Distributing Knowledge-Based Systems Using Mobile Agents

NAOUFEL KHAYATI[*1,**] — WIDED LEJOUAD-CHAARI[*1] — SABINE MOISAN[**] — JEAN-PAUL RIGAULT[**]
* ENSI Tunis, Campus Universitaire La Manouba
[1]SOIE Research Unit, ISG Tunis
TUNISIA
** INRIA Sophia Antipolis – Orion Project
FRANCE
{naoufel.khayati, sabine.moisan, jean-paul.rigault}@sophia.inria.fr ; wided.chaari@ensi.rnu.tn

Abstract: - The objective of our work is to study various distribution methods for knowledge-based systems (KBS) allowing access to (several) remote knowledge bases as well as remote auxiliary programs or inference engines. In particular, we are interested in knowledge-based systems for a task named *program supervision*, with an application to medical imagery. Program supervision KBSs allow to automation the use of complex programs, independently of any particular application domain. They offer original techniques to plan and control program processing activities. The distribution of such systems becomes essential because real applications imply more and more participants on various sites. Given distributed data, programs and knowledge, our aim is to propose convenient and efficient models of distributed program supervision systems. Our current application relates to medical assistance in the use of medical imagery programs for osteoporosis detection in bone radiographies. A distributed approach will allow to execute distant physician requests using an existing system and will favor collaborative work and sharing of knowledge to build new knowledge-based systems. In this paper, we propose a distributed architecture based on mobile agents for program supervision KBS and we show why and how to use mobile agents for such systems. Some scenarios are given to illustrate the functioning of the distributed system, in particular when solving a user-request in medical imagery domain so that its management remains transparent to physicians.

Key-Words: - Distributed Knowledge-Based Systems, Program Supervision, Mobile Agents, Medical Imagery, Osteoporosis detection.

1 Introduction

Many libraries of programs have been developed by specialists in various domains, but the end-users of these libraries do not necessarily master the programs and thus cannot use them in the most effective way. So programs and knowledge on their use must be accessible to non computer specialists and especially to specialists in the application domains of the programs. A solution is to develop systems able to manage the use of these libraries, freeing users from this know-how and allowing them to focus on the interpretation of the results. We thus propose to design program supervision systems which automate intelligent use of programs. Such systems meet well the needs for service sharing which is necessary in several areas like Medical Imagery (e.g. chemotherapy follow-up based on Factorial Analysis of Medical Image Sequences [3]), Astronomical Imagery (e.g. automatic galaxy classification [14]) and Vehicle Driving Assistance (e.g. obstacle detection [11]). Our current application is related to assist physicians in the use of medical imagery programs, more precisely for osteoporosis detection in bone radiographies.

A program supervision system allows end-users (physicians in our case) to run programs, to check the consistency of image analysis methods, to compare algorithms, to evaluate results, to reconsider some parameters, and to readjust them.

Physicians can be in different locations, programs and knowledge on their use can be written by different persons and thus located on distant machines. That is why distributing program supervision systems is interesting. It allows either to simply consult existing distant knowledge bases, or to collaboratively construct new knowledge bases, or to launch a request on a distant KBS with local data. Our goal is to propose solutions for cooperation and sharing that allow teams to share medical imagery programs and knowledge on their use, and thus to benefit from the work of other teams without revealing the source code.

In this paper, we start by defining program supervision systems and situating program supervision according to program reuse. Then, we introduce and discuss the interest of the distribution of such systems. In section 4, we propose a distributed architecture based on mobile agents for a program supervision system and we discuss the used mobile agents' model. Finally, we focus on example scenarios illustrating the functioning of our distributed system.

2 Program Supervision Systems

2.1 Program Supervision and Program Reuse

From one project to another, it is not rare to rewrite already existing pieces of code. This systematic rewriting involves extra time and fees. In order to benefit from past work, reuse techniques to build catalogues of reusable components have appeared [9]. This form of reuse is rather static, i.e. software engineers can reuse the source of some codes or programs and insert it in another application. The program supervision approach aims at a more dynamic reuse of entire programs by end-users who need to run them in the best possible way to process their data.

To this end, we developed environments [4, 13] in order to automate the dynamic use of various programs, or program libraries. These systems make programs accessible to users who don't master them without modifying their code. This form of intelligent reuse [8, 10] is based on Artificial Intelligence techniques and is known as Program Supervision.

A Program Supervision System (Fig.1) is a Knowledge-Based System which ensures the selection and sequencing of programs in various configurations, thanks to the reasoning strategy of its engine and to the knowledge contained in its base. This allows an intelligent reuse of programs and makes them accessible to users who are not specialists of the techniques and algorithms coded in these programs.

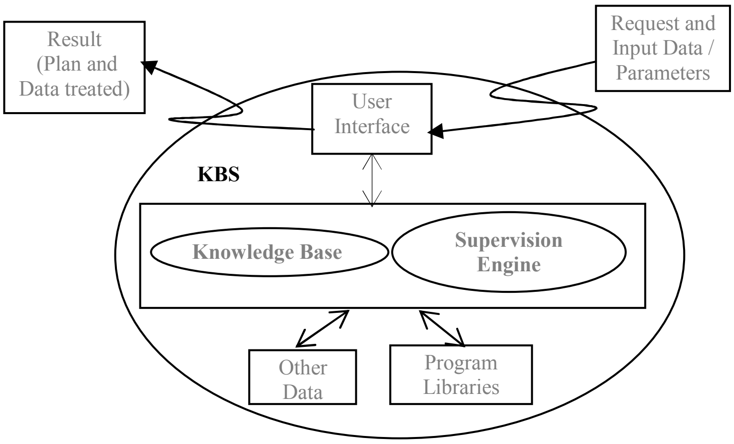

Fig. 1. Components of a Program Supervision System

Such a system is composed, as any KBS, of a knowledge base which contains the know-how on the use of the programs and of a supervision engine which uses this know-how to build an execution plan of the programs and to run it in order to obtain the results. The engine must have all knowledge to plan the programs, to run them automatically, to launch their execution, to produce results, to evaluate them, and to know which corrective actions to undertake (re-planning or re-execution of the current stage) in the event of bad quality results. Moreover, a program supervision KBS also involves a set of programs to be planned and adapted to a precise application domain, a set of data (images in our case) to be processed, and a graphic interface making it possible for users to express their objectives, to follow executions and to see results.

2.2 Modelling for Program Supervision

In order to correctly use programs, an end-user should have in mind:

- A model of programs, i.e. for each program, understand its objective and its behaviour, determine the number and the types of its arguments and its calling syntax.
- A model of possible combinations of the programs, i.e. know which programs can be combined together, what are the data flows between them, remember useful combinations for standard treatments and know how to choose between several alternatives.
- A model of repair strategies, i.e. if during the processing, a program produces an unsatisfactory result, be able to infer the program causing this problem and to decide either to re-execute it with new parameter values or to replace it by another. In the first case, one must be able to find the new values and in the second case to decide of the new program.

A program supervision system thus relies on the same type of models -- encapsulated in its knowledge base -- to automate the planning and the control of execution of programs in various situations.

3 Distributed Program Supervision

Distributed Program Supervision Systems (DPSS) should offer services, on the one hand, to distant users who wish to process their data using program supervision facilities and, on the other hand, to experts and designers who wish to share programs and knowledge. The latter must work in a collaborative way, i.e. must be able to consult information on existing programs, to create new common knowledge bases or to update existing ones by introducing new programs or new knowledge. Supervision environments were originally conceived to be mono-site. However, the components of a supervision environment (see § 2) can be located on various sites. Not only codes and/or data can be on different sites, but parts of the engine itself can be delocalized, for performance reasons, for hardware characteristics, etc. [8]. A mono-site environment is no longer sufficient, because it implies to install all the components, in particular the programs and the complete knowledge base, on a single site, which is not always possible or desirable. On the one hand, installing and maintaining codes remain a heavy problem which requires time and competence. In addition, programs are sometimes not very portable and difficult to install (because they require other utilities or they depend on development environments or compilers). On the other hand, when users on various sites create (parts of) knowledge bases, they in fact develop a competence that should be easily accessible to a user community using similar techniques. The repatriation of this knowledge on all the user sites would induce coherence and maintenance problems. The setup of architectures and services for distributed resolution of program supervision problems can be done by installing knowledge servers which allows to share and to disseminate knowledge on multiple client sites. Each site can allow the use of the resources (programs and knowledge) of other sites in a transparent way. Each site thus becomes a client of the competence of others and a server of its own competence. Consequently, if programs related to the same problems are developed in disseminated teams, distributed supervision can allow a cooperative resolution of problems, in which each stage represents the know-how of a team. It also allows researchers to confront their experiments and to enrich their results. For that purpose, we started studying various distribution methods for program supervision systems. Each form of distribution generates different problems due to the size of the data to be transferred, to the heterogeneity of the languages and development environments, to the specific needs of resources for the execution of some programs, to the management of knowledge coherence, etc.

4 Distributed Program Supervision System Architecture

During former work, we developed a supervision server via the Web, named SPI, which allows the remote consultation and modification of knowledge bases and provides an authenticated access to different users. It manages their requests, repatriates data, delegates processing, recovers the results and returns them to the user. SPI is currently under development to improve the management of concurrent accesses and to maintain the coherence of the knowledge bases.

The architecture we proposed for distributed program supervision systems (Fig.2) includes a set of program servers, a set of knowledge servers, a supervision engine called *Pegase* [12] and the supervision server SPI. This latter plays the role of an interface between the supervision system and end-users by allowing the communication with the other components of the distributed system.

This architecture is also equipped with a metadata warehouse (used to locate the various resources, to define access permissions, etc.) and is managed by mobile agents which are responsible for updating the previous components and for performing requests.

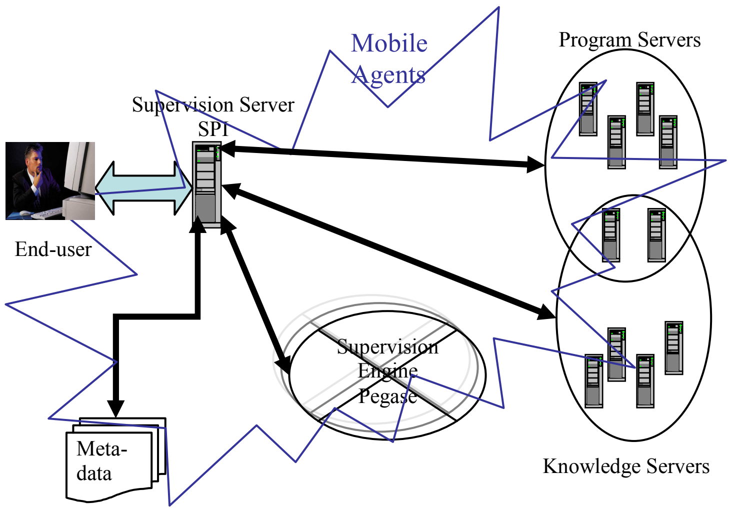

Fig. 2. Distributed Program Supervision System Architecture

4.1 Why Mobile Agents?

Any distributed application implies a multi-localization of the handled entities and requires the mobility of certain of them. The choice of the entities to be moved raises problems of technology (how to carry out mobility?) and requires a size/time compromise (moving too large entities involves a time penalty). The Multi-Agent Systems and particularly Mobile Agents can be adequate solutions to this problem.

Lange and al. [7] defined seven good reasons for using mobile agents. All these reasons apply in Distributed Program Supervision:

- Solving a supervision request requires multiple interactions between the supervision engine and the different servers. The result is a lot of network traffic. Mobile agents allow these interactions to take place locally and no via the network.
- Our system deals with large size data (images), so it will be better to process these images locally because it is generally less expensive to move programs than images.

- Our distributed system must allow the sharing of heterogeneous resources (programs, data and knowledge) by facing hardware and software heterogeneity (interaction mechanisms with the servers, data format). In this case, mobile agents which are only dependent on their execution environment (generally computer-independent and transport-layer-independent) are a suitable solution.
- If a distant host (involved in the supervision process) is being shut down while our system is solving a supervision request, we risk that the treatment will be stopped and the request will never be solved. Mobile agents are able to react dynamically to this unfavorable event (and others) and dispatch themselves to continue their operation on another host if possible.

4.2 Mobile Agents Model

When performing a request, our system needs different classes of agents to communicate between the different components of the architecture: a *Supervisor* agent, several *Solver* agents and possibly *Evaluator* agents. Only the two last ones are mobile. Each class of agent is characterized by its behaviour and its features. This section also deals with agents security.

4.2.1 Agents Behaviour

The *Supervisor* agent is associated to the *Pegase* supervision engine. It has a triple role:

- Determine the number of necessary *Solver* agents to solve the user request and create them. This is possible if we have independent or parallel treatments, if not, only one *Solver* is necessary.
- Plan their itinerary.
- Facilitate the communication and the interaction between the mobile agents (*Solver*s and *Evaluators*), the engine and the supervision server.

Solver agents are created by the *Supervisor* and have to execute distant programs planned by the supervision engine and communicated by the *Supervisor*. For the migration of these agents from a machine to another, different policies can be considered:

- Leave the programs where they are, make the data migrate towards program sites and recover results to transmit them to the end-user.
- Take the programs (sometimes lighter than data) and execute them on the data hosts.

For example, for the first case, when migrating, a *Solver* agent brings necessary data and parameters for the execution of a planned program, performs this program and stores in its context the result and the execution parameters for the next programs.

Evaluator agents are created on the program sites (by *Solver* agents) when needed, i.e. when a program requires evaluation of its results. They correspond to the evaluation phase of the supervision process. For their migration, they move from a program server to the supervision engine, if the evaluation is automatic or from a program server to the user site if the evaluation requires user interaction (manual evaluation). They store the result to evaluate in their context. If the evaluation is manual, the user assessment has to be sent to the *Supervisor* to allow it to decide of the next step.

4.2.2 Agents Features

The *Supervisor* agent is characterized by a local memory containing a dynamic list of its acquaintances (*Solvers* and *Evaluators* with whom it communicates), a request to be solved (a functionality), a list of the knowledge files involved in the supervision process and the generated plan (or part of plan).

In each stage of the supervision process, a *Solver* agent is characterized by a local memory containing a dynamic list of distant programs to carry out and their respective arguments, a dynamic list of hosts to visit (itinerary) and a list of its acquaintances, namely its *Evaluator* agents with which it communicates, the *Supervisor* agent and other *Solvers* if they exist. Moreover, it has the input data for the next step and the result of the previous one.

An *Evaluator* agent is characterized by a local memory containing a result to evaluate, the kind of the evaluation (automatic or manual). Moreover, it is capable of deciding the site to which it must migrate (engine or user site) and a list of its acquaintances, namely the *Solver* agents with which it communicates and the *Supervisor*.

4.2.3 Agents Security

In our work, we use the Aglets platform which is a Java-based programming environment for mobile agents. Thus, this platform takes advantage of the security mechanisms offered by Java. In Addition, Aglets provide a *Security Manager* responsible for protecting hosts and agents against spiteful entities and offers a control for writing and reading accesses to local file system.

There is another issue, related to security at the semantic level concerning program supervision in general and our current application in particular. Since we are dealing with medical applications, confidentiality of the patients' information and diagnosis results is of the highest importance. To this end, we need to complement the knowledge bases with additional information related to this kind of security and confidentiality. Defining the corresponding metadata is a work currently in progress.

4.2.4 Agents Access to Knowledge

A program supervision knowledge base can be distributed and in this case, concurrent accesses problems may arise (see § 5.1.4). Another choice is to embark the necessary knowledge into agents, and thus to decide which piece of knowledge is necessary for which agent. The latter point is a task on which we are currently reflecting.

5 Processing Scenarios

The users of our systems must authenticate themselves to be connected to the supervision server. Consequently, they get their specific access rights, objectives and tasks to be carried out.

Thus, we have users who want to benefit from the supervision services in medical imagery and others who want to remotely update some components of the distributed system (programs, knowledge bases, etc.). We distinguish users of different types:

- Physicians who are "simple users" interested only in the resolution of medical imagery requests;
- Physicians who are "expert users", i.e. having some knowledge in computer sciences and who are also interested in consulting programs and visualizing knowledge bases in order to provide possible suggestions according to their expertise;
- Specialists in image processing who have the capability to update the system by adding, modifying or withdrawing programs.
- Designers of the knowledge bases who update the system after an update carried out by specialists in image processing by adding, modifying or withdrawing knowledge on the use of programs.

So within the framework of distributed supervision, we can distinguish the following functionalities which can be offered by our system, namely:

- consultation and update of the programs and knowledge bases;
- resolution of user requests.

In what follows, we will propose a possible scenario for each above mentioned functionality.

5.1 Scenarios for Consultation and Update

Any user has the right to consult a program or a knowledge file and has the right to visualize a knowledge base. On the other hand, only specialists in image processing can update programs and only knowledge bases designers can update knowledge on the use of these programs.

In order to deal with these operations, the server offers to users an interface enabling to enter key words or to choose in a list the required element.

5.1.1 Consult Programs

When a user wants to consult one or more programs, he must enter:

- the name of the program if he already knows it, or
- a set of key words referring to the program functionality.

Thereafter, the server must be able to seek in the metadata warehouse, the corresponding programs to the user research and must return a link towards the physical site of each program. Visualisation is done in a Web navigator.

5.1.2 Consult Knowledge Files

The user, who wants to consult knowledge files, must enter:

- the name of the file if the user knows it, or
- a set of key words referring to the functionality described by these files or referring to an operator described in these files.

Then, the same principle as previously (see § 5.1.1) will apply.

5.1.3 Visualize Knowledge bases

In this case, the user has to select a knowledge base in a list. A graph, like the one given in figure 3, is posted in a Web navigator.

5.1.4 Update Programs and Knowledge Files

To update a file, the system must be able to locate it on a distant site thanks to metadata information. For this research, the system follows the same principle used for consultation. Once found, the file will be open in edition mode. For a program written in a programming language (C++, MatLab, etc.), the user, being familiarized with this language, can handle it directly. On the other hand, for knowledge files, update is possible through a simple interface allowing for example to:

- add or remove the description of a new program,
- add or remove parameters, data, criteria, etc.
- modify the program state, for example, a program which was optional becomes obligatory or the reverse.

It should be noted that to update these files (programs or knowledge), which may be shared by several users, it is necessary to manage the concurrent accesses inside the distributed system.

A student project is currently dealing with a concurrent accesses management system based on CVS. CVS provides us with textual coherence management, which is useful but not sufficient within the framework of supervision systems where knowledge files are written using a specific expert knowledge description language. This work must add a syntactic coherence to ensure that the knowledge base is always in conformity with the description language syntax (by calling the analyzer of this language).

It is also advisable to guarantee a semantic coherence of the base, which will require even more work.

5.2 Scenario for a Medical Request Resolution

Let us first define two major concepts of the *Pegase* engine: operators and criteria.

- Operators are of two types: primitive and composite. A primitive operator represents a particular program and a composite operator represents a combination of programs. Combinations of programs correspond to decompositions into more concrete operators at various levels of abstraction, either by specialization (alternatives), or by composition (sequences, parallels, etc.).
- Criteria represent decisional information, they are implemented by sets of inference rules which play an important role during the reasoning, i.e. choosing between various alternatives (choice criteria), adapting the programs execution (initialization criteria), diagnosing the results quality (evaluation criteria), and repairing a bad execution (repair criteria and adjustment criteria). These rules are written by experts on the use of the programs.

The following scenario describes the way a request is solved by a DPSS. Let us consider for example a request OSTEO for osteoporosis detection in bone radiographies by a mathematical morphology approach.
Once connected to the SPI supervision server (after authentication), a physician can launch a request by first selecting the knowledge base to be questioned from a list and specifying the input data that he/she wants to be processed. Receiving the request, the server transforms it into a program supervision purpose i.e. in a functionality manageable by *Pegase*. Referring to the metadata contents, the server deduces that OSTEO can be solved thanks to the "Morphology" functionality carried out by the composite operator "OsteoMorpho". Finally, the server creates a *Supervisor* agent and gives him all necessary information (functionality to achieve and input data).
Since the *Supervisor* agent is an interface between *Pegase* and the future agents, it has to forward the previous information to the engine. Now and thanks to the metadata, *Pegase* becomes able to locate all the necessary knowledge to start the supervision process for OSTEO resolution following the steps below.

5.2.1 Planning Phase

The supervision engine starts its supervision process by building a plan or a part of plan for executing the programs.
To this end, *Pegase* begins by decomposing operator "OsteoMorpho" in other more concrete operators (composite or primitive). Fig.3 presents a graph of the osteoporosis detection base, showing the decomposition of operators. "OsteoMorpho" breaks up into a sequence of operators: *lecture, squelettisation* and *analyse*. To be able to decide the number of *Solver* agents and the triggering of the planned operators, we take into account, during the planning phase, the dependencies between programs i.e. whether the results of a program are inputs to some further programs.

5.2.2 Execution Phase

The first operator "*lecture*" is primitive, so it has to be executed. Thus, a *Solver* agent will be created by the *Supervisor* and informed of the following parameters: the program name, its location and the evaluation. The evaluation parameter can be *No* if there is no evaluation for this program, *Man* if the evaluation is manual and *Auto* if it is automatic. Note that "lecture" needs no evaluation. The agent migrates to the distant host of program "*lecture*" having in its context, in addition to the preceding parameters, the input data to be treated.
While the *Solver* is running, *Pegase* continues with a new planning phase by breaking up *"squelettisation"* into two alternative operators: *squeletteBin* or *squeletteGris*. The choice between these two operators can be carried out by the user or by *Pegase* itself, according to existing choice rules.

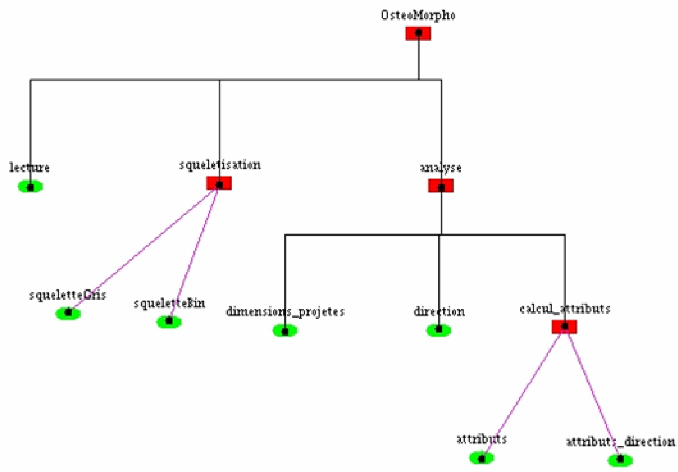

Fig. 3. Osteoporosis Detection Base

- If the choice is automatic, *Pegase* uses choice criteria to decide the operator to plan.
- If the choice must be carried out by the user, the engine sends to the server (via the *Supervisor* agent) a request for choice which can be in the form of a question "Do you want to carry out a binary skeletonization or a skeletonization in grey levels?". The answer of the user will be read by *Supervisor* and transmitted to the engine so that it takes into account this choice in its planning.

Let us suppose that according to the last choice, the *squeletteBin* program is planned.
At this level, the skeletonization requires an evaluation; therefore, the *Supervisor* agent sends the name of the program to be carried out (*squeletteBin*), its location, and the evaluation parameter to the distant *Solver*.
Receiving these parameters, the *Solver* agent migrates (Fig.4) towards the *squeletteBin* site bringing with him the result of "*lecture*" execution. Once the execution of the program is finished, it is time to pass to the evaluation phase of the results.

5.2.3 Evaluation Phase

This phase is handled by the *Evaluator* agents. Such an agent will be created by the *Solver* and informed of the result and the type of the evaluation so that it may decide where to migrate next (Fig.4):

- If the evaluation is automatic, it migrates towards the site of the supervision engine;
- If the evaluation is manual, it migrates to the user site to ask for an answer. The *Evaluator* finishes by sending this opinion (assessment) to the engine.

If the assessment is positive, the engine continues its planning with the composite operator "*Analyse*" and transmits its plan to the *Solver* via the *Supervisor*.
If the assessment is negative, the engine will use repair or parameter adjustment criteria to decide to start planning again or to re-execute the same plan with different parameters.

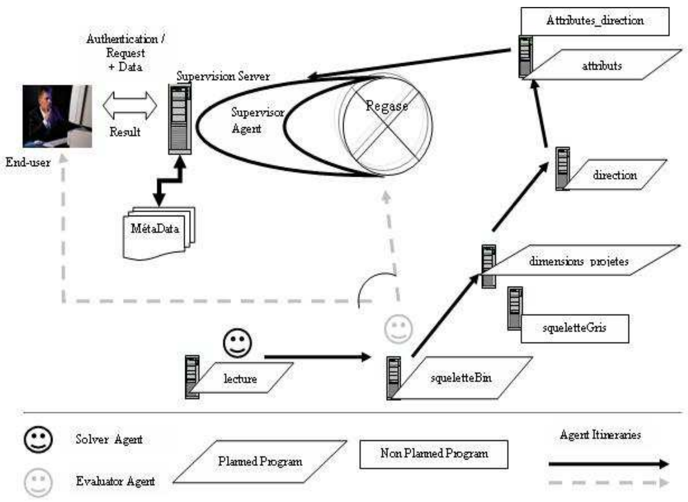

Fig. 4. Example of Agent Dispatching for Osteoporosis Detection

5.2.4 Repair Phase

If it has repair criteria, the engine will repair the previous plan, for example, by adding an optional operator not previously considered or by choosing another operator if there were a choice. In this case, re-planning takes place and the *Solver* will be informed of the new plan and will behave in the same way as with the previous plan.
If it has adjustment criteria, and according to them, the engine readjusts some parameter values of the current program. This readjustment can be automatic (made by the engine) or manual. In case of manual readjustment, the engine transmits a request for parameter adjustment to the supervision server, such as for example: "To repair the plan, do you want to change the parameter *P* from 10 to 9 or to 8?". The server forwards this request to the user, gets the answer and then gives the adjusted parameter value to the *Solver* via the *Supervisor*. The *Solver* runs its program again with the updated value(s) of parameter(s).

This process is repeated until the last program (*attributs* or *attributs_direction*) is executed. The *Solver* agent carries out the chosen operator (we suppose "*attributs*") while being informed that it is the last program so that it can go back to the *Supervisor* site bringing with it the final result (the response to the request). Finally, the *Supervisor* transmits this result and the generated plan (*lecture – squeletteBin – dimensions_projetes – direction – attributs*) to the end-user.

Thus, and thanks to the DPSS, we could assist physician in his osteoporosis detection diagnosis by sequencing and executing a set of programs which use a mathematical morphology approach and about the contents of which the physician has no idea.

6 Extension of the Architecture

As an enhancement of the presented architecture, we can propose its integration in a larger one to allow the participation of a large number of disseminated concerned persons (physicians, for example).
In order to establish an adequate larger architecture for distributed supervision systems, we studied some network techniques such as Peer-to-Peer technology [6] and Grid Computing technology [5, 6]. Grids offer relatively sophisticated services and applications; they usually connect a few sites collaborating for complex scientific applications. On the other hand, P2P systems involve much more participants and offer unsophisticated, limited and specialized services such as file sharing. In our case, distributed supervision systems must be powerful, do not have to be limited to simple operations of file exchange but must rather offer complex functionalities such as execution of distant programs and access in various modes to data and knowledge files; it must also be able to parallelize processing for time reasons, as for example, when we want to apply the same algorithm to the various segments extracted in a radio image, given that an image

can include between 2000 and 4000 segments. Consequently and contrary to [1] which proposes a Peer-to-Peer architecture for the distributed knowledge management, we prefer a Grid one since it offers richer strategies making it possible to accelerate processing and to increase collaboration.

As chosen by Cao and al. [2], we suggest also an architecture combining Grid and Agents. But, we will not use agents for the resource management only (knowledge bases, programs, etc.), but also for performing some complex tasks like the execution of distant programs.

7 Conclusion

Supervision environments were originally conceived to be mono-site. However, their interest and the nature of their components made their distribution necessary in order to offer services to different distant users: specialists in image processing who can try out and compare their programs, knowledge base designers who can describe the use of these programs and physicians who can use these programs through the Web and thus improve their means of diagnosis.

Moreover, the interest of this work was felt by various teams in different domains and especially in medical imagery.

In this paper, we proposed a distributed architecture for a program supervision system based on mobile agents. We showed how such a system behaves to solve a request of osteoporosis detection.

To extend our architecture, we intend to integrate it in a Grid architecture and thus to combine a network technology for distribution (Grid computing) and an artificial intelligence technology for distributed processing (mobile agents). Forthcoming work will concern the integration of new programs from different teams in our distributed system and the use of ontologies to simplify the search for resources in the distributed supervision system.

References:

[1] Bonifacio M. and al., A peer-to-peer architecture for distributed knowledge management, *In Proc. of the 3rd Intl. Symposium MALCEB'2002,* Erfurt / Thuringia, Germany, 2002.

[2] Cao J. and al., ARMS: An Agent-based Resource Management System for Grid Computing. *Scientific Programming*, 10(2), 2002, pp. 135-148.

[3] Crubézy M., Aubry F., Moisan S., Chameroy V., Thonnat M. and Di Paola, R., Managing complex processing of medical image sequences by program supervision techniques, *In Proc. of SPIE Medical Imaging 1997*, Newport Beach, CA, Vol. 3035-85, , 1997, pp. 614-625.

[4] Crubézy M., Marcos M., Moisan S., Experiments in Building Program Supervision Engines from Reusable Components, *In ECAI'98 Workshop on Applications of Ontologies and Problem-Solving Methods*, Brighton, UK, August 1998, pp. 44-53.

[5] Foster I. and al., The Anatomy of the Grid: Enabling Scalable Virtual Organizations, *Intl. Journal of Supercomputer Applications and High Performance Computing*, 15(3), 2001.

[6] Foster I. and Iamnitchi A., On Death, Taxes, and the Convergence of Peer-to-Peer and Grid Computing, *2nd Intl. Workshop on Peer-to-Peer Systems (IPTPS'03)*, Berkeley, CA, 2003.

[7] Lange D.B. and Oshima M., Seven good reasons for mobile agents, *Communications of the ACM*, Vol.42, No.3, 1999, pp. 88-89.

[8] Lejouad W., Etude et application des techniques de distribution pour un générateur de systèmes à base de connaissances. *PhD Thesis*, Nice University, 1994.

[9] B. Meyer, *Conception et Programmation par objets : Pour du logiciel de qualité*, chapitre 3 : « objectif réutilisabilité », InterEditions, 1990.

[10] Moisan S. and Lejouad-Chaari W., Réutilisation intelligente de programmes de vision en environnement distribué, *TAIMA'01*, Hammamet, Tunisia, 2001.

[11] Shekhar C., Moisan S. and Thonnat M., Real-Time Perception Program supervision for Vehicle Driving Assistance, *In Okyay Kaynak, Mehmed Ozkan, Nurdan Bekiroglu, and Ilker Tunay, editors, ICRAM'95 Intl. Conference on Recent Advances in Mechatronics,* pp. 173–179, Istanbul, Turkey, 1995.

[12] Thonnat M. and Moisan S., What can Program Supervision Do for Software Reuse?, *IEE Proceedings-Software. Special Issue on Knowledge Modelling for Software Components Reuse*, 147(5), 2000, pp. 179-185.

[13] Thonnat M. and Moisan S., Knowledge-Based Systems for Program Supervision", *In Proc. of KBUP'95*, INRIA Sophia Antipolis, France, 1995, pp. 3-8.

[14] Vincent R., Thonnat M. and Ossola J.C., Program supervision for automatic galaxy classification, *In Proc. of the Intl. Conference on Imaging Science, Systems, and Technology, CISST'97*, June 1997.