Annals of the University of Craiova, Mathematics and Computer Science Series
Volume 37(1), 2010, Pages 71–79
ISSN: 1223-6934

A Proposal for an Enhanced Mobile Agent Architecture (EMA)

Gabriel Stoian and Claudiu Ionuţ Popîrlan

Abstract. Mobile agents become increasingly important in the highly distributed applications frameworks seen today. Their migration from node to node is a very important issue as we need to safeguard application efficiency, achieve better load balancing and resource utilization throughout the underlying network. This paper presents an enhanced mobile agent architecture based on the regular one by adding some components related to synthetic information about past activities of mobile agent and which subsequently can lead to an improvement of its future activities. We prove the benefits of the proposed architecture considering an adequate case study and providing experimental results.

2010 Mathematics Subject Classification. Primary 68T42; Secondary 68T05.
Key words and phrases. Enhanced Mobile Agents, Recorded Agent Information, Master-Slave hierarchy, Tracy toolkit.

1. Introduction

The concept of mobile agent has been defined in [1], [2], [4]. They are autonomous objects that can migrate from node to node on behalf of the user who have executed them and make use of the databases or computation resources from computers connected by the network. In order for a mobile agent to be able to migrate, there must be a virtual place, the so-called mobile agent system, that supports mobility.

The mobile agents technology promotes applications made up of network-aware entities (agents) capable of changing their execution environment by transferring themselves while executing (migration). However, mobile agents technology has failed to become a sweeping force of change, and now faces competition in the form of remote procedure call (RPC) technologies([12]).

The current interest in mobile agents is broadly justified by the advantages their presence provides in Internet applications:

- mobile agents can carry the code to manage remote resources and do not need the remote availability of a specific server, thus leading to a more flexible application scenario; Instead of moving large amounts of data to a single point where it is searched, information retrieval moves the data-searching code to the data. In [5] it was described a new type of search engine, in which Web pages are analyzed locally by a mobile agent that was sent to the Web server. The agent only sends back a summary of the Web pages, and therefore might reduce network traffic considerably.
- mobile agents can dramatically save bandwidth, by moving locally to the resources they need, instead of requiring the transfer of possibly large amounts of data;

Received December 20, 2009. Revision received February 09, 2010.

- mobile agents do not require continuous network connection, because interacting entities can move to the same site when the connection is available and then interact without needing further network connection; as a consequence mobile agents intrinsically suit mobile computing systems;

Significant research and development into mobile agency has been conducted in recent years, and there are many mobile agent architectures available today. The AgentLink project [11] maintains a list of ongoing projects with regard to any kind of agent-related topics and also maintains a list of available agent toolkits. Nevertheless, several issues still need to be faced to make the mobile agent technology widely accepted: secure and efficient execution supports, standardization, appropriate programming languages and coordination models. The appeal of mobile agents is quite alluring - mobile agents roaming the Internet could search for information, find us great deals on goods and services, and interact with other agents that also roam networks (and meet in a gathering place) or remain bound to a particular machine. However, research so far, does not exploit enough previous activities experience to make mobile agents more efficient. For example, in [6] was introduced a mobile agent migration technique based on decision tree learning for P2P networks. The proposed solution has the objective of an agent based routing algorithm (ABRA) which tries to give the best path between two nodes ([7]).

In this paper a new approach regarding mobile agents architecture is presented. Starting from the basic structure of mobile agents was developed an enhanced mobile agent architecture which was validated on a case study.

2. Enhanced Mobile Agents Architecture

Mobile agents consist of three components: *code*, *data*, and *execution state*. *The code* contains the logic of the agent, and all agents of the same type use the same code. The code must be separated from the code of the agency so that it can be transferred alone to another agency, and the code must be identifiable and readable for an agency (e.g., in the form of a file from the local file system or a byte stream from the network). Usually, as in other programs, an agent's code consists of more than one file (e.g., in the Java programming language they could be many class files).

The second component of an agent is *data*. This term corresponds to the values of the agent's instance variables if we assume an agent to be an instance of a class in object-oriented languages. The data is sometimes also called the object state. It is important to note that not all data items an agent can access are part of its object state. Some variables reference objects that are shared with other agents or the agency software itself, for example, file handlers, threads, the graphical user interface, or other resources and devices that cannot be moved to other servers. Thus, we have to restrict the agent's immediate data to those data items the agent owns and that are movable.

The third component is the *execution state*. The difference between object and execution state information is that the elements of the object state are directly controlled by the agent itself, whereas execution state information is usually controlled by the processor and the operating system. What this means depends very much on the decision of the mobile agent toolkit designer and the underlying execution environment (processor, operating system, virtual machine). In some toolkits, an agent's execution state is comprised of the current value of the instruction pointer and the

stack of the underlying processor. In others it is not possible to determine the execution state of an agent at all. In most Java-based toolkits, for example, the agent itself is responsible for copying information about its current execution state on the level of the programming language into the object state and restoring it after successful migration.

The proposed *Enhanced Mobile Agents (EMA)* architecture improves the basic structure of mobile agents and does not implement an entire new mobile agent. Instead, it has been designed to complete the functionality of already available agent systems, and it is not bound to any specific implementation: it can be associated to different Java-based mobile agent systems with only a slight extension.

FIGURE 1. Enhanced Mobile Agent Architecture (EMA)

Within basic mobile agents structure we add the *Recorded Agent Information (RAI)* component, as shown in Figure 1, in order to register certain synthetic information (i.e. registration of visited nodes, successful queries, time calculations, etc.) according to defined *Record()* method. This component is intended to act like an aeronautical "black-box" which can provide a lot of information that can be used to improve and/or correct the functioning of airplanes. Similarly, RAI will be used to improve/adjust the behavior of enhanced mobile agents by certain methods included in agent code.

One of the main advantages of the proposed architecture is versatility which allows defining, trying, and tuning of agent functionality by designing and updating the implementation of the code which process RAI. As an example, RAI can be used to feed learning methods in order to control the activity of mobile agents. Resuming, the developer must perform following actions:

- Design and implement of the RAI collecting method
- Design and implement of the RAI processing algorithm
- Design and implement of the agent control mechanism

When all agents of a mobile agents system are similar, from the hierarchical perspective, RAI component is not apprehended as a benefit. The real advantage of having RAI component become obvious in a hierarchical approach, when more Slave

Agents are subordinate to a Master Agent – this case will be treated in an experimental study (Section 3).

2.1. Master Agent. Slave Agent. EMA advantages can be highlighted very clearly in case of master/slave mobile agents system. This system includes one Master Agent and several Slave Agents. The Master Agent receives complex requirements (interrogations) and create specific Slave Agents in order to process parts of a certain complex activity. In this case, the Master Agent may use RAI component to adapt its behavior and/or the actions directed to Slave Agents in order to improve the solving of assignments. Also, Master Agent can use its RAI component for the same purpose and at the same time can provide access to it. EMA architecture for both Master and Slave Agents is presented in Figure 2.

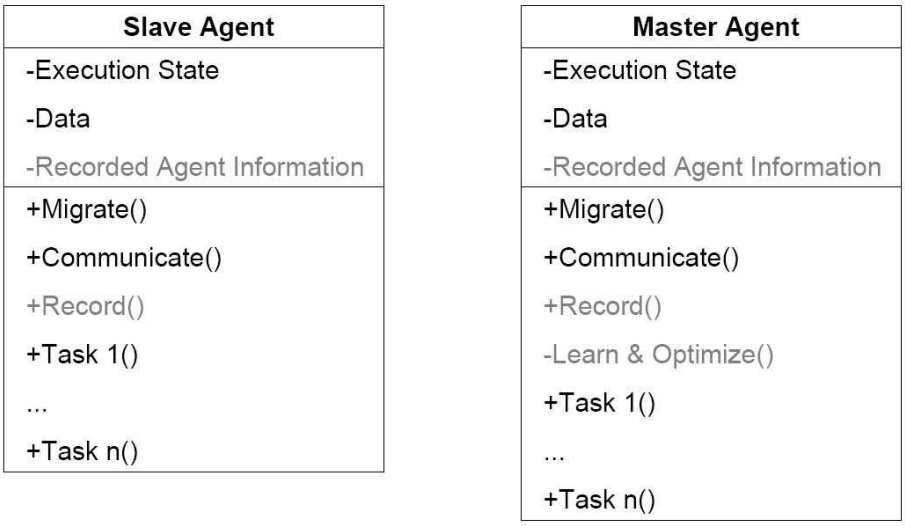

FIGURE 2. Master/Slave Mobile Agents

In the next two examples we look at the implementation proposed for a Slave Agent and a Master Agent, using Tracy ([2]) mobile agent toolkit, which is implemented in the Java programming language.

```
//SlaveAgent.java
import de.fsuj.tracy.agent.*;
import java.io.Serializable;
import de.fsuj.tracy2.plugins.migration.interfaces.IAgentMigrationContext;
import de.fsuj.tracy2.plugins.migration.interfaces.IMonitor;

public class SlaveAgent extends Agent implements Runnable, Serializable{
    //...
protected abstract void Migrate(){
    // Defines how the agent should be transferred to the next destination.
    Context.getContext("migration");
    //...
    migrate(destination(), new int[] {units[0]}, definedDataItems, useCache());
    //...
 }
```

```
 private void Communicate(){
     //Define communication mode
     IAgentMessageContext.getContext( "message" );
     //...
 }
 private void Record(){
      //Defines how the agent should record information
     IAgentMessageContext.getContext( "recorded" );
     if( domainNodes != null ){
         shellCxt.writeToUser( "Known domain nodes are:" );
         for( i=0; i < domainNodes.length; i++){
             shellCxt.writeToUser( domainNode[i].toString() );
         }
     }
     //...
 }
 public void Task1(){
    // Define Task1 for Slave Agent
 }
 //...
 public void Taskn(){
    // Define Taskn for Slave Agent
 }
}
//MasterAgent.java
import de.fsuj.tracy.agent.*;
import java.io.Serializable;
import de.fsuj.tracy2.plugins.migration.interfaces.IAgentMigrationContext;
import de.fsuj.tracy2.plugins.migration.interfaces.IMonitor;

public class MasterAgent extends Agent implements Runnable, Serializable{
    //...
public MasterAgent(){
    // do some initialization
 }
public void startAgent(){
    // do something
 }
protected abstract void createSlaveAgent(){
    // Define how the Master Agent creates Slave Agents
 }

protected abstract void Migrate(){
    // Define how the agent should be transferred to the next destination
 }
private void Communicate(){
    //Define communication mode
 }
private void Record(){
    //Define how the agent should record information
 }
protected abstract void LearnOptimize(){
    // Define learning algorithm and optimization method;
    DecisionTreeBuilder builder =
```

```
       new DecisionTreeBuilder(learningSet, testAttributes, goalAttribute);
     DecisionTree tree = builder.build().decisionTree();
//...
 }
public void Task1(){
    // Define Task1 for Master Agent
 }
//...
public void Taskn(){
    // Define Taskn for Master Agent
 }
}
```

3. Case Study and Experimental Results

To evaluate the efficiency of Enhanced Mobile Agents (EMA) architecture we consider the following scenario: *A romanian client wants to travel in three major cities of Europe – London, Paris, Rome – comes to the head office of a tour operator from Craiova and asks for information regarding accomodation, restaurants, local attractions, etc.*

The solution is based on four agencies located in the considered cities - the one located in Craiova is *the home agency*. The network topology is shown in Figure 3. Each node provide an execution environment with different hardware configurations and different operating systems (Windows and Linux). The client requirement assumes finding those objectives that meet certain criteria related to the type of accommodation (price, breakfast, Internet, number of beds in the room, hotels classification, refrigerator), local attractions (type, costs), etc. For implementation we used Tracy ([2]) mobile agent toolkit.

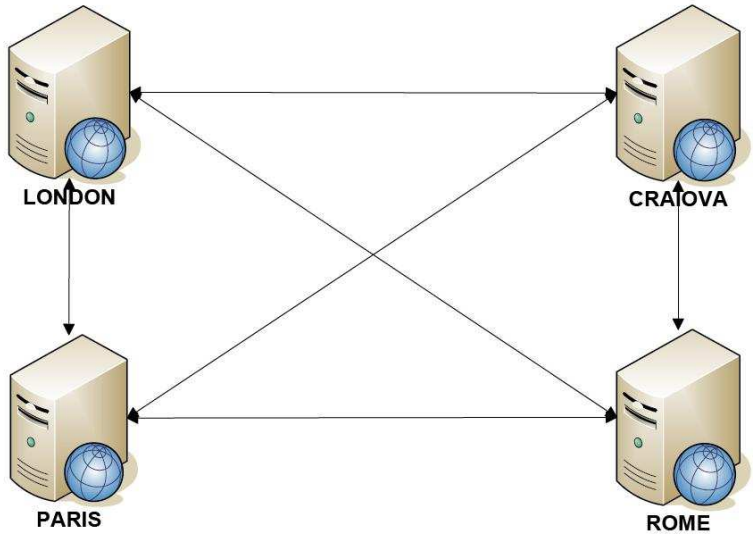

FIGURE 3. Network topology and agencies locations

Initially, the home agency will contain only the Master Agent. Based on the assignment it will create at least one Slave Agent to each of the other agencies. To accomplish their mission, those Slave Agents, will migrate to the appropriate agency, retrieve needed data locally, process it, and send their results. Also, each Slave Agent will record information,

using RAI component, about queries duration, and Master Agent will use this to improve the assignment solving. We choose a simple improvement method which will increase by 1 the number of Slave Agents sent to an agency if the recorded query duration is greater than 1 second (see Alghoritm 1).

Algorithm 1 Assignment Improve

```
for eachSlaveAgent do
  if RecordedQueryDuration > 1 then
    NumberOfSlaveAgents++;
  end if
end for
```

To emulate local data retrieving, on each computer, will be performed SQL queries on relational database which contains records in a table with structure described in Table 1. The number of records in each local database (with the same structure) are: London – 50000, Paris – 70000, Rome – 40000.

TABLE 1. Accommodation Database – Table structure

Id	Name	Type	Class	Price	Breakfast	Internet	Beds_number	Refrigerator

3.1. Experimental results. The results presented in this subsection proves the efficiency of EMA architecture. The assignment implies to process a query that involves finding the facility names that meet the following accomodation conditions:

- Class $\geq$ 3 stars;
- Price $\leq$ 60 €;
- Breakfast = YES;
- Internet = YES;
- Beds_number = 2;
- Refrigerator = YES.

In the first run the Master Agent creates only one Slave Agent for each of the London, Paris, and Rome agencies. The query duration stored in RAI component of each Slave Agent are presented in Table 2.

TABLE 2. Experimental results – First Run

	London	Paris	Rome
Recorded Agent Information (RAI)	Slave Agents		
	London1	*Paris1*	*Rome1*
NumberOfRecords	50000	70000	40000
QueryDuration (sec)	0,5	**1,4**	0,2

The final result which aggregates the Slave Agents results was obtained in 1,9 seconds.

The second run with the same requirements will look different, because based on information recorded in RAI component of the *Paris1* agent, the Master Agent will decide to create two Slave Agents – *Paris1* and *Paris2* – which will co-operate by splitting the records in two ranges. Thus, *Paris1* will process the query in records whose names begin with a letter in the range (A - K), and *Paris2* will process the queries in the range (L - Z). The results of this test are presented in Table 3.

TABLE 3. Experimental results – Second Run

	London	Paris		Rome
Recorded Agent Information (RAI)	Slave Agents			
	London1	*Paris1*	*Paris2*	*Rome1*
NumberOfRecords	50000	20000	50000	40000
QueryDuration (sec)	0,5	0,5	**1,1**	0,2

The final result on second processing of the query was obtained in 1,6 seconds. When trying to understand the values of execution times we have to keep in mind that the mobile agents code is executed quasi-parallel.

Using the same method, at third processing of the query, Master Agent will create three Slave Agents on agency Paris: (*Paris1*, *Paris2* and *Paris3*). Again the Paris-related agents co-operation method will consists of splitting the database record (in three ranges this time). Thus, *Paris1* will process the query in records whose names begin with a letter in the range (A - K), *Paris2* will process the queries in the range (L - S), and *Paris3* will process the queries in the range (T - Z). The results of this test are presented in Table 4.

TABLE 4. Experimental results – Third Run

	London	Paris			Rome
Recorded Agent Information (RAI)	Slave Agents				
	London1	*Paris1*	*Paris2*	*Paris3*	*Rome1*
NumberOfRecords	50000	20000	22000	23000	40000
Query Duration (sec)	0,5	0,6	0,7	0,8	0,2

The final result on third processing of the query was obtained in 1 second.

4. Conclusions and Future Work

In this paper we have introduced Enhanced Mobile Agent (EMA) architecture, based on common mobile agent structure. This architecture has been designed to complete the functionality of already available mobile agents, and it is not bound to any specific implementation. The results obtained can proves that the Mobile Agents efficiency can be effectively improved using EMA.

As future direction of our research, we intend to study the possibility to develop a toolkit based on EMA and to design strong learning algorithms fed with recorded agent information which lead to an increased efficiency in solving assignments. Also, an interesting challenge for the EMA architecture is to prove its qualities in reasoning environments generated by new semantic schema named *deductive path* [3].

References

[1] S. Russell and P. Norvig *Artificial Intelligence: A Modern Approach*, Prentice Hall, 1995.
[2] P. Braun and W. Rossak, *Mobile Agents: Concepts, Mobility Models, & the Tracy Toolkit*, Elsevier Inc.(USA) and dpunkt.verlag(Germany), 2005.
[3] N. Tandareanu and M. Ghindeanu, Path-based Reasoning in Semantic Schemas, *Annals of the University of Craiova - Mathematics and Computer Science Series* **35** (2008), 171–181.
[4] J. Baumann, *Mobile Agents: Control Algorithms*, Lecture Notes in Computer Science, Springer, 2000.

[5] P. Thati, P.-H. Chang and G. Agha, Crawlets: Agents for high performance web search engine, *Proceedings of the 5th International Conference (MA 2001)*, Lecture Notes in Computer Science **2240** (2001), 119-134.
[6] Y. Ali, H. N. Elmahdy and S. Ahmed, Optimizing Mobile Agents Migration Based on Decision Tree Learning, *World Academy of Science, Engineering and Technology* **33** (2007), 326–332.
[7] J.N.K. Liu, B.N.L. Li and T.S. Dillon, An improved naive Bayesian classifier technique coupled with a novel input solution method, *IEEE Transactions on Applications and Reviews* **31** (2001), no. 2, 249–256.
[8] Q. Wenyu, S. Hong and X. Defago, A Survey of Mobile Agent-Based Fault-Tolerant Technology, *Parallel and Distributed Computing, Applications and Technologies* **5** (2005), 446–450.
[9] D. Kotz and R.S. Gray, Mobile Agents and the Future of the Internet, *ACM Operating Systems Review* **33** (1999), no. 3, 7–13.
[10] D. Johansen, R. van Renesse and F.B. Schneider, Operating System Support for Mobile Agents, *Technical report, Department of Computer Science, Cornell University*, USA, 1994.
[11] `www.agentlink.org`
[12] `http://en.wikipedia.org/wiki/Remote_procedure_call`

(Gabriel Stoian) University of Craiova Faculty of Mathematics and Computer Science, Department of Computer Science, 13 Alexandru Ioan Cuza Street, Craiova, 200585, Romania
E-mail address: `gstoian@yahoo.com`

(Claudiu Ionuţ Popîrlan) University of Craiova Faculty of Mathematics and Computer Science, Department of Computer Science, 13 Alexandru Ioan Cuza Street, Craiova, 200585, Romania
E-mail address: `popirlan@inf.ucv.ro`