Volume 5, Issue 10, October-2015 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

Multi Agent System and Distributed Applications Optimized Communication using Mobile Agents

Najat Rafalia, Jaafar Abouchabaka
Ibn Tofail University, Science Faculty, Computer Science Department
LARIT Laboratory, Morocco

Abstract— Abstract----In this paper, we present a mobile implementation of a strategy of communication between agents in a distributed environment. This strategy is used by the □$_{AD}$-calculus witch is an elementary functional distributed agent language [1]. It's based on a static analysis which allows determining the parts of a message that must be transmitted. The agents we consider have a functional script and manipulate the terms of the □$_{AD}$-calculus. The expressions of this language correspond to those of the □-calculus extent by some agent primitives. The mobile implementation of communication strategy witch we present in this paper is based on mobile agents. In the distributed agent system, each host has its mobile agent witch can migrate to the other hosts with the parts of messages, that must be transmitted outside. Mobile agent migrates to the hosts which have a lot of interactions with his host. In the case of the low interactions, this approach become less efficient, it's penalized by the size of the mobile agent. In this second case, the messages towards distant sites are addressed to the server agent of the sender site in order to treat and send them to their destination. In the receiver site, the server agent treats the external messages that arrive and transmit them to their receivers.
This work combines into a global system for the valuation of the agent languages through a distributed virtual machine AGDVM using the mobile agents towards a new distributed agent development framework.

Key terms: Multi Agent System, Mobil Agent, Communication, Distributed System, Distributed Programming.

I. INTRODUCTION

We consider a distributed system of agents. The agents we consider have a functional script and manipulate the terms of the □$_{AD}$***-calculus*** which is an elementary functional distributed agent language [1], described in section3. The expressions of this language correspond to those of the□ □-calculus extent by some agent primitives.
Agents communicate by exchanging messages. A message is a functional term. The message transmission causes the message address to be put in the receiver mail queue. This implantation doesn't involve any problem when the agents are in the same site because they share the common memory. If the agents are in some distant sites, they could not get to each other site memory. So, sending the message reference is insufficient.
We have integrated in the □$_{AD}$***-calculus*** a lazy strategy of message communication between agents [1]. For each agent susceptible of receiving a message m which is a functional term, we determine the part of m, that must be sent. This is accomplished by a static analysis of the application code.
In this paper, we present a mobile implementation of this communication strategy, based on mobile agents. In the distributed agent system, each host has its mobile agent witch can migrate to the other hosts with the parts of messages, that must be transmitted outside.
Mobile agent is a soft entity that can migrate during the execution from host to other host. It migrates with its code, its execution state and its data. The mobility is controlled by the application not by the execution system. Le but of the migration is to reach distant data and resources, to do the treatment locally and to move just necessary data.
So, mobile agent migrates to the hosts which have a lot of interactions with his host. In the case of the low interactions, this approach become less efficient, it's penalized by the size of the mobile agent. In this second case, the messages towards distant sites are addressed to the server agent of the sender site in order to treat and send them to their destination. In the receiver site, the server agent treats the external messages that arrive and transmit them to their receivers.
Using mobile agent involves the optimization of the traffic because exchanged messages become locals and discharge the network. Mobil agents can use the resources of distant hosts, so, it discharges its initial host. The size of the mobile agent adds a surcharge when it's migrating but this charge is compensated by local interactions.

II. MULTI AGENT SYSTEM

Multi-agent systems programming languages, platforms and development tools are important components that can affect the diffusion and use of agent technologies across different application domains. In fact, success of multi-agent systems is largely dependent on the availability of appropriate technology (i.e. programming languages, software libraries and development tools). Multi-agent system can be realized by using any kind of programming language.

A. Concept of an agent

Agents are considered one of the most important paradigms that on the one hand, may improve on current methods for conceptualizing, designing and implementing software systems. On the other hand, they may be the solution to the legacy software integration problem [1].

In general, an agent is an autonomous entity that performs one or several tasks in order to achieve some goals. In the domain of networking, an agent can run even if the user disconnects from the network. Some agents run on dedicated servers, others work on standard platforms. Agent can act on behalf of users to collect, filter, and process information. They can act autonomously and react to changing environments.

There are several definitions of agent, the most adopted is that (Jennings, Sycara and Wooldrige, 1998) defined "agent is a computer system that is situated in some environment, and that is capable of autonomous action in this environment in order to meet its design objectives" [2].

Genesereth and Ketchpel, 1994; Wooldrige and Jenning, 1995; Russell and Norvig, 2003), all definitions agree that an agent is essentially a special software component that has autonomy that provides an interoperable interface to an arbitrary system and/or behaves like a human agent, working for some clients in pursuit of its own agenda.

An agent is a component of software and/or hardware which is capable of acting exactingly in order to accomplish tasks on *behalf of its user (Nwana[3], 1996).*

For me, an agent is a piece of program has standard characteristics intends to achieve its predefined tasks through its environment.

B. Characteristics of an agent

An agent has several characteristics. I will identify three characteristics: **autonomy, learning or adaptive, and cooperative.**

1) Autonomy: It refers to the characteristic that an agent can operate on its own without the need for human guidance. In other words, an agent is independent and makes its own decisions; a concequence of the agents being autonomous that leads the system tends to be decentralized [2]. For example, when a message is sent to an agent, the agent (being autonomous) as control over how it deals with the message. In order to do so, an agent has to be proactive in the sense that it has the ability to take the initiative rather than acting simply in response to its environment [4].

Autonomous regards to agent means that they decide for themselves whether or not to perform an action on request from another agent. They are capable of a flexible behavior, and each agent of a system has its own thread of control.

2) Cooperative: Interaction with other agents is necessary to accomplish a complicated task. In order to cooperate however, an agent must possess a social ability that allows it to interact with other agents. Agent interaction is often viewed in terms of human interaction types such as negotiation, coordination, cooperation.

3) Adaptive: An entity, by sensing and acting upon its environment, tries to fulfill a set of goals in a complex, dynamic environment. Properties: it can sense the environment through its sensor and act on the environment through its actuators; it has an internal information processing *and decision making capability; it can anticipate future states and possibilities, based on internal models (which are often incomplete and/or incorrect); his anticipatory ability often significantly alters the aggregate behavior of the system of which an agent is part.*

C. Classification of Software agents

Study the types and entities of existing software agents is refer to classification. It can be classified according to: the tasks they perform; their control architecture; the range and effectiveness of their actions; the range of sensitivity of their senses; or how much internal state they posse [5]; all that are based on Nwana [3] in which used a set of criteria to classify agents, these criteria emerged some types of agents: **Collaborative**, **Interface**, **Mobile**, **Reactive**, **Intelligent**.

Study the types and entities of existing software agents is refer to classification. It can be classified according to: the tasks they perform; their control architecture; the range and effectiveness of their actions; the range of sensitivity of their senses; or how much internal state they posse [5]; all that are based on Nwana [3] in which used a set of criteria to classify agents, these criteria emerged some types of agents: **Collaborative**, **Interface**, **Mobile**, **Reactive**, **Intelligent**.

1) Collaborative: A collaborative agent is a software program that helps users solve problem, and taking care of low-level details.

2) Interface: Interface agents support and provide assistance to a user learning to use a particular computer application by employ artificial intelligence techniques. Interface agents learn to assist users by through the following ways in which it observing and imitating the users: receiving positive and negative feedback from the user; receiving explicit instructions from the user and by asking other agents for advice.

3) Mobile: A mobile agent is a computer entity with ability to travel between different nodes in a computer network, capable of reasoning, use the network infrastructure to run in another remote site, search and gather the results, cooperate with other sites and return to his home site after completing the assigned tasks.

4) Reactive: responds (in a timely manner!) to changes in its environment.

5) Intelligent: An agent is intelligent if it is able to learn and sense, acts and reacts to its external environment. Agent should be blessed with artificial intelligence techniques [6] to enhance its characteristics like autonomy, cooperation, and learning. The goals of an intelligent agent is to be robust in case of failure of actions in challenging environments, agents must be able to recover its state instead of waste time trying to follow plans that are no longer relevant or applicable.

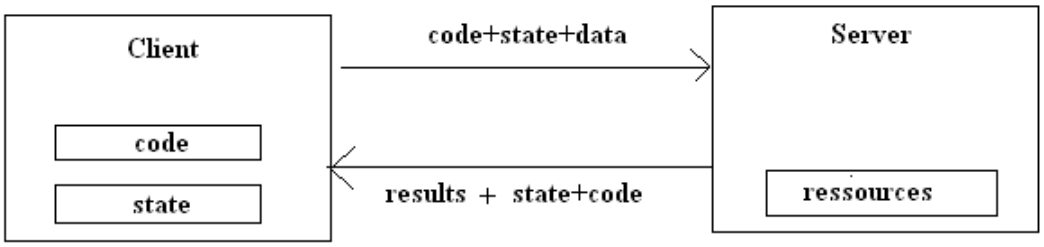
Those type of agent result to combine two or more of a set of acquired characteristics which explained above.

III. MOBILE AGENT

Mobile agent is a composition of computer software and data which is able to migrate (move) from one computer to another autonomously and continue its execution on the destination computer.

More specifically, a *mobile agent* is a processing that can transport its state from one environment to another, with its data intact, and be capable of performing appropriately in the new environment. Mobile agents decide when and where to move. Movement is often evolved from RPC methods. Just as a user directs an Internet browser to "visit" a website (the browser merely downloads a copy of the site or one version of it in the case of dynamic web sites), similarly, a mobile agent accomplishes a move through data duplication. When a mobile agent decides to move, it saves its own state, transports this saved state to the new host, and resumes execution from the saved state.

A mobile agent is a specific form of mobile code, within the field of code mobility However, in contrast to the Remote evaluation and code on demande programming paradigms, mobile agents are active in that they can *choose* to migrate between computers at any time during their execution. This makes them a powerful tool for implementing distributed applications in a computer network.



picture 1 :mobile agent

Avantages

Some **advantages** which mobile agents have over conventional agents are:

- Computation bundls -converts computational client server round trips to relocatable data bundles, reducing network load.
- Parallel processing -asynchronous executing on multiple heteregenous network host.
- Dynamic adaptation – actions are dependent on the state of the host environment.
- Tolerant to network faults - able to operate without an active connection between client and server flexible maintenance to change an agent's actions, only the source (rather than the computation hosts) must be updated.
- Bandwidth conversion which converting the bandwidth from one host to another host.
  One particular advantage for remote deployment of software includes increased portability there by making system requirements less influential.

□AGD-CALCULUS: AN ELEMENTARY FUNCTIONAL DISTRIBUTED AGENT LANGAGE

The □*AGD-calculus* is a distributed extension of the □AD-calculus, for the agent model, described in [1]. The □-calculus is an elementary functional language [4].The □AGD-calculus constitutes a low level functional distributed agent language to which the high level agent languages could be compiled. The new version of the □AGD-calculus integrates the instruction **migrate(Hi)** allowing the migration of an agent from host to another site.

To represent data and the messages, □AD-calculus integrates a structure of term which corresponds to a tree. It also integrates a pattern-matching mechanism in order to recognize the messages.

The agent behavior is an expression of the □AD-calculus extended by the primitives for the creation and the manipulation of the agents and those for the pattern matching and the construction of the terms.

□ □ □ □AGD-calculus Syntax

The □AGD-calculus expressions are constructed by the terms of the algebra engendered by a set of constructors, a set of variables and the mechanisms of abstraction and application.

The abstraction on a variable of the □AGD-calculus is generalized to an abstraction on a pattern which is a term of the algebra engendered by the constructors. The □ □AGD-calculus contains also the primitives **send**, **create, become** and **migrate** for the manipulation of the agents.

- **send(a,m)** to send the message **m** to the agent **a.**
- **become(b)** to replace the behavior of the agent executing this primitive, by the behavior **b.**
- **create(b)** to create a new agent with an initial behavior **b**.
- **migrate(Hi)** to migrate to a host number i

In the expressions of the □AGD-calculus, it's necessary to distinguish the agent behaviors which could contain functional computations, the primitives **send** and **become**, from the messages or communicable values which are the results of a pure functional computation. These values could be described by the following syntax:

m = x variables, symbols, address, numbers
| C(m,m,….,m) construction of terms
|Create(B{Acquaintances}) creation of an agent

| self the individual agent address

Note that the messages must be partially valued in order to be filtered. The mechanism of patter-matching takes charge of this lazy valuation.

The result of the **create** operation is an agent address which is a communicable value.

The **self** variable designates implicitly the agent which executes the behavior and allows to this agent to send to himself a message.

The behaviors have the following syntax:

B{acquaintances}= □**P.Fa** abstraction on a pattern

|□**P.Fa,** □**P.Fa, …..,** □**P.Fa** composition of abstractions on several patterns

The actions **Fa** have the following syntax:

Fa = send(**m**,**m**); **Fa**

| create(B{f'{acquaintances}) ; **Fa**

| become(B{acquaintances})

| migrate(Hi)

Example 1*:* **consultation and change of the cell value**

The following behavior **Cell{v}** is a behavior of an agent **Cell** which sends the initial value "**v**" to an agent **a** when it receives the message **Pair(Get,a)**. When **Cell** receives the message **Pair(Set,n)**, it changes its initial value "**v**" by the value "**n**"**.**

Cell{v} = □Pair(Get, a). send(a,v); become(Cell{v})

□Pair(Set, n). become(Cell{n})

The following expression creates an agent 'A' and sends him the message pair(Set,4) :

A = create(Cellf{0}), send (A, Pair(Set,4))

B. Lazy strategy of communication

The execution of a transmission **send(a,m)**, consists of the valuatio of the receiver agent **a** and that of the message **m**.

If the valuation of the agent **a,** detects that this later is in a distant site, then, the transmission of the message address is not sufficient because distant agents could not get to each other site memory. In this case we opt for a lazy transmission between distant sites.

The general problem is presented as follow:

Given a message C(C_1,;C_2;..., C_n) destined to an agent **a**, *what are in this message the necessary levels to accomplish the pattern-matching and the treatment of the message by the agent* **a**.

Our lazy communication strategy consists of two phases:

a. **Static analysis phase**: it's accomplished at the time of the compilation. It allows determining the parts of the message, that are necessaries for the pattern-matching and the treatment of this message. These parts are expressed in the level number of the tree which represents the message.

b. **Dynamic transmission phase**: because the static analysis is not always informative, several parts of the message are not detected necessary at the time of the compilation. So, this phase allows completing these needs in the execution.

Static Analysis

The static analysis concerns in fact, all the patterns of the application behaviors. The analysis of an initial behavior involves, through the **become**primitive, to analysis the replacement behaviors starting by this initial behavior. It consists of four principal steps:

- **Marking**: through each pattern, the marking phase marks the necessary parts in a message which will be filtered by this pattern and treated by the action corresponding to the successful pattern-matching. An action corresponds to a sequence of several **send** and several **create**, ended by a **become**.
- **Flattening :** marking is don behavior by behavior. A part can be necessary in a behavior and not necessary in other one. The flattening step allows to "**flatten"** the results of marking concerning the same pattern which appears in an initial behavior and in other replacement behaviors from this initial behavior. The ended set of replacement behaviors can be determined through the application code. This warrants the termination of our algorithms.
- **Compilation of the patterns**: the necessary in a pattern is expressed by a number of levels. This phase consists of associating to each pattern the number of its necessary levels.
- **Compilation of the send**: the ad-equation between the patterns and the messages **m** figuring in the **send(a,m)**, and the use of the precedent phase results, allow to compile the transmissions **send(a,m)** into **send(a,m,L)**, where **L** is the number of **m** levels which are necessaries to accomplish the pattern-matching and the treatment of **m** by **a**.

The detail of the static analysis phases is presented in [1].

The static analysis determines for each class of agents having the same initial behavior $\mathbf{b_i}$, the number of necessary levels in a message **m** which can be filtered and treated by the behavior $\mathbf{b_i}$.

We group the different values of **NecLev** in a table called **table of needs** (see table 1). In this table, each value **L=NecLev(b_ii,[m])** corresponds to the number of levels in the message **m**, which, each agent having the initial behavior **b**i, needs in order to value **(b_i m)**.

m is the message filtered by the pattern **[m]**.

If the message m is not filtered neither by bi nor by any replacement behavior obtained from b_i then NecLev(b_i[m])=0.

IV. MOBILE IMPLEMENTATION OF OUR COMMUNICATION APPROCH

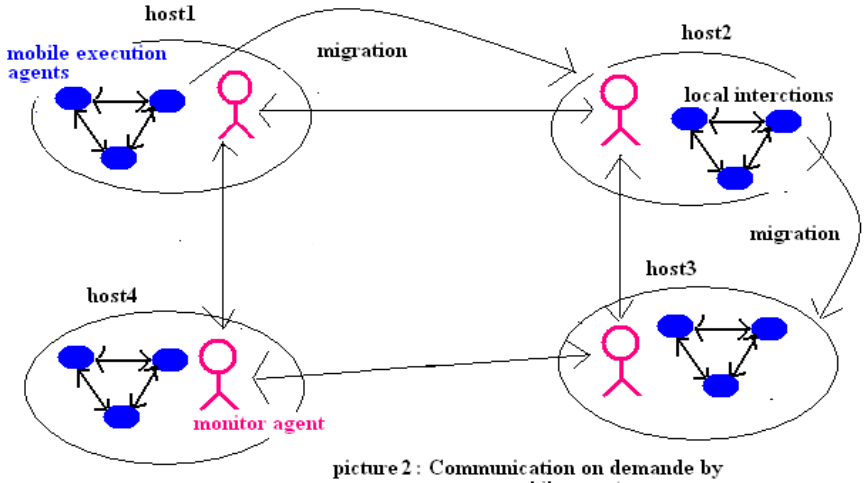
The agent model provides a unit of encapsulation for a thread of control along with internal state. An agent is either unblocked or blocked. It is unblocked if it is processing a message or has messages in its message box, and it is blocked otherwise. Communication between agents is purely asynchronous: non-blocking and non-First-In-First-Out (non-FIFO). However, communication is guaranteed: all messages re eventually and fairly delivered. In response to an incoming message, an agent can use its thread of control to 1) modify its encapsulated internal state, 2) send messages to other agents, 3) create agents, or 4) migrate to another hos (picture 2).

The create primitive (**create(X_0)**) allocates a unique mail address to the newly created agent, and creates a thread which represents the computation potency of this created agent. The system makes an adequacy between the agent and its mail queue. So, the name of an agent and its mail queue address become synonymous.

Each new agent is created in the host having the minimal number of process. It's important to allow the programmer to ignore the details concerning the physical location of the agents in different processors which constitute the network of the program execution. To that effect, every site has a **monitor agent**. This agent manages the distribution of the agents and the communication between sites.

The mobile implementation of communication strategy witch we present in this paper is based on mobile agents. In the distributed agent system, each host has its mobiles agents witch can migrate to the other hosts with the parts of messages, that must be transmitted outside.

Mobil agent is a soft entity that can migrate during the execution from host to other host. It migrates with its code, its execution state and its data. Mobil agent can receive messages in his transit, so the mobility don't handicap the it's communication process. The mobility is controlled by the application not by the execution system. The objectif of the migration is to reach distant data and resources, to do the treatment locally and ***to move just necessary data.***



picture 2 : Communication on demande by mobile agents

So, mobile agent migrates to the hosts which have a lot of interactions with his host. In the case of the low interactions, this approach become less efficient, it's penalized by the size of the mobile agent. In this second case, the messages towards distant sites are addressed to the server agent of the sender site in order to treat and send them to their destination. In the receiver site, the server agent treats the external messages that arrive and transmit them to their receivers.

Using mobile agent involves the optimization of the traffic because exchanged messages become locals and discharge the network. Mobil agents can use the resources of distant hosts, so, it discharges its initial host. The size of the mobile agent adds a surcharge when it's migrating but this charge is compensated by local interactions (picture 3).

In each host, the static analysis is done by some local calculator agents. The results of the static analysis are deposits in the table of need as explained above.

Our approach also allows remote code invocation when distant site dispose of needed service.

V. CONCLUSION

We have presented a mobile implementation of lazy strategy of term communication in a distributed environment of agents. We have used a mobile agent which migrates to distant sites having massive interactions with its site. This approach minimizes the traffic in the distributed system of agents. We are testing it on some consistent benchmarks cost concerning the execution time. This work combines into a global system for the valuation of the agent languages through a distributed virtual machine AGDVM using the mobile agents towards a new distributed agent development framework.

REFERENCES

[1] {Najat. Rafalia & Jaafar bouchabaka, An extension to □ □-calculus for Distributed Agent Programming. April 2013. International Journal of Management & Information Technology, Volume 3 N° 3, 2013, ISSN 2278-5612.

[2] Kang Lianghan, Cao Donggang, 2012. An extension to Computing Element in Erlang for Agent Based Concurrent Programming 2012 IEEE, 15th International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing Workshops..

[3] Greg Michaelson. Published at August 2011 by Dover Publications, (first published 1989). An Introduction to Functional Programming Through Lambda Calculus.

[4] The USENIX Association, All Rights.5th USENIX Conference on Object-Oriented Technologies and Systems (COOTS '99), San Diego, California, USA, May 3–7, 1999, JMAS:

[5] **]** A Java-Based Mobile Agent System for Distributed Parallel Computation. Lesgand L. Burge III, Howard University, K. M. George, Oklahoma State University, © 1999

[6] Gul Agha,Chris Houck and Rajenda Panwar. August 1991. University of Illinois at Urbana, II 61801, USA. Distributed Execution of Agent Programs Sciences Forth workshop on language and compiler for parallel computing.

[7] Henry E.Bal, Adrew S. Tanenbaum, September 1989. Department of Mathematics and Computer Sciences, Vrije University, Amsterdam, Jennifer G. Steiner, centrum voor wiskinde en Informatica, Amsterdam, the Netherlands. Programming Language for Distributed System. ACM Computing Surveys, Vol 21.

[8] Gul Agha (1986) Retrieved 2012-12-02.. Agents: a Model of Concurrent Computation in Distributed- Systems. Doctora Dissertation, Mit Press.

[9] F. Ballo, J. Saunier, F. Badeig. Evaluation of environment contextual services in multil agent systems, Communication in Computer end Information Science 271, springer, 2013.

[10] Saunier, F. Ballo, S. Pinson ; A Formal Model of Communication and Context Awerness in Multiagent System. Journal of logic Language and Information. Springer 2014.

