Mobile Agent Framework for Parallel Computing: MAFPC

Cyril Dumont, Fabrice Mourlin
Paris 12 University
LACL
Créteil, France
dumont_cyril@yahoo.fr, fmourlin@yahoo.fr

***Abstract* — the parallelization of numeric applications highlights several strategies which can be applied depending on numeric context. But a feature is often missing: the adaptability. The definition of a numeric context means the configuration of resources such as memory, processor load and communication graph, but the resource availability is an evolving feature. It is not predictable and the adaptable aspect is essential.**

Because the execution has to be driven by the availability of main resources, the agents of a numeric computation have to react when their context changes. This paper proposes a framework called MAFPC, based on mobile agents; they represent a control layer. This layer plays the role of an observer which manages the computing context. The observations can also generate events for more global views such as monitor, debugger or tools to track down performance issues on demand.

***Index Terms*—mobile agent, monitoring, numeric case study, scheduler.**

I. Introduction

THIS document presents our approach of parallel adaptive programming technique. This need comes from our past experience about parallel programming and especially in numeric analysis domain. The basic features of any simulation allow users to create a initial configuration. A configuration is about all the resources used during an execution: processor memory, input / output devices, communication channels but also some actions such as reporting and saving. An initial configuration can not take into account all the change of a simulation context even potential bug or breakdown.

With a classical parallelization of code, an execution of code meant that all the external features should be fixed before the start point (the beginning of the execution). This kind of constraints is not accepted today, because frequently, the input data are not predictable or risks have to be considered during execution. Thus, it is essential that the main features of the execution context are managed during the whole execution of a code. For instance, if a given node of the network is not alive, the numerical simulation does not stop but the activity has to be adapted over the other nodes.

The adaptability of a code means that the resources are used as well as possible. If the resource is processor, it involves that some code has to be exported onto another processor which is recently free. This is the main reason why adaptability is always realized by the use of mobile agent.

This paper describes an abstract framework for itinerant agents that can be used to implement numeric remote applications. The idea of harnessing computational power of networked computers is not new. It has long been an active area of research. Job scheduling systems have covered a wide range of needs, from traditional batch job queuing, to load sharing, and cycle stealing; see [1] for an excellent review of leading packages. There are also parallel programming environments that provide task scheduling, load balancing, and even fault tolerant services in support of parallel applications on clusters [2][3].

In this paper, we introduce a new approach for the development of Java-based numeric applications over our framework of mobile agents. Our approach is based on the idea of using mobile agents for the control of resources. These agents are launched to different hosts on the network and they cooperate and communicate among themselves in order to solve large problems fast. The numeric application is considered as a client of the agent control layer.

The driving force motivating the use of mobile agents is twofold. First, mobile agents provide an efficient, flexible and asynchronous method for searching for information or services in rapidly evolving networks: mobile agents are launched into the unstructured network and roam around to gather information. Second, mobile agents support intermittent connectivity, slow networks, and lightweight devices. This second property makes the use of mobile agents very attractive. In our case, it relieves the remote client from any unnecessary overhead by moving it to the heavyweight servers at the fixed network. Furthermore, it allows a straightforward adaptation of our approach to the emerging, and very popular, wireless environments.

This work was realized at the L.A.C.L. (Laboratory of Algorithm, Complexity and Logics of Paris 12 university, France).

Cyril Dumont is with the L.A.C.L., Paris 12 university, France (e-mail: dumont.cyril@gmail.com).

Fabrice Mourlin., is with the L.A.C.L., Paris 12 university, France corresponding author to provide phone: +33-615051551; fax: : +33-145170350; e-mail: fabrice.mourlin@wanadoo.fr)

We structure our paper as follows: a first part where we explain how the control layer is deployed, a second part describes the exchange of data between the mobile agents (which observes the resources) and the numeric application and thirdly, we detailed how to develop with our framework. Finally, we conclude about the first results we obtained with recent computing application.

II. A CONTROL LAYER AS AN GLOBAL OBSERVER

A. Concepts of a Mobile Agent System

Mobile agents are defined as active objects (or clusters of objects) that have behavior, state and location. Mobile agents are autonomous because once they are invoked they will autonomously decide which locations they will visit (their roadmap) and what instructions they will perform (their task). This behavior is either defined implicitly through the agent code (see e.g. [4]) or alternatively specified by an itinerary at runtime modifiable (see e.g. [5]). Mobile agents are mobile since they are able to migrate between locations that basically provide the environment for the agents' execution and represent an abstraction from the underlying network and operating system.

With the properties printed out above it has been often argued that mobile agents provide certain advantages compared to traditional approaches as the reduction of communication costs, better support of asynchronous interactions, or enhanced flexibility in the process of software distribution.

The use of mobile agents has been particularly promising in application domains like information retrieval in widely distributed heterogeneous open environments (e.g. the World Wide Web), network management, electronic commerce, or mobile computing. This last domain contains applications which use large set of data and also intensive computations. For instance, electromagnetic simulations mean to solve Maxwell timed equations and the size of the data depend on the frequency of the experiment. If the frequency is greater than 1 GHz, the set of data can not be treated with only one processor. Moreover, if the volume of the experiment is greater then 10 m^3, then the limits of the computation needs also more processors.

Several implementations already exist but few of them hide technical features from their environment. Data exchange is also a strong constraint in agent community; data type has to be preserved from the sender to the receivers. These main constraints helped us to select a mobile agent framework. Because there is no framework which respects these features, we developed our own framework which was presented at the ESM 2005 conference [6].

B. Mobile agent framework with JINI

1) Jini introduction

By using objects that move around the network, the Jini architecture makes each service, as well as the entire network of services, adaptable to changes in the network. The Jini architecture specifies a way for clients and services to find each other on the network and to work together to get a task accomplished. Service providers supply clients with portable Java technology-based objects that give the client access to the service. This network interaction can use any type of networking technology such as RMI, CORBA, or SOAP, because the client only sees the Java technology-based object provided by the service and, subsequently, all network communication is confined to that Java object and the service from whence it came.

When a service joins a network of Jini technology-enabled services and/or devices, it advertises itself by publishing a Java technology-based object that implements the service API. This object's implementation can work in any way the service chooses. The client finds services by looking for an object that supports the API. When it gets the services published object, it will download any code it needs in order to talk to the service, thereby learning how to talk to the particular service implementation via the API. The programmer who implements the service chooses how to translate an API request into bits on the wire using RMI, CORBA, XML, or a private protocol.

2) Our Jini approach

Few numeric approaches exist over the Jini framework. The Jini events are rich data structure and it is quite difficult to manage and to filter all of them. A first work is called JGrid. Because future grid computing systems will be pervasive, invisible, and provide access to a wide range of services, at any time, and from a variety of devices. The scale and nature of next-generation grids pose important research and engineering problems.

The JGrid project at the University of Veszprem (MTA SZTAKI, Eotvos University of Sciences and Sun Microsystems Hungary, and is funded by the Hungarian Ministry of Education) investigates the use of Jini as a potential infrastructure for next-generation grids. The project is motivated by the fact that future-generation grid computing systems will differ from current ones [11]. Future grids will also be used not only for high-performance computing tasks, but also for carrying out business, and assisting society and individuals in many aspects of everyday life.

Our approach differs from JGrid project because our solution is not only dedicated to grid computing. Our mobile agent definition allows us to use agent not only for the implementation of a business algorithm but also for the observation of the algorithm. It means our mobile agents are more polyvalent and adaptable to their context.

Our mobile agent framework consists of two main components. The first component is mobile agent; that is, entities with some task to do. A real application contains several kinds of mobile agent. Each kind has its own task and each agent has its own road map. A task can depend on the location where the agent is. For instance, the class path, which is used for the class loading, is a typical local resource.

The second component is the mobile agent host(s), the service that provides the mobile agents' execution platform. In

a distributed environment, we can have one-to-many agent hosts as well as one-to-many agents. To be an active agent platform, a given node in the system must have at least one active agent host. Figure 1 describes the framework components.

Both agent and agent host have to publish their availability in public registry. This publication contains a record of data, each field of this record qualify a facet of the service: signature, scope, exception case, etc.

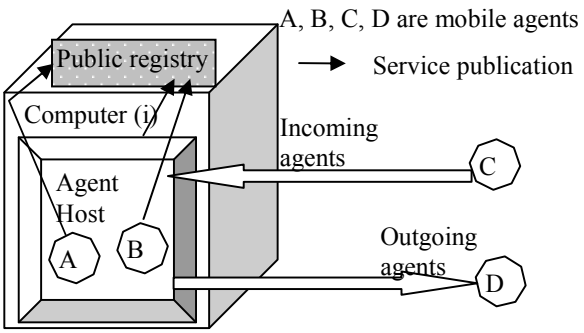

Figure 1: agent framework architecture for one workstation.

To sum up, each node of the network of workstations possesses at least one agent host and a public registry at the beginning of an execution. This agent host has to publish into the local public registry its ability to receive agents. This service filters specific demands of mobile agent. It is also possible to overload the reception service. Each of the services describes a precise process of reception (control, propagation, or eventually measure).

Our framework is based on Java language and Jini technology. The Jini networking system is a distributed infrastructure built around the Java programming language and environment. Jini is the name for a distributed 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 [7]. Jini can be used for mobile computing tasks where a service may only be connected to a network for a short time, but it can more generally be used in any network where there is some degree of change.

The basic communication model is based on the semantic model of the Java Remote Method Invocation system, in which objects in one Java virtual machine communicate with objects in another by receiving a proxy object that implements the same interface as the remote object. This communication model is the core feature for moving agents. The proxy object deals with all communication details between the two processes. The proxy object can introduce new code into the process to which it is moved. This is possible because Java byte codes are portable, and it is safe because of the Java environment's built-in verification and security.

To this underlying communication model the Jini system adds some basic infrastructure and parts of a programming model. The infrastructure provides a mechanism by which clients and services can join into the Jini network (figure 1), while the programming constructs encapsulate mechanisms that allow reliable distributed systems to be built. Java provides the Jini system with a mechanism to move mobile objects, including their code, safely and efficiently from a service to a client of that service. The Java type system forms the basis for identifying services, and its polymorphic nature lets it treat requests for service as requests for something that implements at least a certain type, although the service might offer more. However, the requirement for the Java language and platform is only at the network level; mobile agent programmers can use our Mobile Agent Framework to implement a mobile agent system that can live in the Jini environment.

C. Deployment step

In this section, we explain how a computing context can be initially deployed. It is obvious that some code has to be deployed before the computing code is launched. This step is not a static constraint, it has to be initially done before a workstation (or a node of the network) is used for the computing. Also, if the used net contains three workstations, a first code is deployed on each of the three workstations.

This first code is used to publish a first service (called code manager) in the public registry of each workstation. This service expresses that the node is able to receive a piece of work or a part of the load of the application. It means it is free for computing. Figure 2 shows a monitor code which deploys this first layer.

This exportation is frequently coupled with some input data which are specific to each agent host. For example, a computation has to be done on a very large amount of data. Also, the input data described the bounds of the area which is booked for that node. It can also contain some criteria about the scheduling of the computation: how to converge to a result, a termination condition, etc.

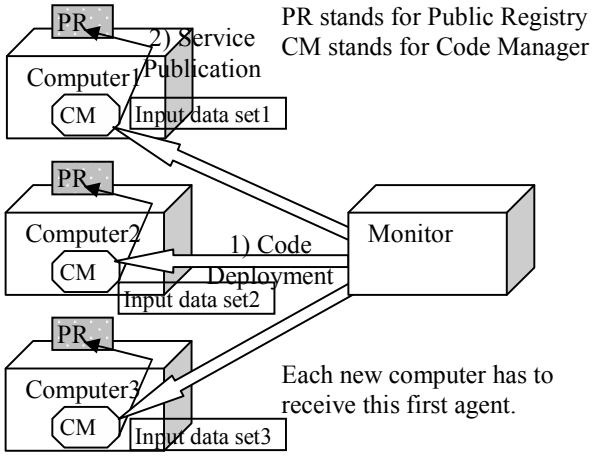

Figure 2: deployment step with three free workstations from a monitor node.

When there are a lot of code managers, it is essential to build a hierarchy of dependency between the code managers for the collect of result or for insuring a better robustness. Also, the input data contains feature about the neighborhood of the node (which is the responsible node, which are the first close nodes to it, etc.).

Now, three nodes are identified as member of a community of agent host where it is possible to realize some computations. Thus, a first case is about the start of the computing code. Because a community of agents is known, the first start of the computing code is realized with that community. But it does not mean that the community is locked. It is possible to grow or to reduce its population of agents.

To start the computation, the monitor calls all free code managers and it uses their ability to migrate computing agent from the monitor onto their host. After this migration, each code manager is responsible of the scheduling. It means that several steps are considered: the reception of a computing agent, the control of its integrity, the start of its task, the end of its task and the management of its need.

The reception of a computing agent is a quite complex task because a migration of agent is not only a code which travels from one node to another, but this agent has also a state and its state travels with it.

The code manager checks the computing agent does not access to private resources and it assigns permissions to this kind of agent. Then, its task is launched. This task can be a local activity or a more complex activity with communication with other agents. The whole lifecycle of the computing agent is managed by the code manager.

Some more details have to be added about the lifecycle of the agent hosts. When a code manager received a computing agent, it changes the status of its published service (in public registry) during the execution of the task of that agent. This means that it is not free for a new computing activity, but it is free for the reception of other agents useful for the computing agent.

When the task of the computing agent is finished, it updates the status of its public service to express that it is available as an agent host for a new computing case.

The notion of community means a group of mobile agents deployed on several workstations. Every ones collaborate to the same computing application. Also when a code manager of an agent host expresses via its own public service that it is free, this involves that it can be included in another community (another computing application).

In the first part of the paper, we chose to define only one code manager per agent host for the simplicity of the explanation. This is not a limitation at all. An agent host can have more than one code manager. In that case it is the support of several computing cases. The figure 3 shows that code managers are useful for isolating each computing case. They play the same role as a class loader. It means that it creates an area for interactions between agents which work on the same computation. But an agent of the same class managed by another code manager has another class (even if the names are the same). This insures safe computations; there is no effect from one execution on to another one.

The number of code manager can be arbitrary limited initially by the monitor, but this limit can be the result of an observation of a load balancing algorithm.

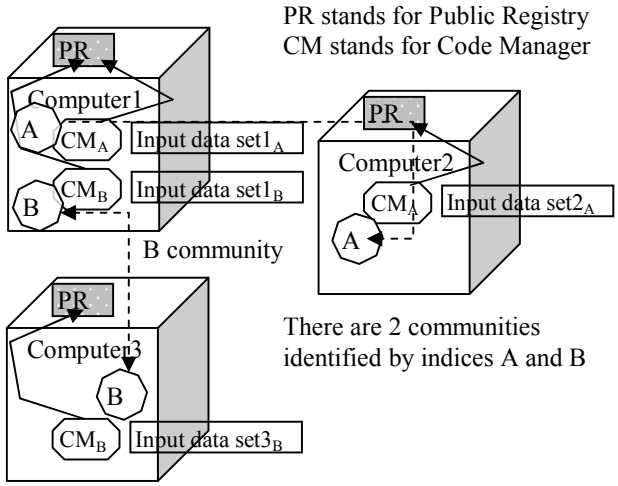

Figure 3: different communities can be shared the same set of agent hosts.

The creation of a community is realized by the monitor and its creation involves the identification of a first set of agent hosts. Also, if there is no creation of community by an agent host, some agent host could be not ready for the reception of a code manager if they have already received code managers of other communities. For instance, this means that the load of an agent host is to important for the execution of anything else. In that case, the new code manager will be received with the busy status. This information means that this code manager can not accept a computing agent for the moment, but maybe later when the load performance of the agent host will allow it.

The code manager takes the role of an observer of the local state of the agent host. It is also a decider about which actions is possible or not. This is a key feature when the computing agent needs some additional data. The scheduling of that management is detailed in the next section.

III. EXCHANGE OF DATA BETWEEN THE MOBILE AGENTS

The lifecycle of a computing agent is structured into several states (figure 4: UML state chart). Depending of the state, functions can be done by this agent. The transitions between the states are managed by the code manager.

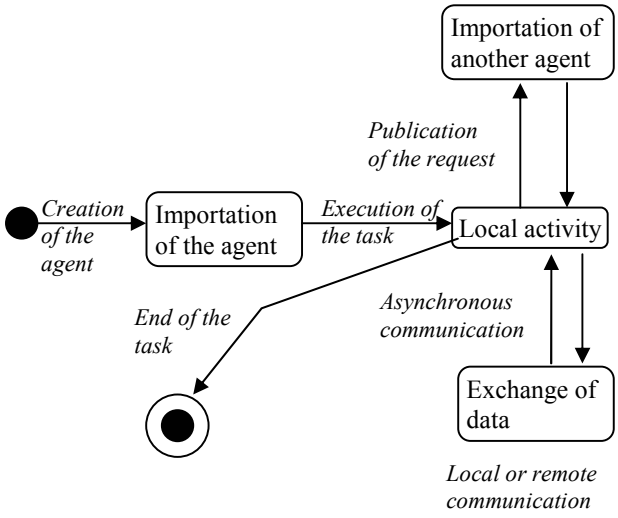

Figure 4: lifecycle of a computing agent

The subject of that section concern both states "Importation of another agent" and "Exchange of data". Both of them are done under the control of the code manager.

A. Importation of another agent

A computing agent can need the help of another one in a lot of case. For instance, before the end of the task of an agent, the results have to be collected and merged with other results. Also, the computing agent can ask its code manager for the reception of a collector agent.

During the task of an agent, a complex and repetitive computation can be done by a specialized agent (for solving temperature equation, for converting Cartesian coordinates into Spherical coordinates, etc.). Also, the agent will ask for the code manager to receive it.

All the requests are buffered and each request corresponds to a service demand which is published into the public registry. If the desired agent is present on the same agent host, it will satisfy the demand when its current task will be finished. If this kind of agent is not on the current agent host, the demand is propagated to the other public registries. Then, the process continues and if an agent host contains such agent, it will answer to the demand.

When several agent hosts answer to a request, the code manager selects the nearest agent host (depending on the input data it received at the beginning). The reception of a response is just an observation; the process of migration has to be done just after. It is quite similar as the first deployment except that it concerns two code managers.

The migration of an agent means that the mobile agent is unpublished from the public registry of the agent host where it was. The code and the state of that agent are serialized to the agent host which asked for it. At the reception of that agent the process is similar as shown figure 4. The interactions between this new agent and the other ones which were already managed by the code manager of the community, local exchanges of data allow setting up the parameters of the task to be done by the new agent. Also interactions will be necessary to have the results of the task and to use them into a more complex work. This is the subject of the next section.

During a migration, before deleting the service of the public registry, a code manager can saved information about the activity of that agent into a log file and some additional information can be saved about the authors of the request, etc. These data could be used for a post mortem analysis.

B. Exchange of data

Two kinds of data exchanges are considered: local or remote. In that context, local means exchange inside a community on the same agent host and remote means exchange inside the same community but on distinct agent hosts. There is no possible communication between two agents which do not belong to the same community. There is only one possibility: the migration of an agent. Also the communities share the same agent power but not the data of these agents.

In our framework, all the communications are asynchronous and their trace can be done. When an agent wants to send data to another one, it does not know where this agent is. It just wants to exchange information. This is why the code manager is the core of the strategy.

An emission is a message which is read by the code manager. In that case its role is like a postman. It reads the destination and publishes the demand about that destination. As before two cases are possible: it is local or not.

If the destination is another local agent, an agent will answer to the code manager that it accepts message from the first agent (a reference to a service of the destination agent). The communication will be done through the invocation of the incoming message service of this agent.

If the destination is remote then a code manager of the same community will answer that the desired agent is on the agent hot, it manages. As before the code manager of the destination will send a proxy reference of the incoming message service of the destination agent. And then, the communication will be done. This incoming message service is common to all the mobile agents. It just manages a buffer of messages with information about them. The figure 5 shows a remote exchange and the insertion into the message box.

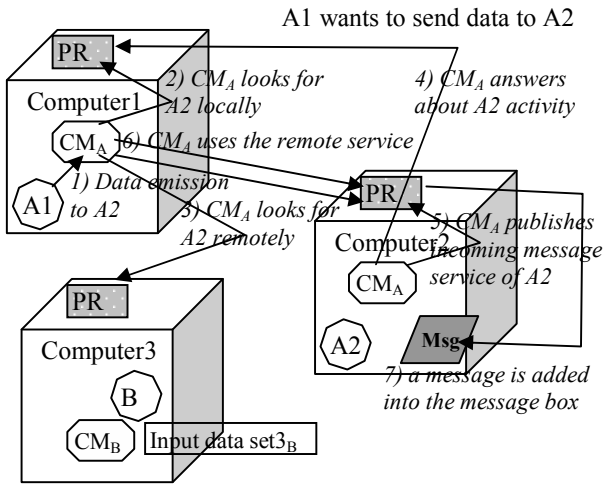

Figure 5: Remote communication between two agents of the same community.

When the code manager does not receive any answer about a possible destination. It means that it does not exist or maybe it is wrong, also the message is cancelled.

Several strategies can be developed when a destination agent is not found. First, it is possible to wait until another code manager answers right information. But this can create a gridlock into the multi agent system. A more common strategy uses timeout. After waiting enough time the request is considered as a cancel action. Then the A1 agent has to decide what it has to do in that context. An exception can be raised or the task can be stopped if the priority of the message is high.

When the receiver is missing, a default behavior can easily be defined as we explained before (lost of information, exception, etc). But the symmetry is not so obvious. When an agent is waiting for the reception of data (for instance in electromagnetism computation where the value of the fields depends on the values of the neighbor cells) starvation of agent can be observed. This is why it is so important to define default value of the message. This allows the receiver to raise exception or to stop its activity if the value is not emitted.

We use in a previous work [8] a concept of evolving canvas. The situation we explained can be more complex because of the mobility of agent. A lot of cases can occur for instance two requests for the same agents occur onto the same agent host. Only one of them will be satisfy also the first code manager which would use the confirmation, will realize the importation of the agent. The other code manager will lose and would have to try again its request. This kind of technical feature is not taken into account by our framework MAFPC but an under layer called JIMA (for Jini Implementation of Mobile Agent); it is a result of a work with Mâamoun Bernichi (L.A.C.L, Paris 12 university).

IV. The framework MAFPC

Our main objective is to provide an API for people who are expert into numerical analysis or Fortran programmer (a lot of numerical application are written into Fortran language). A first version of our framework is based on a set of packages (computation, agent, communication, etc). The main constraint is to hide technical features; the user does not have to be an expert on distributed system or mobile agent system. Furthermore, the user has to convince about the advantages of an object oriented approach.

A. Description of MAFPC API

Numerical programs are often described as a statement diagram and data flow diagrams are frequently used as user documentation.

Experience into numerical analysis allows identifying programming patterns. Some previous work has already studied this subject but it is often limited to object oriented approach (Gof patterns [9], J2EE patterns, Alexander patterns). Numerical analysis is instruction oriented also; this activity domain is not so studied. Effective use of the fundamental object-oriented concepts can improve an application's design. These fundamental concepts are the foundation that object-oriented design patterns and our MAFPC patterns build upon.

The following object-oriented concepts are preserved via the use of our API and all its packages:

- Cohesion: it is the degree to which the methods and attributes of a class focus on only one purpose in the system.
- Encapsulation: an object is a capsule that holds the object's internal state within its boundary.
- Coupling; it is a measure of how dependent classes are on other classes. The tighter the coupling between two classes, the more likely that a change in one class will require a change in the second class.
- Implementation inheritance
- Composition: it builds complex objects from simpler objects, which can be built from even simpler objects. These objects form an object hierarchy where one object uses another as part of its implementation.
- Interface inheritance: it is the separation of an interface definition from its implementation. While the implementation inheritance can cause tight coupling, interface implementation can help maintain loose coupling.
- Polymorphism: it provides the ability to invoke the operations of specific objects through generic references. As a result, polymorphism facilitates writing more generic and flexible code.

We consider few packages of the MAFPC API. A first package called computation contains classes which apply and extend the command pattern. This pattern offers the advantages of separating the use of an algorithm from the algorithm technology code and simplifying the client's dependency on the business logic.

Some basic statements are defined as basic behavior like 3D iteration or 4D iteration. Some more complex statements concern vector and matrices: global sum, reverse.

Another version of strategy pattern is also extended. It allows the programmer to encapsulate a family of algorithms for interchangeable use. This is particularly powerful in matrix computation.

A second package is about the definition of computing agent. We propose several kinds of agent:

- Repetitive agents for the definition of no termination agent like a sensor or an infinite computation
- Collector agent, it is necessary for the construction of a global property or a result at the end of a work,
- Seeder agent: it is essential at the beginning of a global work when each agent has its own task but waits for the reception of the input data. It is the symmetric of the collector agent.
- etc…

A third important package allows the user to define community of agent. Finally a package describes communication schema like a lattice of possible exchange..

B. How to use the API

The concepts of MAFPC API are a guide of developer. For instance, maintaining high cohesion has the following advantages:

- Avoids side affects of changing unrelated code within the same class.
- Improves code readability by clarifying the role of a class.
- Facilitates the creation of small reusable components.

Or maintaining strong encapsulation that includes data hiding has also advantages:

- Allows the implementation of a class to vary without changing the interface that other classes use to access it
- Allows developers of other classes to use the class without knowing all of its implementation details
- Prevents inappropriate modifications of an object's attributes

Implementation inheritance has also interesting:

- Organizes classes according to inheritance relationships
- Avoids duplication of code that is common to all classes of a certain super type

However, implementation inheritance is frequently abused. It is the tightest form of coupling between two classes and has the following disadvantages:

- Forces subclasses to inherit everything from its super class, even when it is not appropriate for that subclass.
- Changes to the super class can affect the subclasses. Whenever the super class is changed, all subclasses should be retested

Polymorphism offers the classical features:

- Enables the user to write generic code that does not depend upon a
- specific subclass allows the user to code fewer methods, because a super type can be specified as the parameter type

For example, the user task of an agent is built as an extension of a predefined class (belonging in computational package). Because of this kind of development, polymorphism insures that user statement will be applied. Also, a task is subdivided into subtasks which can be reused for another user task. The cohesion concept guarantees there is no perturbation between all the parts. But the composition concept of our API allows the user to be sure that this task could be a subtask into a more complex task which will be developed for another agent.

Patterns reuse ideas rather than code. Patterns emphasize practical and proven designs rather than original and invented work. This means that a numerical analysis expert can make design decisions with greater confidence. Rather than reinventing known solutions, the users are free to concentrate on the new features and challenges of a given system.

We use our API for the development of numeric simulation in electronic domain and electromagnetic domain. Of course, these codes are not a new version of an existing one but they are used when adaptability are essential. For instance, reconfiguring a system occurs frequently during a long time simulation (more than one day). In that case, MAFPC is used to add mobile feature to a traditional application. Our patterns enable simple and rapid communication between developers who can communicate through well-understood patterns and pattern names, rather than describing every class or detailed-design decision. This speeds up the adoption or rejection of particular design alternatives.

V. Conclusion

Patterns capture recurring solutions and their trade-offs to similar problems in a given context. Patterns are not simply about the shape of the solution. They are also about understanding where a developer can use a pattern and its consequences. A design or design style that works in one context can be completely inappropriate, and sometimes harmful, when used in a different context.

Because patterns capture the rationale and the structure between the elements in a system, MAFPC user can use its patterns to understand and document existing application. They are guides to understanding existing applications, frameworks, and libraries. Because patterns capture common and repeating designs, they are sometimes invented again or used by habit in developing a system.

Our first application of MAFPC patterns stressed the importance of adding new feature to existing numeric analysis code. Some facets have to be enhanced such that introspection of existing code and data. But we are convinced into our approach and the first results provide new working direction.

References

[1] J. Jones and C. Crickell. Second evaluation of job queuing/scheduling software. Tech. Report NAS-97-013, NASA Ames Research Center, 1997.

[2] F. Berman. High-performance schedulers. In I. Foster and C. Kesselman (eds.) The Grid: Blueprint for a New Computing Infrastructure. Morgan Kaufmann Pub. August 1998.

[3] C. Mascolo, G. P. Picco, and G.-C. Roman. A fine-grained model for code mobility. Tech. Report WUCS-99-07, Department of Computer Science, Washington University in St. Louis, March 1999.

[4] R. S. Gray. "AgentTcl: A Transportable Agent System", Proc. CIKM'95 Workshop on Intelligent Information Agents, 1995..

[5] D. Wong, N. Paciorek, T. Walsh. "Concordia: An Infrastructure for Collaborating Mobile Agents", in Proceedings of the First International Workshop on Mobile Agents, MA'97, Springer Verlag, 1997..

[6] Maamoun Bernichi et Fabrice Mourlin, "A New Behavioural Pattern for Mobile Code ",In ESM 2005, University of Porto, Porto, Portugal, 24-26 October 2005

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

[8] Guide to Jini Technologies," Jan Newmarch (June 2001): http://jan.netcomp.monash.edu.au/java/jini/tutorial/Jini.xml

[9] Gamma, Erich, Richard Helm, Ralph Johnson, John Vlissides. Design Patterns: Reusable Elements of Object-Oriented Software. Reading: Addison-Wesley, 1995.

[10] Fowler, Martin, Refactoring: Improving the Design of Existing Code. Addison Wesley, 2000.

[11] Zoltán Juhász, Péter Kacsuk and Dieter Kranzlmüller, "Parallel Program Execution Support in the Jgrid System", the International Series in Engineering and Computer Science, Distributed and Parallel Systems, Cluster and Grid Computing, December 31, 2005