ISSN:0975-9646

Cyril Dumont et al. / (IJCSIT) International Journal of Computer Science and Information Technologies, Vol. 1 (5) , 2010, 392-401

# Jacobi computation using mobile agent

Cyril Dumont[1], Fabrice Mourlin[2]

*LACL, Department of Computer Science, Paris East University, 94000 Creteil, France*

*Abstract— Physical phenomena occur in a wide range of mathematical applications: from fluid to solid mechanics, electromagnetic and electrical engineering. Engineers working toward an optimized modelling must develop their software and physical system together. This development follows a standard life-cycle with design, coding and test. Validation is become quite complex and is subdivided into several parts: unit, integration, functional and system. Each piece of validation brings its brick. Entire system is tested as per the requirements. They are about numerical results, performance, architecture and fault tolerance. We defined a distributed architecture for numerical computation based on the use of mobile agents. A group or space of agents manages a whole computation, from its registration until its validation. Their role is multiple: they prepare input data and operate tasks but above they administer distributed architecture. The main impact is failure management. Failure can concern not only software services but also material. Also, the completion of an execution can need task recovery, anomaly reporting or favourite configuration. All these details consist of a main report about validation of distributed application though simulation.*

*Keywords*— Mobile agent, distributed application, space computing, architecture modelling, fault tolerance, performance measure, reporting.

## I. INTRODUCTION

Numerical software was introduced very early in the sixties; First implementations are done with FORTRAN language and dedicated platform [1]. The main goal was to transfer numerical analysis and algorithmic expertise to practitioners [2] [3]. But, the users want numerical applications which run fast, are easily moved among computing platforms, always produce the right answer, and are easy to understand and integrate with their new applications. Also, developers had to develop several version of their numerical software depending on material architecture of end users or depending of software architecture of users' projects [4].

In that context, software lifecycle [5] become complex especially in numerical analysis domain where a set of features has to be taken into account: efficiency, portability, reliability and usability are of primary concern. Behind these words, architecture constraints are hidden and thus, software building is a more complex challenge. In comparison with other domain such as web development [6], developers want to work in respect of layer approach. Each layer is about a specific concern and everything is designed platform independent. The objective is to reduce the number of application version.

When the main key word is efficiency, the challenge becomes to find out best use of computing resources. And because, time is always money, the second key word is reliability which could be declined as adaptability in a context where resources can be unavailable. A developer does not accept to lost computational time due to hardware failure. So when another resource can be used, the computation has to exploit it instead of aborting [7]. This involves third important key word: portability, which means that a resource ought to be used even if it is not the same as the previous one [8]. These three main directives drove our definition of a numerical development plat form based on mobile agents [9]. We introduce definition of mobile agent into next section and how they answer to a part of the challenge. Then, we present how mobile agents allow developer to observe software properties such that visited nodes, pending requests, etc. Next; we explain the constraints due to numerical analysis and the definition of libraries. The next part is about our implementation of our platform and definition of computing space. We use it to solve a referent example about Jacobi equation solving and finally, we provide our results on our platform and next directions for our work.

## II. MOBILE AGENT PROGRAMMING

Mobile agents offer new solutions to a spectrum of problems frequently encountered in distributed applications. At the same time, their properties provide new pragmatic concerns and allow defining new approach of distributed systems [10].

### *A. Mobile agent properties*

The life of software has increased and it became common to use software whose lifetime is greater than that of material. Also, software properties are essential for a greater lifetime. Good software has to be modular, loosely coupling, portable, and so on [11]. But its software architecture takes more importance place in its evaluation. This architecture is often static in the sense that it is decided at the installation step. So, what happens when material has to be changed? This was always a limit of software life. New installation process was to be engaged.

Mobility plays a crucial role in the adaptability of software. This property allows software or piece of code to migrate from a material to another one. Even if this change needs specific permission, it is a solution to change of materials. In

fact, movement of code has to be considered as an event like start, stop or suspend. A migration can be decided by program, for instance, a collector agent wishes to move to another work station because its activity is ended on the current node. A migration can be decided by observation, for instance, a monitor agent can export observer agent on new work stations because these devices are new on the network. In that case, mobile is managed by container instead of by agent itself.

This adaptability hides portable ability of the agent [12]. Its behaviour starts on a workstation and continues on another one. Moreover, a mobile agent is loosely coupling by construction. It cannot depend on a specific physical resource because this one has a precise location on network and the agent is mobile. Also, a mobile agent can need computing resource but this one is changeable. As first results, we defined mobile agent as an elementary brick of distributed system. We compose mobile agent into more complex structure called agency or space where they operate [13]. It is a restricted domain where specific security rules are applied to mobile agents. It is also, an execution area whose the limits are not constant.

### *B. Communication cost reduction*

A key advantage of mobile agents is that they can decrease communication costs. In client server architecture, messages are sent from client to server and they have to understand same message format. This is also a limit of software life [14]. Now, consider that the message is not only a set of data but also a set of methods to exploit these data. The format can evolve if its API respects same interfaces [15].

In another approach, part of the client or the server can move to the other side of the communications link. This means that the client or server has moved, interactions between the two can bypass the network. Very often, client part can be considered as a mobile requester. It contains only a request and its mission is to deliver its demand to a right server. If the first one is busy or unavailable, it will move to another server and so on. This is why we explain that mobile agents exploit in a better way a distributed architecture where resources are not reliable or where configuration is evolving [16]. But migration cannot be a free operation. If mobile agent first decides to move on another site, this site or agent host can accept or refuse its arrival: this is negotiation protocol with specific rules per node.

### *C. Negotiation before agent importation*

Mobility is done under control of the receiver on destination node. When a mobile agent moves to a new site, it is first studied by the agent host. We can divide mobile agents into two main categories [17]. First, there is “mobile client”. As we explained just before, it contains message and ability to read and manage the data. This means that such mobile agent exposes a provided interface (called “Messenger” for instance) which will be used by the host. The analysis of the signature of all the operations of this interface insures that this agent will not access to local resources of the host. To sum up, this mobile agent is just an observer and it will be accepted by the host if there is no more filter on message type.

Secondly, there is “mobile service”. In that case, an activity moves near to client data. If this is possible, this activity will be done on these data. Before, agent host has to check what is requested locally by the activity. All these constraints are declared into a requested interface (called “Invoker” for instance) which will be used by the host (Fig1). The analysis of the signature of all the operation lists the requirements of the mobile agent. The consequence could be an acceptance or a refusal from agent host to mobile agent. In case of importation, new permission will be assigned to this mobile agent. These will allow it to use local resources as mentioned into its requested interface.

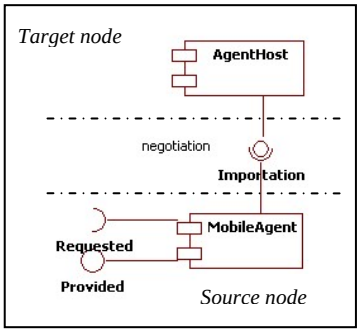


Fig. 1 : UML component diagram on agent importation

Both cases (mobile client or mobile service) are two extreme examples. Often, a mobile agent has a provided interface and a requested interface. For instance, through, requested interface mobile agent mentions that it ought to read local data set. And through its provided interface, it will expose its results (for instance, maximum and minimum values of the set). This negotiation algorithm can be more complex when requested interface is about local data exchange with other mobile agents present on the same agent host [18] [19]. Because the controls are often sequential and no deterministic, a first mobile agent has to be accepted even if the second agent is not already there. As example, if we consider an agent host which has four values. These real values are coefficients of a quartic equation of the fourth degree. To solve its equation, the host needs to receive first an agent (called “Analyzer”) for parsing the equation: is it a biquadratic equation or a quasi symmetric equation. Depending on the result, it will import an agent “Solver”. This mobile agent exposes a requested interface where it requires the use of Analyzer agent.

The negotiation protocol depends not only on the requirements of input agents. Some no functional properties can be added. First features are about the origin of mobile agent. Because a computation can be interrupted when an intruder item occurs, agent host has to know the agent base. This means the base where the agents are, is published into a registry. We want to avoid that a “Solver” agent can replace another one accidentally. Also, we defined the concern of agency or agent space, which delimits an area where an agent base is available. Thus, an agent host can only receive mobile

agent from the agent space where the host belongs to. This structure of space is initially build at the beginning of the computation case and can be considered as a first reading as a container of a computation.

Because security is often considered as a key feature in web project, the negotiation protocol can take into account the provider of the mobile agent through a certificate. This means that a public key can be read to identify the issuer of the agent. When the public key is recognised, additional permissions can be assigned to the agent and so improve its execution on the agent host. To be stable, a negotiation protocol must be opened to new rules such that localisation (it means the previous sites where the mobile agent was), time to run before the end of the lease, version management.

Over the set of nodes belonging to a space, the same negotiation protocol is often applied, but we can configure a subpart with one strategy and another without any control. This can be interesting when some nodes are not available, we can apply a negotiation protocol which always fails.

### *D. Mobile agent manageability*

The requirements for managing distributed applications depend on types of applications. Short-lived computations or long-lived computations do not require same controls [20]. In the first case, run time is less than few minutes. Administrator does not have enough time to observe really what happens. But he will record data about run time for a post mortem analysis. When execution time is more than few minutes, we guess that administrator can apply observations on to an agent space. The administration of mobile agents provides methods that can be used to manage agents belonging to a space.

The definition of these operations comes from the properties of mobile agent system. First, mobility involves that an observer needs to know where mobile agents are over the network. This information is obtained from agent hosts. When a mobile agent moves from one node to another one, agent host receives mobile agent and records its importation request with the success or failure of the negotiation. Also, the collect of these data provides enough details to build a geographical map about the activities of mobile agents.

But this map is only a snapshot of an execution. For a long-lived computation, an observer needs to follow mobile agent to understand its itinerary over network. This could provide details about exploitation of computing resources. This can explain also delay during the computation. When, two mobile agents (Analyzer and Solver) have to communicate directly on to a specific agent host. If the itinerary of Solver agent is shorter than the itinerary of the other, it could be blocked, waiting for an exchange of data.

Communication can also be enhanced to become more expressive for management. A basic improvement is to use message queue to keep trace on requests. In that case, mobile agents exchange message asynchronously. This approach offers three advantages. First, administrator can access to message queue to control its length and its contents. The message order can be changed; some of them can be deleted. Secondly, message can be stored by observer for a post mortem analysis. Finally, delays are suppressed when a sender is waiting for a receiver. Then, administrator can interact with distributed execution through message queues. If a mobile agent is considered not enough powerful for the amount of messages, administrator can simulate agent host and ask for a new importation of agent. This kind of strategy is welcomed when activity is unpredictable or when delays are cumulated until a final incident.

Direct management operations on mobile agent need to design mobile agent to be administrable. This aspect is often treated as a technical facet distinct of business activity of agent. So, their internal properties are observed by the use of dedicated framework such as JMX or by the use of web service with SOMA [21], [22].

## III. EXPERIENCE AND APPLICATION DOMAIN

We started our research activities on mobile agent for more than a decade. First implementations show important properties, explained in previous sections. First applications were about distributed system management: new implementation of service location protocol [23]. We built another implementation of AAFID algorithm defined by Spafford and Zamboni [24], where mobile agents help to detect intrusion on a network.

We defined a distributed monitor based on an agency which collects anomalies of cluster of web servers [25]. This project was realized for end users called AnswerDesk Corp. which manages information for call centers. This work has supported extensions for improve communication with other applications. This means that XML messages were accepted as input and output. Monitoring reports were built from mobile agent observation and transform into XML stream. As results, reports were used by other tools like mail and editor for publication [26].

Another application domain is numerical analysis and the definition of a platform for the management of computing cases. In that context, mobile agents are used to manage heterogeneous codes into a space where data are prepared for a distributed computation. We applied our platform for several case studies. Each of them brings new improvements for our platform, about performance, administration and also for interoperable exchange of mobile agent [27] [28]. We consider that all the examples are convincing and they show the essential role of mobile agent into distributed computations. We used several numerical codes such that, Choleski, Pi calculus or Cardan solver because we wish to show our approach is polyvalent. In this document, we decided to introduce our recent improvements with the use of Jacobi computing code. This is a reference computation and some numerical solvers already exist. So it is possible for readers to compare our results. Moreover, Jacobi computation interferes into more complex computations in thermodynamic simulation. Thus, this application can be thought as a part of a more complex case study.

Application of mobile agent is not restricted to use of local network. We can consider mobile agent as a mobile service and export it on remote node through http protocol. First

validation is already done for data collection of RFID sensors. The subject of our prototype was the authentication of users for remote working [29].

## IV. REQUIREMENTS FOR NUMERICAL COMPUTATION

We begin by outlining critical requirements for numerical computing. Many new languages offer features that can provide significant benefits for developers of mathematical software [30]. Languages that reduce programmer time and increase software reliability, involve often cost in computer resources. Generally, the requirements are ordered and performance is main concern, followed by memory management.

### *A. Performance for numerical part*

A platform form for numerical code has to be efficient. It often contains a lot of computations on large set of data and its evaluation has to be fast enough for building a benchmark. Performance concerns not only the cost of functions of methods, but also resource accesses and code placement on computing resources.

When placement is declared at the beginning of computation, this does not accept any perturbation. If a processor is not usable, the computation can be aborted or performance can be heavily damaged. When execution context is no reliable or when computing resources have to be shared between users, then platform ought to be fault tolerant. This means that the platform has to manage material architecture. If all resources are similar like a grid of processors, the management is simplified, but it can become more complex when network is heterogenous (clock cycle, cache size, etc).

### *B. Memory management*

Recent studies and experiments have shown that in computationally intensive applications where objects are being allocated and released more frequently, garbage collection can actually observed carefully [31]. This becomes much more difficult when memory is distributed onto a set of processors.

Explicit memory management has proved to be a fruitful source of bugs, crashes, memory leaks, and poor performance [32]. Also it may be preferable to leave memory management to the operating system and to observe its effects. This problem is even more important than the data size is huge. For instance, when multi dimensional matrix is used with FDTD computation and the number of items can be bigger than a million of float value.

The platform has to manage occurrences of large data for reducing duplication and also for optimizing their accesses in a multi thread application. It can also offer pre-statement on these large data. For instance, after data extraction from a data source, data have to be prepared for computation. Also, this is in close relation with numerical code.

When numerical code is written into an object oriented language, implementation of alternative arithmetic parts, such as complex, interval, and multiple precision requires the support of new objects with value semantics. The size of classes is essential and developers have to preferred lightweight classes [33].

### *C. Reuse of code*

Code rewriting can be an approach when new language is chosen with new features. But constraints of new languages do not allow same performance. Java software is often perceived to be slow as compared to corresponding C/C++ or FORTRAN software. For some computationally demanding algorithms, straightforward implementations in Java may run 100-150 times or more slower than C++ or FORTRAN. In the past, problem algorithms have included floating point intensive algorithms such as FFTs (Fast Fourier Transformation) and integer functions such as alignment byte manipulations [34].

Also, a platform for numerical code evaluation has to accept code written in different languages, such as Java, C#, C++ or FORTRAN. This programming language is always the most common and some applications are considered as reference in numerical domain. This means that no developer will decide to rewrite such application into a more convenient programming language, even if maintainability is improved.

### *D. Data security*

Traditional distributed systems enable users to use data and applications on distant networks without confining them to networks that they are directly connected to. Unfortunately, development of data security in distributed systems takes place simultaneously with the development of the network [35].

When a set of computing resources are shared by two users, each of them what to have the insurance that his data and his code is distinct from his colleague. Even if this colleague belongs to the same team, a platform has to isolate not only strategic data (because they can represent a important mathematical model for instance), but also execution of code. If the components of both executions are mixed, it will certainly cause errors during simulation and waste of time.

If separation of code is easy to do, it is not so easy to prove when end users want to do it. Very often, geographical isolation is used to establish that there is no interference at all. Data preservation has to respect the same constraints except that the size is not the same. In some code this size corresponds to a material limit. Also, its distribution on set of computing resources set more serious problems of management.

This remark requires defining a container of computation run time. This container will be responsible to component loading for code part and also, manager for data scattering. Of course, other facets can be added to that container, such that distributed transaction. It is a distributed control mechanism analogous to database transactions for controlling access to shared memory in distributed computing. This aspect has also to be isolated into a run time structure.

Thus, it appears that use of container (agent space) is crucial into a shared environment. Finally the containers have to be observed through platform which plays the role of controller. This corresponds to the management of state of

containers (its automaton is basic at that level and will become more structured into next step of implementation).

## V. IMPLEMENTATION OF A MOBILE AGENT COALITION

The implementation of our architecture is completely written in Java. The Java language and his APIs provide portability (very useful in a heterogeneous network of machines). One of these APIs is specially adapted to our architecture: the Jini API. Jini is the name for a distributed infrastructure computing environment that can offer "network plug and play". A device or software service like an agent can be connected to a network and announce its presence, and clients that wish to use such a service can then locate it and call it to perform tasks [36].

### *A. The JavaSpaces Technology*

JavaSpaces technology [37] is a high-level communication tool for processes into a distributed application. It is a space-based model with a main element: a *space*. A space is a shared, network-accesible repository for objects. A space stores *entries.*

You can invoke four primary operations on a JavaSpaces service:

- *write :* Writes new entry into a space
- *read :* Makes a copy of an entry in a space
- *take :* Retrieves an entry from a space
- *notify :* Notifies a specified agent when entries that match the given template are written into a space

Each operation has entries as a parameter. Some are templates, which are a kind of entry. The write() operation is a store operation. The read() and take() operations are a combination of search and fetch operations. If a take() or read() operation doesn't find an object, the process can (or not) wait until an object arrives.

Unlike conventional object stores, objects are passive data. Therefore, processes do not modify objects in the space or invoke their methods directly. In order to modify an object, a process must explicitly remove, update, and reinsert it into the space.

### *B. The MCA Architecture*

Our architecture is based on JavaSpaces technology. We will present the main elements for the implementation of the resolution of a computation case (Fig. 3).

#### *1) The MCASpace*

The *MCASpace* is a specialized space for numerical computation. It is actually a subclass of a space. It contains a limited number of types of entries. These entries all implement the abstract class *Storable (Fig. 2)*, which serialize all entries put on the *MCASpace* (in XML format). We describe then these types of entries:

- *ComputationCase :* When an object of that type is added to *MCASpace*, a transaction is created. All entries specific to this computation case will be part of this transaction.
- *MCAProperty:* represents a global property of a computation case with a name and a value. These properties are shared by all agents participating in a computation case
- *DataHandler:* Such entries can simulate a distributed memory. Indeed, each entry of this type gives access to a resource : write access with the method *getInputStream* and a read access with the method *getOutputStream*. *I*
- *Task:* It is a representation of a task which is a part of the computation case. A *Task* has the following properties :
  - ***name***: it must be unique. In a transaction, there cannot be two Tasks with the same name.
  - ***parameters***: it is the parameters list of the Task. It can be null.
  - ***compute_agent_name***: it is the name of the *ComputeAgent* needed to execute the Task.
  - ***worker:*** it is the name or address of the *ComputingWorker* which executes or had executed the this task.
  - ***result:*** it is the result of the Task. It can be null if the Task uses the *DataHandler* without giving any result.
  - ***parentTask:*** it is the name of the Tasks which are needed to be executed before the current Task.

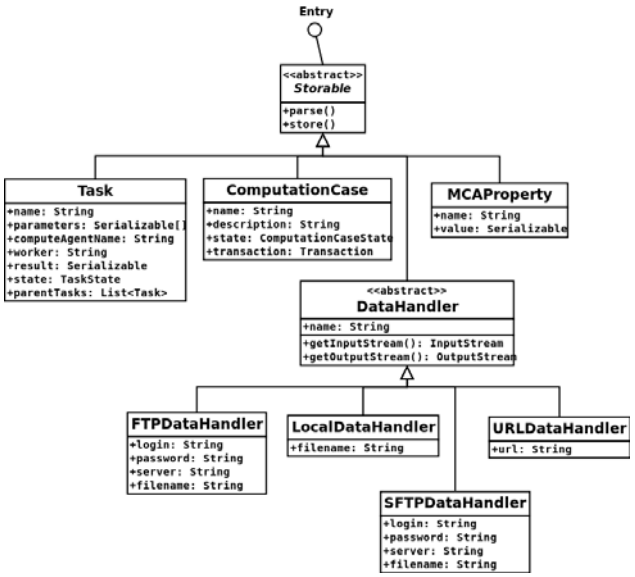


Fig. 2 Entries class diagram

*Barrier Synchronization* – Barriers are a common and simple technique to synchronize agents in a distributed computation. Barriers are easy to implement in our architecture with a shared entry (named *Barrier*) in the *MCASpace.*

2) *ComputingWorker* agents

A *ComputingWorker* is a agent that executes a part of of the computation case algorithm (using the *ComputeAgent* we will see in the next section *C.*). The number of *ComputingWorker* may vary during the execution of the case.

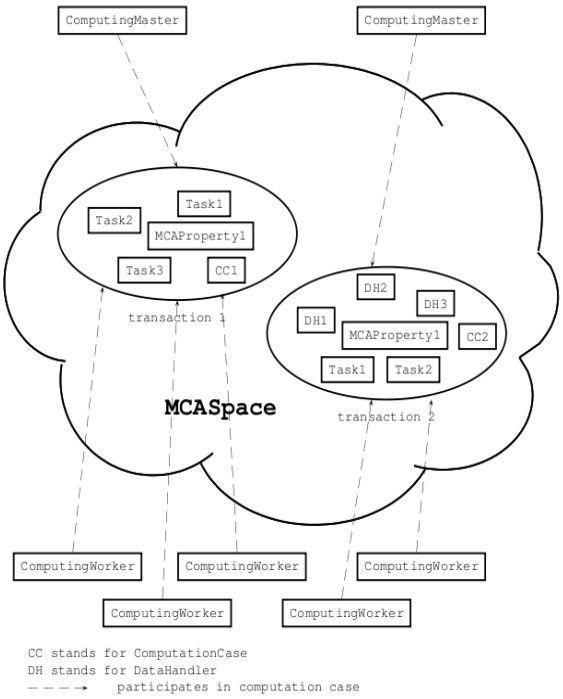


Fig. 3 Main MCA Components

3) *ComputingMaster* agent

In all cases, the *ComputingMaster* writes the first task to execute on the *MCASpace* and then signals the end of its computation case to the coalition of agents involved in the computation case resolution.

The *Master* agent plays several roles:

- Scheduling: creation and writing of Task objects necessary for the resolution of the case.
- Data partitioning: creation and writing of *DataHandler* objects necessary for the resolution of the case.

*C. Definition of a mobile agent*

*ComputeAgent* is a mobile agent available on a lookup for all the *ComputingWorkers*. When a *ComputingWorker* needs one *ComputeAgent*, it gets a copy of it. This agent must implement an interface *ComputeAgentInterface*, and redefine the method *execute*. This method takes two parameters: one Task and a set of *MCAProperty*; it returns a result. It is here that we find the computation algorithm. We have to define the different *ComputeAgent* of the computation case to define the different Tasks.

There are two types of *ComputeAgent:*

- A Java agent: the code of the computation case is developed in Java code. The *ComputeAgent*, necessary for the executions of the task, is a class that implements the Java interface *ComputeAgentInterface* and redefines the method execute. Then the code is completely written in Java language and can use multiple Java API available.
- A native agent: this time, the code of the computation case is already existing and developed in another language (C/C++). It is here that adaptive runtime for numerical code makes sense. (Fig. 4)

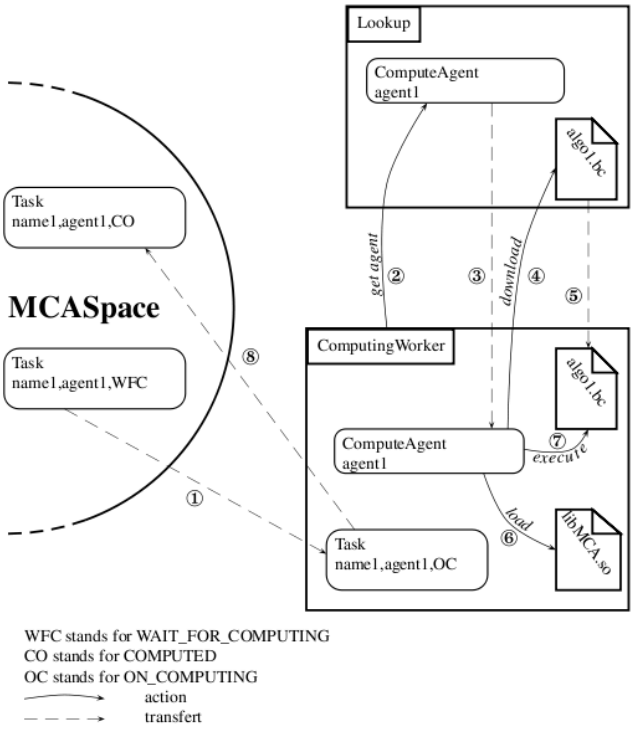


Fig. 4 Execution of a native task

## VI. APPLICATION EXAMPLE: THE JACOBI RELAXATION

The Jacobi Relaxation is an iterative algorithm that is used to approximate Laplace differential equations. The Jacobi Relaxation technique can be used in a variety of applications, including the simulation of temperature transfer (as we'll see shortly).

### A. Background

The problem that interests us is the temperature distribution in a square box whose boundaries are subjected to a constant temperature. From Fourier's law [37], which describes the transport of heat in a homogeneous medium, we can take the heat will spread within the field based on a dynamics described by the following partial differential equation:

$$\frac{du}{dt} = \alpha\left(\frac{d^2u}{dx^2} + \frac{d^2u}{dy^2}\right)$$

with $u$ the function representing the temperature and α the coefficient of thermal diffusion in the area studied. In a closed system (which is our case) the temperature will tend towards a steady state. The problem is therefore to solve the Laplace equation

$$\frac{d^2u}{dx^2} + \frac{d^2u}{dy^2} = 0$$

We consider that the problem lies in the x0y plane. To solve it, we will discretize square box into a mesh $\{(x_i, y_j)\}$ with step $h$ and $h = x_{i+1} - x_i = y_{j+1} - y_j$. Obviously, the higher the accuracy threshold, the lower the numerical solution will be close to the theoretical value.

Using the method of finite differences, we obtain

$$u(x,y) = \frac{1}{4}\left(u(x+h,y) + u(x-h,y) + u(x,y+h) + u(x,y-h)\right)$$

and we note that the function $u$ at point $(x,y)$ is the average of its 4 adjacent points.

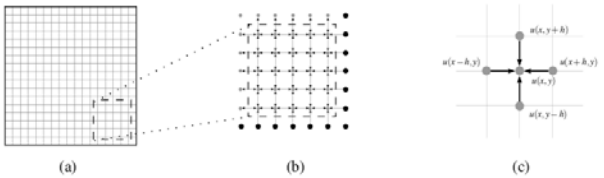


Fig. 5 Modeling of a square box: (a) a 2-D mesh; (b) communication between each vertex at each iteration; (c) representation of a vertex $u(x,y)$ in the $x0y$ plane.

#### 1) sequential algorithm

To solve this problem we represent the function $u$ by a matrix $U$ of real numbers and proposed the following iterative scheme

$$U_{i,j}^{k+1} = \frac{U_{i+1,j}^{k} + U_{i-1,j}^{k} + U_{i,j+1}^{k} + U_{i,j-1}^{k}}{4}$$

By imposing a temperature at the boundaries of the domain, the problem is solved by letting evolve the temperature until the dynamics stabilizes. We use the Dirichlet boundary condition - When imposed on an ordinary or a partial differential equation, it specifies the values a solution needs to take on the boundary of the domain.

We deduce the following algorithm

*1. Initialization*. We impose the value $T_{out}$ to all the points on the *West* boundary ($j = 1$) and the value $T_{out} = 0$ to all the points on the East ($j = n$), North ($i = 1$) and South ($i = n$) boundaries.

*2. Iteration*. For each value $U_{i,j}$, we calculate the average of neighboring values. Then we compute the absolute value of the difference $\delta$ of value between the old and the new value. Finally we calculate $\varepsilon = max(\varepsilon, \delta)$ to know the biggest difference in iteration.

*3. Test*. We define threshold accuracy $\sigma$. If $\varepsilon < \sigma$, the process is stopped. Otherwise it restarts the iteration. The process converges to the solution. If the accuracy threshold was not reached, the process will be *MAXITER* maximum iterations.

#### 2) parallel algorithm

There are many ways to parallelize this algorithm. The simplest way is to create as many threads as there are processes on the network, where each process performs this algorithm for a subsection of the matrix. This is known as a sub-matrix decomposition. In this way, each process iterates through one a sub-matrix of the overall matrix and updates the values of the cells in its own sub-matrix.

This algorithm exhibits data parallelism due to the fact that the same set of steps are applied to multiple pieces of data. In this case, the procedure is an average being computed and the different pieces of data are sub-matrix.

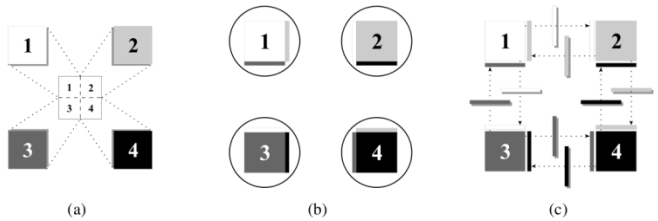


Fig. 6 (a) sharing matrix in sub-matrix; (b) each process receives a sub-matrix and the boundaries of neighbouring sub-matrices; (c) at each iteration the boundaries are update and should be exchanged with the processes that deal with neighbouring sub-matrices

With this partition, each sub-matrix may communicate with two, three, or four neighbors, depending of their position (respectively at a corner, a border, or in the center of the whole matrix). This partition is more effective when the data to processor ratio is large.

Communications appear at sub-matrix boundaries to send boundaries values to neighbors and receive values from neighbors.

### *B. Implementation of Jacobi Relaxation on our architecture*

If a resolution of a specific computation case does not require redefinition of a "specific" *ComputingWorker*. The resolution of the Jacobi relaxation using our architecture requires the development of two new "specific" classes:

- *JacobiMaster:* An agent, instance of this class, writes a *ComputationCase* entry into *MCASpace* and gets the new transaction. Then he writes two *MCAProperty*: one for the desired accuracy and one for the maximum number of iterations.

  This *ComputingMaster* divides the file containing the matrix into files containing sub-matrix and he writes the corresponding *DataHandler* entries on *MCASpace*.

- *JacobiComputeAgent:* This class derives from *ComputeAgentInterface.* An agent mobile, instance of this class, is on a Jini lookup, somewhere on the network, not necessarily on the same machine as the MCASpace. This agent reads the two *MCAProperty*. It reads data corresponding to the task using the *DataHandler*.

  Then it executes a local compute, it updates the value of each cell. For the boundaries, JacobiComputeAgent need to communicate with their neighbours. Each agent writes two, three or four *DataHandler* corresponding to these boundaries.

  Before reading the data necessary to calculate its boundaries, a JacobiComputeAgent use a barrier (cf. The MCA Architecture) to wait until all other agents have finished computing the current iteration. When the barrier is passed the agent can execute a "border" compute using the *DataHandler* corresponding to the boundaries of its sub-matrix neighbouring.

## VII. MAIN RESULTS

In this section, we report on experiments designed to test the effectiveness of our approach. We start with results about Jacobi computation published into 44th IEEE Conference on Decision and control [39]. These results are also available on network. It appears that our numerical results are equivalent and our time measure provides interesting time distributions. They highlight ration between computing time and management time of platform.

We observe that data preparation is unimportant comparing to agent space creation and management, but it cost increases with size of data, but the ratio is quite weak.

Our tests are also about reliability. We do not have any reference with equivalent work in numerical domain. The scheduling of our tests starts by interruption of computing resource. This is done easily by use of system interruption. The observed effect is the cancellation of task registration. Then its initial state is reinitialized. Next, a worker books this task and redoes it as if it was a new one. By the end, the results are obtained even if delay can be recorded.

Next test is about the share of computing resource. Because we want to insure that components are not shared and also that data are well managed, we used same computing resource by to agent spaces. Then we observed what is loaded by both spaces and we recorded code base of each loaded component. It appears that code origin is preserved and also our control strategy can be enforced by use of signature. We did not used that in our experiments because its control involves a time overhead.

We observed also memory mapping during experiments. Our goal is not to find memory leaks but to check hierarchy of access to main data. This step is equipped by use of managed beans which notify read and write events to a bean console. After several test suites, we do not identify any violation of access from one agent space to another one. Of course, this is not verification but a validation for convincing that our approach has good properties.

Next, we recorded performance test by use of distinct configurations of memory allocations. A configuration is based on set of parameters which customize not only memory size but also its subdivision into generations. This configuration contains also a selection of garbage collector algorithm dedicated for distributed system. To sum up, garbage collector interruption cause delay in our simulation and the number of passes can be reduced if young generation size is increased. But it is also essential to limit the period of each garbage pass.

As a drawback, when new computing resources are added to our platform, their management involve memory allocation and precise memory configuration is less important. So, we concluded that this customization cost much more time to set than an adapted management of computing resource.

By the end of our tests, we considered that a deployment on architecture of eight computing resources was the low limit to apply our Jacobi solver, but this value is closely related to the

input data. We currently continue test step to determine low limits with other reference matrices.

## VIII. CONCLUSION

We explain through our document the important role of a platform dedicated to simulation of numerical code. We started from our past experience with mobile agents and our knowledge into numerical computing and we build a case study around a reference example.

This example stresses the advantage of our platform for validation of software architecture. After a set of test, the objective is to design the most well adapted architecture for a distributed computation. Because, these features depend on empirical concern (data and code), simulations is the most efficient way. In our example, this was obtained after a classical validation step through reference tests.

## REFERENCES


[1] J. J. Dongarra, C. B. Moler, J. R. Bunch, and G. W. Stewart. LINPACK Users' Guide. SIAM, Philadelphia, 1979.

[2] E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. D. Croz, A. Greenbaum, S. Hammarling, A. McKenney, S. Ostrouchov, and D. Sorensen. LAPACK Users' Guide. SIAM, Philadelphia, 1992.

[3] A. J. C. Bik and D. B. Gannon. A note on native level 1 BLAS in Java. Concurrency: Practice and Experience, 9(11):1091-1099, Nov. 1997.

[4] R. F. Boisvert, S. Browne, J. Dongarra, and E. Grosse. Digital software and data repositories for support of scientific computing. In N. Adam, B. K. Bhargava, and M. Halem, editors, Advances in Digital Libraries, number 1082 in Lecture Notes in Computer Science, pages 61-72. Springer-Verlag, New York, 1996.

[5] Guidelines for the Successful Acquisition and Management of Software Intensive Systems (GSAM), Version 3, Chapter 5, USAF Software Technology Support Center, May 2000.

[6] Pressman, Roger S., "Understanding Software Engineering Practices, Required at SEI Level 2 Process Maturity," Software Engineering Training Series, Software Engineering Process Group, 30 July 1993.

[7] Java language proposal. Visual Numerics, Inc., 9990 Richmond Ave., Ste. 400, Houston, TX 77042-4548. Available at http://www.vni.com/products/wpd/jnl/JNL/docs/intro.html, 1997.

[8] G. Fox, X. Li, Z. Qiang, and W. Zhigang. A prototype Fortran-to-Java converter. Concurrency: Practice and Experience, 9(11):1047{1061, Nov. 1997.

[9] Cyril Dumont, Fabrice Mourlin: A Mobile Computing Architecture for Numerical Simulation UBICOMM 2007, November 4-9, 2007 - Papeete, French Polynesia (Tahiti) IOS Press (6 pages),

[10] G.R. Andrews, 1991, Paradigms for process interaction in distributed programs. ACM Computing Surveys, 23(1):49–90.

[11] E. Gendelman, L. F. Bic, and M. B. Dillencourt, 2000, An application-transparent, platform-independent approach to rollback-recovery for mobile-agent systems. In ICDCS 2000: 20th International Conference on Distributed Computing Systems, Tapei, Taiwan.

[12] Frederick C. Knabe. Language Support for Mobile Agents. PhD thesis, School of Computer Science,CarnegieMellon University, Pittsburgh, Pennsylvania 15213,December1995. Technical report CMU-CS-95-223.

[13] Andreea Barbu, Fabrice Mourlin: A Higher Order-Calculus Specification for a Mobile Agent in JINI. SNPD 2003, Fourth International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD'03), October 16-18, 2003, Lübeck, Germany pp 203-209.

[14] L.F. Bic, M. Fukuda, and M. Dillencourt, 1996, Distributed computing using autonomous objects. IEEE Computer, 29(8).

[15] Anand Tripathi, Tanvir Ahmed, Vineet Kakani, Shremattie Jaman. "Distributed Collaborations using Network Mobile Agents", 2nd International Symposium on Agent Systems and Applcations/4th International Symposium on Mobile Agents, September 2000.

[16] Anand R. Tripathi, Muralidhar Koka, Sandeep Karanth, Abhijit Pathak, and Tanvir Ahmed. "Secure Multi-Agent Coordination in a Network Monitoring System", Software Engineering for Large-Scale Multi-Agent Systems, 2002 (SELMAS 2002), 251- 266, Springer, LNCS 2603, 2003.

[17] G. M. Gavalas D., Greenwood D. and O. M. Advanced network monitoring applications based on mobile/intelligent agent technology. Computer Communications Journal, 23(8):720–730, April 2000. http://citeseer.nj.nec.com/268291.html.

[18] Garrido L, Sycara K. Multi-agent meeting scheduling: preliminary experimental results. Proceedings of the Second International Conference in Multi-Agent Systems (ICMAS'96), Kyoto, Japan; 1996.

[19] Bouzid M, Mouaddib A-I. Cooperative uncertain temporal reasoning for distributed transportation scheduling. Proceedings of the International Conference on Multi Agent Systems; 1998. pp. 397–398.

[20] A. Corradi, C. Stefanelli, F. TarantinoHow to Employ Mobile Agents in Systems Management, Proceedings of the Third International Conference on The Practical Applications of Intelligent Agents and Multi-Agent Technology (PAAM'98), London, UK, March 23-25, 1998. Pages 17-26.

[21] P. Bellavista, A. Corradi, C. Stefanelli: An Integrated Management Environment for Network Resources and Services, IEEE Journal on Selected Areas in Communication (JSAC), Special Issue on "Recent Advances in Network Management and Operations", Vol. 18, No. 5, pp.676-685, May 2000.

[22] F. Baschieri, P. Bellavista, A. Corradi, Mobile Agents for QoS Tailoring, Control and Adaptation over the Internet: the ubiQoS Video on Demand Service, published in the proceedings of the 2nd IEEE International Symposium on Applications and the Internet (SAINT'02), Nara, Japan, January 28-February 1, 2002, IEEE Computer Society Press.

[23] Andreea Barbu et Fabrice Mourlin, "From higher order Pi calculus specification to RMI implementation", CSITeA'03, International Conference on Computer Science, Software Engineering, Information Technology, e-Business, and Applications (June 5-7, 2003 Rio de Janeiro, Brazil) BRJ Academic Editor pp241-251,

[24] Maamoun Bernichi et Fabrice Mourlin, "Java mobile agents for monitoring mobile activities", In Eurocon'05 conference, Serbia & Montenegro, Belgrade, November 22-24, 2005.SMB Editor pp63-73

[25] Bernichi, M. & Mourlin, F. "Software management based on mobile agents", Second International Conference on Systems and Networks Communications, IEEE Computer Society Press (6 pages), Cap Esterel, France, Aout 2007

[26] Mekki, R. & Mourlin, F. "Mobile agent as interoperable mobile service for monitoring ", Second International Conference on Systems and Networks Communications, IEEE Computer Society Press, Porto, Portugal, September 2009

[27] Dumont, C. & Mourlin, F. "Space based architecture for numerical solving", ISE2008: 5th International Conference on Innovation in Software Engineering, 10-12 December 2008 - Vienna, Austria (IEEE CS 6 pages)

[28] Cyril Dumont, Fabrice Mourlin, "Adaptive runtime for numerical code", MOSIM'2010: 8th ENIM/IFAC International Conference on Modelling and Simulation, IOS Press (8 pages),

[29] Agourare, K. & Mourlin, F. "Authentication and Location Control via RFID Analysis" ETFA 2009, Emerging Technologies and Factory Automation, September 23 - 26, 2009, University of Balearic Islands (UIB), Spain NTCS Editor 86 pages,

[30] Press, W. H., Flannery, B. P., Teukolsky, S. A., and Vetterling, W. T. 1998. Numerical Recipes in C++: The Art of Scientific Computing. Cambridge University Press, Cambridge, Mass,

[31] Berger, E.D.; Zorn, B.G.; McKinley, K.S. (2002). "Reconsidering custom memory allocation". Proceedings of the 17th ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications. ACM Press New York, NY, USA. pp. 1–12.

[32] Jose E. Moreira, Sam P. Midkiff, and Manish Gupta, From flop to megaflops: Java for technical computing. Proceedings of the 11th International Workshop on Languages and Compilers for Parallel Computing, LCPC'98. IBM Research Report 21166, 1998.

[33] Bruno Blanchet, Escape Analysis: Correctness Proof, Implementation and Experimental Results, in Proceedings of the 25th ACM SIGPLAN-

SIGACT Symposium on Principles of Programming Languages, January 19-21, 1998.

[34] Schatzman J. C.; Writing high performance Java code which runs as fast as FORTRAN, C or C++, Society of Photo-Optical Instrumentation Engineers, Bellingham, WA, INTERNATIONAL (Revue) SPIE, Bellingham WA, ETATS-UNIS (2001) vol. 4521, pp. 106-114 ISBN 0-8194-4245-3,

[35] Guynes, C., Golladay R, & Huff R. (2000), Database security in a client/server environment, SIGSAC Review, 14, pp. 9-12.

[36] Jan Newmarch, Foundations of Jini 2 Programming, 2006

[37] E. Freeman, S. Hupfer, and K. Arnold. JavaSpaces Principles, Patterns, and Practice. Pearson Education, November 1999.

[38] J. Legras. Techniques de résolution des équations aux dérivées partielles. Paris, Dunod, 1956

[39] J.-P. Aubin, A. M. Bayen, and P. Saint-Pierre. A viability approach to Hamilton-Jacobi equations: application to concave highway traffic flux functions. In 44th IEEE Conference on Decision and control and European Control Conference, pages 3519–3524, Sevilla, Spain, Dec 2005.