such a
service. */
  mc_MutexLock(mutex_id);
  mc_DeregisterService(agentID[0], service[0]);
  mc_RegisterService(mc_current_agent, service,
numService);
  mc_MutexUnlock(mutex_id);
  mc_DestroyServiceSearchResult(agentName,
serviceName, agentID, numResult);
 }

 for(i=0; i<numService; i++) {
  free(service[i]);
 }
 free(service);
 return 0;
}
void ParseNode (xmlDocPtr doc,xmlNodePtr cur) {
static int i;
i++;
while(cur!=NULL);{
  if(cur->type==XML_ELEMENT_NODE){
   if(!(xmlStrcmp(cur-name,(const xmlChar*)"Sortie"))){
       results-iel2[1]=atof(xmlNodeListGestring(doc,
            cur->xmlChildrenNode,1));
       printf(" the date of sortie of the book is
             %f\n",results_fsts2[1];
```

```
        }
        parsenode(doc,cur->xmlchildrenNode)
        }
cur = cur->next;
}
i--;
return 0;
}
     ]]>
    </AGENT_CODE>
</DATA>
<DATA number_of_elements ="0" name = "results_fsts3"
      complete = "0" server = "fsts3.fsts.ac.ma:5135">
<DATA_ELEMENT> </ DATA_ELEMENT >
<AGENT_CODE>
     <![CDATA[
#include <stdlib.h>
#include <string.h>
int main() {
  int i, numService = 1, mutex_id = 55, *agentID,
numResult;
  char *funcname = "ParseNode", **service,
**agentName, **serviceName;
  MCAgent_t agent;
  service = (char **)malloc(sizeof(char *)*numService);
  for(i=0; i<numService; i++) {
    service[i] = (char
*)malloc(sizeof(char)*(strlen(funcname)+1));
  }
  strcpy(service[0], funcname);
  mc_SearchForService(service[0], &agentName,
&serviceName, &agentID, &numResult);
  if(numResults < 1) {
    /* No agent is found to have provided such a service. */
    mc_RegisterService(mc_current_agent, service,
numService);
  }
  else {
    /* An existing agent is found to have provided such a
service. */
    mc_MutexLock(mutex_id);
    mc_DeregisterService(agentID[0], service[0]);
    mc_RegisterService(mc_current_agent, service,
numService);
    mc_MutexUnlock(mutex_id);
    mc_DestroyServiceSearchResult(agentName,
serviceName, agentID, numResult);
  }

  for(i=0; i<numService; i++) {
    free(service[i]);
  }
  free(service);
  return 0;
}
void ParseNode (xmlDocPtr doc,xmlNodePtr cur) {
static int i;
i++;
while(cur!=NULL);{
   if(cur->type==XML_ELEMENT_NODE){
     if(!(xmlStrcmp(cur-name,(const xmlChar*)"Sortie"))){
          results-iel2[1]=atof(xmlNodeListGestring(doc,
                cur->xmlChildrenNode,1));
          printf(" the date of sortie of the book is
                 %f\n",results_fsts3[1];
         }
         parsenode(doc,cur->xmlchildrenNode)
         }
cur = cur->next;
}
i--;
return 0;
}
     ]]>
    </AGENT_CODE>

  </AGENT_DATA>
</DATA>
</TASK>
```

Program1.The task of mobile agent 1 performed on the host fsts2 then the host fsts3

Likewise, the task of the mobile agent “mobileagent2” is locks the mutex, find the date of return the book and unlocks the mutex on both the host fsts2 and fsts3.

5. CONCLUSION

This paper presents an XML-based approach for agent communication and migration of distributed application in mobile-C. Mobile-C conforms to the IEEE FIPA standards, it’s integrates an embeddable C/C++ interpreter into the platform as a mobile agent execution engine in order to support mobile agents. Mobile agents, including its data state and code, are carries to a remote agent platform via ACL messages which will be encoded in XML. Our work shows that using XML to encode different types of messages is simple, and easy to change .Thus, the synchronization functions guaranteed the protection of shared resources by using the mutex in multiple hosts.

6.REFERENCES

[1] K.Straber, J.Baumann and F.Hohl.. Mole - A Java Based Mobile Agent System. Institute for Parallel and Distributed Computer Systems, University of Stuttgart,1997

[2] D. Lange, M.Oshima. Programming and Deploying Java Mobile Agents with Aglets. Addison-Wesley: MA, 1998.

[3] D.Wong, N.Paciorek, T.Walsh, J.DiCelie, M.Young, B.Peet. Concordia: An infrastructure for collaborating mobile agents. Proceedings of the First International Workshop on Mobile Agents (MA’97) (Lecture Notes in Computer Science, vol. 1219). Springer: Berlin, 1997; 86–97.

[4] F.Bellifemine, G.Caire, A.Poggi, G.Rimassa.JADE: A software framework for developing multi-agent

applications.Lessons learned. Information and Software Technology 2008; 50(1–2):10–21.

[5] R.Gray, G.Cybenko, D.Kotz,R.Peterson, D.Rus. D'Agents: Applications and performance of a mobile-agent system. Software—Practice and Experience 2002; 32(6):543–573.

[6] H.Peine. Run-time support for mobile code. PhD Dissertation, Department of Computer Science, University of Kaiserslautern, Germany, 2002.

[7] H.Peine .Application and programming experience with the Ara mobile agent system. Software—Practice and Experience 2002; 32(6):515–541.

[8] D.ohnansen, K.Lauvset, R.V.Renesse, F.B. Schneider, N.P. Sudmann, K. Jacobsen. A TACOMA retrospective.Software—Practice and Experience 2002; 32(6):605–619.

[9] MACE—Mobile agent code environment. Available at: http://wwwagss.informatik.uni-kl.de/Projekte/Ara/mace.html .

[10] N.P.Sudmann,D.Johansen. Adding mobility to non-mobile web robots. Proceedings of the IEEE ICDCS00 Workshop on Knowledge Discovery and Data Mining in the World-wide Web, Taipei, Taiwan, 2000; 73–79.

[11] B.Chen, H.H.Cheng. A run-time support environment for mobile agents. Proceedings of ASME/IEEE International Conference on Mechatronic and Embedded Systems and Applications, No. DETC2005-85389, Long Beach, CA, September 2005.

[12] B.Chen,H.H.Cheng,J.Palen. Mobile-C: A mobile agent platform for mobile C/C++ agents. Software—Practice and Experience 2006; 36(15):1711–1733.

[13] B.Chen, D.Linz, H.H.Cheng. XML-based agent communication, migration and computation in mobile agent systems. Journal of Systems and Software 2008; 81(8):1364–1376.

[14] Mobile-C: A multi-agent platform for mobile C/C++ code.2009.

[15] H.H.Cheng. Scientific computing in the Ch programming language. Scientific Programming 1993; 2(3):49–75.

[16] H.H.Cheng.Ch: A C/C++ interpreter for script computing. C/C++ User's Journal 2006; 24(1):6–12.

[17] Ch—An embeddable C/C++ interpreter.2011

[18] B.Chen, H.H.Cheng, J.Palen. Integrating mobile agent technology with multi-agent systems for distributed traffic detection and management systems, Transportation Research Part C 17 (2009) 1–10