Contracting and Moving Agents in Distributed Applications Based on a Service-Oriented Architecture

B.Schulze **E.R.M.Madeira**

Relatório Técnico IC - 96 - 20

Dezembro de 1996

Contents

Contracting and Moving Agents in Distributed Applications Based on a Service-Oriented Architecture

B.Schulze[1, 2], E.R.M.Madeira[1]

{schulze | edmundo}@dcc.unicamp.br

1. Institute of Computing / Unicamp PO Box 6176 13083-970 Campinas, SP - Brazil.

2. Brazilian Center for Physics Research / CNPq, 22290-180, Rio de Janeiro, RJ - Brazil.

Abstract

This paper presents a service-oriented platform for development and execution of distributed applications based on contracting stationary and migrating services. Services are seen as active objects build on top of a middleware using CORBA and added features. Customized services add to the middleware the ability to handle transparently application start-up and distribution according to load-balancing and inverse caching application demand. Services can be considered of any kind ranging from scientific specialized processing to data archiving juke-boxes. An application on system management in scientific experimental environment is driving the work on some aspects of the architecture and the management.

keywords: load-balancing, agents distribution, ORB, distributed processing, service-oriented architecture.

1. Introduction

A service-oriented application should make use of available services as much as possible and start new services when they are not available. Just like in any other environments one can make use of services from the shelf and self customize what is not available.

Some of the aspects regarding the development and execution of such a service-oriented application include: initial contracting of services, distribution and start-up of additional services needed by the application but not encountered anywhere inside the group of contracted servers. At runtime the application must also handle: querying and replying of the services, and eventually substituting some failing service.

Finally the application has to be considered being shutdown at some point. The application considered is oriented to Distributed System Management in scientific environment and is organized in services related to the managed environments. Specific services to support the proposed architecture are redistributed during runtime, regarding load-balancing and inverse-caching [Goldszmidt96], i.e., code is moved close to data.

Section 2 presents some insights from AI and OO suggesting simplifications to distributed programming. Section 3 contains a description of the proposed architecture based on services as active objects. Section 4 describes implementation details of the service-based architecture while Section 5 contains concluding remarks and acknowledgments.

2. Insights from AI and OO

An interesting notion is the one of *components* [Orfali96] as stand-alone objects that can plug-and-play across networks, applications, languages, tools and operating systems. Distributed objects are, by definition, components because of the way they are packaged.

An analogy [Russel95] regarding knowledge engineering and programming gives an interesting insight to the paradigm of programming at a higher level of abstraction. Table.1 is a reproduction presenting four basic steps:

Knowledge Engineering	**Programming**
Choosing a logic	Choosing a language
Building a knowledge base	Writing a program
Implementing the proof theory	Choosing or writing a compiler
Inferring new facts	Running a program

TABLE 1. Four steps in programming [Russel95].

The approach is that it requires less work deciding only what objects and relations are worth representing and which relations hold among which objects. There is no need to compute de relations between objects.

There is the need only to specify what is true while an inference procedure figures out turning facts into a solution of the problem. If we consider that in a same context a fact is true regardless of what task is trying to be solved, then knowledge bases can be reused for a variety of different tasks without modification.

The debugging task is expected to be easier because any given sentence is true or false by itself, while the correctness of a program statement depends strongly on its context.

The notions above introduce the field of agent-based software programming [Genesereth94], as an attempt to make all sorts of systems and resources interoperable by providing a declarative interface based on first-order logic.

2.1.Distributed Problem Solving and Multi-Agent Systems

The DPS approach uses distributed computing environment to solve problems which are naturally distributed while complex. Agents are pre-programmed for cooperation with methods to guaranty this under coherence, robustness and efficiency. The quality of a DPS system is the measurement of the system global performance in solving the specific problem.

Since experiences in Social Sciences demonstrate that it is not simple to establish this new properties in a collection of individuals, MAS studies the basic assumptions about agents which should guaranty the possibility of a cooperative action in society.

In a Multi-Agent System [MAGMA95], agents range from simple automata to knowledge-based systems, while interactions between agents go from physics-based models to Speech Acts. Agents organizations are incorporated into complex systems and environments are guided by the type of application.

2.2.Agent Classification

“The notion of an agent is meant to be a tool for analyzing systems, not an absolute characterization that divides the world into agents and non-agents.” [Russel95].

An agent need not to be a program at all [Franklin96], but software agents are, by definition, programs that must measure up to several marks to be an agent, presented in table.2:

Property	Other Names	Meaning
reactive	(sensing and acting)	responds in a timely fashion to changes in the environment
autonomous		exercises control over its own actions
goal-oriented	pro-active purposeful	does not simply act in response to the environment
temporally continuous		is a continuously running process
communica-tive	socially able	communicates with other agents, perhaps including people
learning	adaptive	changes its behavior based on its previous experience
mobile		able to transport itself from one machine to another
flexible		actions are not scripted
character		believable “personality” and emotional state.

TABLE 2. Agent classification [Franklin96].

2.3.Problem Solving Architecture

An ideal architecture would be able to solve any problem by: knowing everything and being able to interact with any other system. However to be practical an architecture has to come to a good solution to a problem of a particular environment and be able to communicate accordingly in order to sense and react.

Communication is an important example of the range that exists between extremes that an architecture has to fit in. If one considers security, there should exist a secure encryption while considering interoperability there should exist a general de-encryption. In fact the moderator between this extremes is *time*, i.e., the time to come to a solution which is still useful in time.

An interesting approach [Fischer95] is the understanding and building of interactive knowledge

media or collaborative problem solving environments rather then the traditional goal of understanding and building autonomous, intelligent thinking machines.

In collaborative problem solving systems, users and the system share the problem solving and decision making and different role distributions may be chosen depending on the user's goal, the user's knowledge and the task domain.

A collaborative system should address the point of what part of the responsibility has to be exercised by human beings, and how to organize things for an effective human communication with the computational system.

- a partial understanding and knowledge of complex task domains is acceptable;
- two agents can achieve more than one, especially by exploiting the asymmetry between agents;
- breakdown are not as detrimental, especially if the system provides resources for dealing with unexpected;
- semi-formal system architectures are appropriate, and
- humans enjoy "doing" and "deciding" by being involved in the process.

3. A Service-Oriented Architecture

The notion of agents presented up to here is associated to a notion of services and the building of service-oriented applications:

- *agents* are all kind of services used by an application;

  --> *available* services are offered by an agency;

  --> *non available* services are customized by the application at some site and after that handled as an available services;
- *agency* is a basic component able to offer services to an application;
- *negotiation* of services are handled by a *trader* [Trader95];
- *trader* is another service that is able to locate other services in a pool of contracted agencies.

3.1.Services

Computing with services is a higher level of abstraction in implementing any application reducing the development effort to the specific objects not available anywhere and to the interconnection of all the active objects regarding the application. The interconnection of these objects will deal with: *contracting*, *locating*, *requesting* and *replying*. The term *active objects* [Sichman95, Orfali96] is also equivalent to agents in a multi-agent environment.

Available services: can be of any kind, like remote: co-processors, databases, data crunching, archiving, etc.

Non-available services: needed by an application can be of any kind, like the above, but for some reasons it is just not available in the context. Non availability can have different meanings like:

- the application has non authorized access to a service;
- a specific service is not available where needed;
- a service is temporarily disconnected;
- the service is a too specific computation of the application and has to be customized.

The application has to handle this unavailability accordingly and customize the missing service. The customization of a service will handle with: code transportation, resource allocation for execution, naming and registering of the service. After customization the application can deal with the customized service just as it deals with any other already available service. Any service can make use of other remote services and for that there is an inter-service communication.

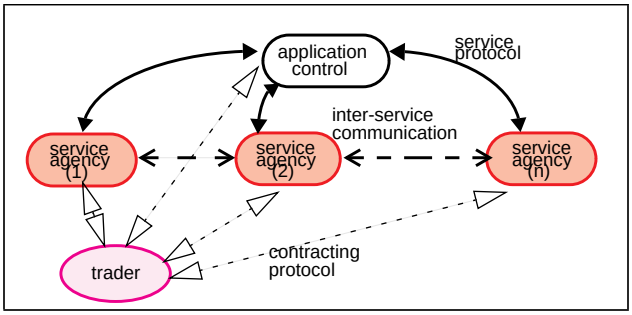

FIGURE 1. Service-Oriented Application.

3.2.Service-Oriented Agency

The agency architecture is composed of an object broker and a collection of agent services, which may include or not as services an *agent mobility service* and an *availability service*.

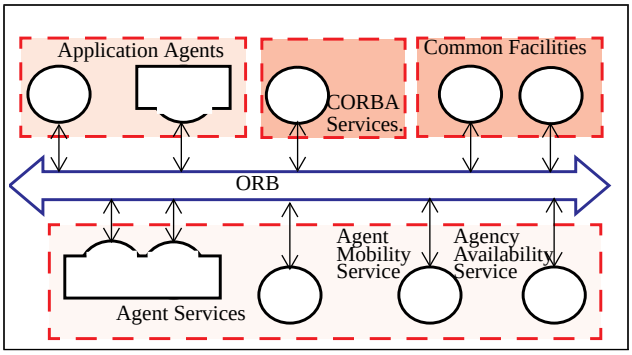

FIGURE 2. A CORBA based agency model.

An agency with agent mobility and availability services is able to run new services loaded by the application itself, i.e., the agency is open to new services or agents to be loaded by an application demanding this kind of service.

In straight relation to the CORBA model and its object services, a service offered by an agency can be called agent and the collection of services called agent services, as sketched in Figure 2.

Middleware: The Multiware platform [Loyolla94] sketched in Figure 3 is the platform on top of which this work is being done. CORBA allows for a good degree of flexibility in the implementation of the core ORB. It can be implemented as a set of runtime libraries, a set of daemon processes, a server machine, or part of an operating system [Orbix96]**.**

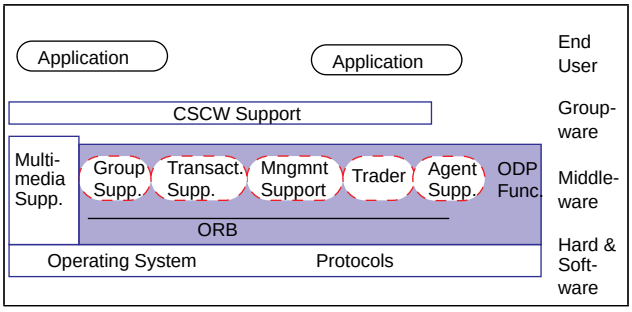

FIGURE 3. Multiware Platform.

3.3.Life-Cycle and Persistence

Up to here services can be identified in different phases during its life-cycle:

- start-up - stationary - migration - removal.

Start-up: involves contracting and distribution like it is considered for any application.

Stationary: phase of a service can be temporary or indefinite according to the characteristics of the service. Making services available for general usage involves management and distribution of this services in order to guaranty availability as much as possible. One can think of this services as stationary most of the time as long there is no major problem with the network or host on which this services are running. But thinking of services as *always available* demands a natural need to make smooth moves in case of some failure in the environment.

Migration: of a service is demanded by the environment or the service itself and usually in attendance to load balancing, inverse caching needs, or redistribution due to some failure in the environment.

Migration involves persistence of code and status, i.e., before moving the agent has to save the variables that define its status and persistently store them. Both, status and code, are moved as sequences and both are persistently stored at the receiving site, followed by a removal at the sending site after the move has been successfully completed. At the very moment when the agent is instantiated it reads back its status into the original variables.

In all situations after arrival the agent is instantiated by the agent support in order to recover from the status file its memory on what it has to do. If it has just to do nothing and go idle that is coded in the status.

Removal: follows shutdown or migration of a service. In this phase there is the possibility of using a migratory agent passed as a token in order to handle any application termination and proper shutdown. Again, this token agent is composed of code and data.

3.4.Mobility Service

Mobility service supports the reception of an agent, its persistent storage and the registration of its interface on the ORB. Basic setup and execution steps are as follows:

- at the sending / receiving end:
  --> ORB running;
  --> registering the mobility service;
  --> calling the mobility service;
  --> marshalling / un-marshalling & sending / receiving of agent;
  --> remove/store agent from persistent storage.

There are some additional steps for moving an agent:

- at the sender
  --> disable any new request by removing the interface registration;
  --> put the agent on a dispatch queue;
  --> the service (agent) continues running until it concludes the execution of any previous request already attended;
  --> is killed when it goes idle.
- at the receiver
  --> publication of agent interface;
  --> instantiation when receiving a request.

The persistence service is needed for code and status storage of moving agents.

3.5.Availability Service

When a new service is going to be setup at some site, there is the need to locate and allocate resources on an agency. In order to identify these agencies open to new services, another service is included in the agency itself: an *availability service* which informs the level of availability of the agency.

The availability service evaluates the loading of an agency using the performance metrics included in the instrumentation facility [Queiroz96]: *response time*, *throughput*, and *utilization*. Utilization allows different paramenters to evaluate loading in terms of: CPU, memory, disk, networking activity, number of users / processes. This numbers are computed including the *specmark* of the particular host in order to allow a comparative value to other hosts.

The availability level of the agency is published in order that this parameter can be obtained from a querying to the agency or via a *trader.*

Availability Evaluation: One can think of an evaluation process or daemon just being started when there is an availability request, however, availability has to consider a certain backtracking in time, reflecting the time the application will execute. Considering this approach availability evaluation demands a continuous running daemon on every host which puts its resources available. There is also an associated periodic logging of the host loading history.

Continuous running process doing logging are not always welcome because they demand CPU time and storage resources. However one can think of a dedicated firmware based co-processor(s) on general purpose hosts doing this kind of task while logging can be constrained to a maximum number of history samples, by removing intermediate samples from the history archive as time passes by.

Cheap dedicated firmware co-processors which can be remotely programmed, like Sun´s Java chips [Javachip96], can help a lot in distributed monitoring tasks and management applications.

3.6.Trading Service

In case of querying via a trader [Trader95, Lima95], the query includes a range of availability of a specific kind of resource. The trader replies returning a list or simply the most available agency. The selection phase can include a direct interrogation before contracting for the loading of the new service by the application.

An additional step at this point allows fine tuning, by using a customized agent to evaluate the agency more closely in case of a very sensible application. This can be added as a trading extended service at the trader side or at the application level itself.

4. Some Implementation Details

4.1.Orbix

The ORB consists of library functions linked with clients and servers together with the runtime system. This approach should provide improved throughput and better application distribution.

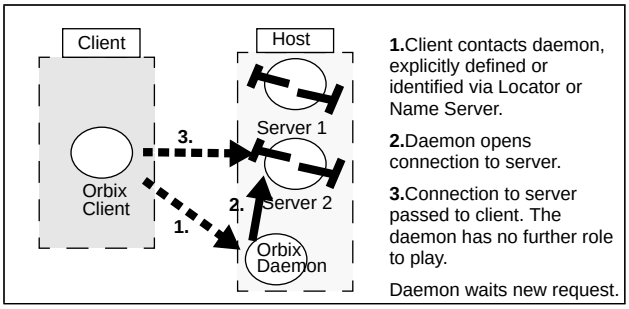

FIGURE 4. Client/Server connection w/orbix daemon.

The **server library** can both issue and receive remote object operation requests, while the **client library** can only initiate such requests. A **daemon** is responsible for launching server processes dynamically as required, in accordance with the various activation policies described in the CORBA Specification. Non-distributed client / server applications in the same address space, can be built using the server library alone. Figure 4 shows how Orbix handles client / server connections.

API: An important step in generating a distributed application is the definition of the interfaces by which clients and servers will communicate. IDL interfaces are formal standardized interfaces defining the provided services and how clients shall invoke them. Figure 5 shows interfaces on the ORB in inter-service communication.

IDL: compiler uses the IDL file to generate the client stubs and the server skeletons. If client and server are going to be on different platforms then the same IDL will be compiled for both platforms with appropriate client and server code.

Stub & Skeleton: Stubs can be used within the client application as local object method invocations, while on the Server side these methods, i.e., skeletons must be implemented to produce the desired result. Server application developer can work completely independent of the client developer.

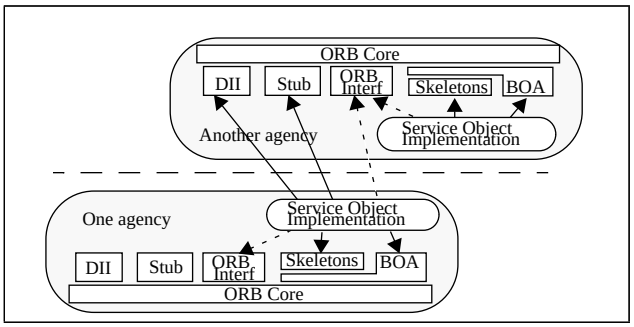

FIGURE 5. CORBA Server / Server communication.

Dynamic Invocation: A dynamic method invocation by a client establishes communication between client and server transparently initiated by the daemon. If the host supports the particular server, the daemon will establish a connection with the server, pass it back to the host and drop out.

In all cases the client / server interoperability is established while location and nature of the server host is kept completely transparent. The daemon is only concerned with the appropriate servers to be running and client / server connections, it is not an ORB itself.

BOA (Basic Object Adaptor): On Orbix, BOA functions are: publishing the interface of an object in the implementation repository (putit), remove it (rmit), list all registered interfaces (lsit), detailed listing of an interface (catit), show an active object status (psit), kill an active object (killit).

The **locator** service is used for transparent inter-agency service location. On Orbix it is possible to include a set of interfaces in a group and also create a group of hosts.

4.2.Passing code messages

There is currently an OMG's request for proposal on *passing object by value*. The subject of passing objects [Lange95] is handled here on a Orbix platform passing code as a message in sequence type parameter format.The following results are still based on Orbix 1.3.

The object to be sent is read from the persistent implementation repository at the sending site as a binary sequence into an IDL sequence type parameter. The sequence is sent as a normal parameter to the receiving site and persistently stored in its local implementation repository.

The implementation repository is based on a file system, and this imposes that every agent has to be on a separate file, specially if this agent is going to be moved somewhere. A better approach would be the usage of an object-oriented database where status and code could be persistently stored specially when the number of agents starts to increase a lot. This is directly related with *persistence service*, but is not the subject of this work.

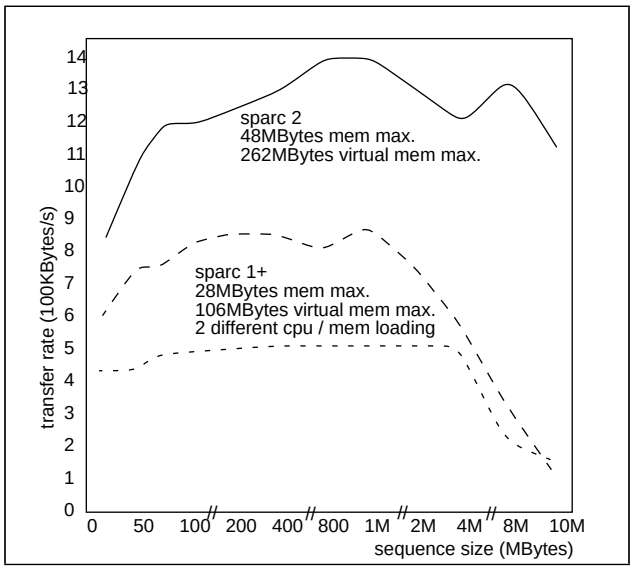

FIGURE 6. IDL-sequence passing over ethernet in a non dedicated environment with client/server running locally.

Messages of ascii type and binary type up to the size of 1MByte are considered for most applications and being passed without major problem.

In order to estimate an upper bound on the message size and a lower bound on the transfer rate, the performance was measured on the less performing machines in the considered network. The plots presented in Figure 6 and 7 are based on sparc 1+ and sparc 2 machines non-dedicated to this purpose over ethernet running and sequence sizes from 10KB to 10MB. In Figure 6 the transfer rate drops quickly as the message size goes above 2 MB due to memory allocation and because client and server processes run on the same machine reaching higher rates then in Figure 7 where client and server run on different machines.

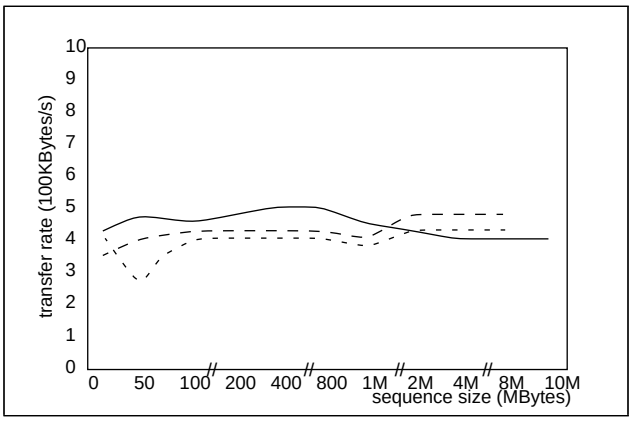

FIGURE 7. Same conditions as before but with client/ server on different hosts.

Actually the memory allocated to pass the sequence imposes a practical limit to message size, but a compromise can be obtained by spliting the full message into packets reflecting a predefined buffer size. There is a reference [Schmidt95] where image files for multimedia applications up to 64 MBytes are passed in packets of 124 KBytes using an additional external high speed channel.

4.3.Application start-up

The steps regarding start-up of an application under the described environment is resolved by the same mechanism described for the contracting of available services and the loading of new services, i.e., existing services are contracted and unavailable services are distributed to selected agencies departing from an implementation repository site.

This has nothing to do with OMG's Object Start-up Service [OSS95] which deals with the start-up of the ORBs.

4.4.Implementation Repository

There are different possibilities for the distribution of unavailable services regarding their location at start-up.

In a simple case, the implementation repository of the application is hold locally at the user's machine

and the new services are redistributed according to the application's demand.

At the other extreme, a more sensible application where more reliability is needed a implementation repository server can be added to the context in order to hold a copy of all services that are to be distributed in this context. With such a server, start-up can always start redistribution from this repository server. The same server is in the last position of every list of possible implementation repositories used by any application in this context. This server guaranties that an implementation repository copy always exists, even in case of failing hosts.

The possibility of failure of this main repository server exists and it can be treated using the same approach, i.e., adding another server to increase redundancy and so on up to level of acceptable cost / reliability ratio.

4.5.Unavailable services distribution

Independent of the kind of implementation repository, the new services are redistributed according to the application's demand on load balancing and / or inverse caching. This means that if distribution is not demanded by the application in the current environment circumstances, then it runs just locally. Another possibility is that the application has to wait in the start-up queue for sufficient resources.

4.6.Results on application distribution

An application start-up result is presented next, where the two already mentioned situations demanding migration of objects are considered, i.e., load balancing and inverse caching.

Inverse caching: conditions are considered as follows:

- an object needs to move to a specific site in order to run attached to a contracted service;
- the moving object has to contract the local agent support and demand migration to where the specific service is;
- the agent support locates the specific service and sends the agent calling the remote agent support as a inherited class / method.

Load-balancing: conditions are similar to the previous ones with additional considerations:

- an appropriate agency has to be contracted under the availability service query criteria;
- availability services are identified in a unique way by concatenation with the host name and invocation using multicast;
- the selected availability service name is passed to the local agent support which locates the specific service and sends the agent whose migration was requested.

Evaluation: of the an application was made were a stationary client shares the execution of the problem with a moving server. The client searches for enough memory and cpu for the moving server to execute. For that the client contacts every host in the allowed group, including itself, for availability. The status of the execution is kept by the client while the server when restarted on another host is always in its initial state.

In order to decouple the test from the previous test of code passing performance in section 4.2, the actual code of the server is made available on every host so that only the execution is passed to the server on the next selected host. The transparent distribution of the application allows exactly the same application to be started several times running concurrently with every other new one started.

4.7.Updating and Debugging of an Application

Once a application system is released and running, it can be very cumbersome to bring it down for an update and subsequent start-up and associated debugging.

In an application build with agents each agent can be shutdown and restarted separately without a shutdown / start-up of the whole application. The updating and debugging of a full application can be simplified to the updating of an agent.

Some additional considerations can be done about updating when resulting in replication, spliting or inclusion of a new agent into the context. This may demand the update of the interface of some agent or

group of agents. In case of merging, the resulting agent may inherit all the interfaces of the group of merging agents.

4.8.Management Aspects

Agents: to be moved have to regard interoperability. In this paper some testing of performance is done in a homogeneous environment and migrating agents could be implemented in a common C++. An important ability is to be able to upgrade platforms and software over the life-cycle of a process in a more flexible manner, replacing components over time and / or a complete software update. The similarities between Java and C++ makes it atractive for thie work with Java introducing some interesting features [OSF96]. The performance issue of running interpreted as opposed to compiled code is not necessarily a major constraint while the basic requirement is that any new remote component must be able to have a Java runtime like.

Problem solving strategies: can be more then one and so different strategies could be allowed in different agents, with the most effective giving the earliest response.

Management: can regard the application or the agencies. Management of applications stands for task management but the management of the agencies is outside the focus of the application which may not have authority for that. For the application, if a service is not available it should get another one from a trading service and / or decide how to proceed. Management of agencies consists of managing a collection of agencies seen from the service provider side and stands to distributed system management. Anyhow services which interact with other remote services is an application itself using task management.

Levels: are proposed for a step wise allowance of sensing and reacting, with the history of the system being used for every new hint issued. According to section 2.3 a learning phase about the environment is needed followed by an execution phase and a successive repetition of both will allow the system to be kept up-to-date.

Some fixed services for management were implemented [Queiroz96] and next services will regard the advantages introduced by mobility

5. Conclusion

The proposed architecture introduces an *availability service* as a way to combine the use of fixed services and moving services in a environment where available services are contracted and unavailable services are customized and then contracted. This is clearly illustrated by an application start-up where transparent code distribution and later redistribution due to a specific demand is possible. Every machine open for new applications in a certain context has to have the availability service built as a fixed service. The usage of the trader and the mobility service are associated and particularly important in heterogeneous environments.

Distributed Client / Server paradigm can be extended with the usage of code migration. Migration or mobility makes it possible to reduce network traffic, to optimize load balancing, and to provide better response time to the user. Mobility can be considered as an essential component of future open systems improving software distribution at large, specially if the technology runs across multiple platforms. Additional requirements are introduced, particularly on security and interoperability that make it hard to run with the same performance of native code. But, after all, native code is not mobile.

Some scientific applications in large experimental areas need to run continuously during long periods but also stand dynamic reconfigurations due to continuous upgrading of the experimental hardware/software and also due to the search for special events. For management of applications with the characteristics above it is quite important to have the ability to upgrade platforms and software over the life-cycle of a process in a more flexible manner, replacing components over time and / or a complete software update.

Acknowledgments: The present work has support from: FAPESP, CAPES and CNPq.

6. References

[Cardozo93] . Cardozo, E., Sichman, J. S., Demazeau, Y., *Using the Active Object Model to Implement Multi-Agents Systems*, Proc.of the 5th IEEE Conf.on Tools with Artificial Intelligence, Boston, USA, 93.

[CFA95]. OMG, *Common Facilities Architecture, Revision 4.0*, OMG Document # 95-1-2, 3 January 95.

[CORBA9x]. OMG, *The Common Object Request Broker: Architecture and Specification, rev 2.0, july 1995.*

[Fischer95]. Fischer, G., *Rethinking and Reinventing Artificial Intelligence from the Perspective of Human-Centered Computational Artifacts,* Lecture Notes in Artificial Intelligence, #991 Springer, October, 95, pp 1-11.

[Franklin96]. Franklin,S., Graesser,A., I*s it an Agent, or just a Program?: A Taxonomy for Autonomous Agents,* Proc.of the 3rd International Workshop on Agent Theories, Architectures, and Languages, Springer, 96, http://www.msci.memphis.edu/~franklin/AgentProg.html.

[Goldszmidt96]. Goldszmidt, G. S., *Distributed Management by Delegation,* PhD Thesis, Graduate School of Arts and Sciences, Columbia University, 96.

[Java96]. OSF, *Java Mobile Code Paper*, January 15, 96. http://www.gr.osf.org/projects/web/java/whit_pap.htm.

[Javachip96]. Sun Microsystems Inc., SunnyVale, CA, USA February 2, 96, http://www.sun.com/sparc/newsreleases/nr95-042.html.

[Iglesias95]. C.Iglesias, J.C.Gonzalez, J.R.Velasco, *MIX: A General Purpose Multiagent Architecture*, Lecture Notes in Artificial Intelligence, #1037 Springer 96, pp 251-266.

[Lange95]. Lange,D.B., Chang,D.T., *Aglets Workbench*, IBM Corporation, August, 96, http://www.ibm.co.jp/trl/aglets.

[Lima95]. Lima Jr., L.A.P., Madeira, E.R.M., *A Model for a Federative Trader*, Open Distributed Processing: Experiences with Distributed Environment, pp.173-184, Chapman&Hall, 95.

[Loyolla94]. Loyolla, W.P.C., Madeira, E.R.M. Mendes, M.J; Cardozo,E., Magalhães,M.F., *Multiware Platform: An Open Distributed Environment for Multimedia Cooperative Applications*, IEEE Computer Software & Applications Conference, COMPSAC'94, Taipei, Taiwan, November 1994.

[MAGMA95]. MAGMA:http://cosmos.imag.fr/MAGMA/magma_research.html ,1st Feb., 95.

[Queiroz96]. Queiroz,A., Madeira,E., *Management of CORBA objects monitoring for the Multiware platform*, private communications.

[ObjInt1]. Schmidt,D.C., Vinoski,S., *Object Interconnections: Introduction to Distributed Object Computing (Column1)*, C++ Report Magazine, January 95.

[ObjInt2]. Schmidt,D.C., Vinoski,S., *Object Interconnections: Modeling Distributed Object Applications (Column2)*, SIGS C++ Report Magazine, February 95.

[ObjInt3]. Schmidt,D.C., Vinoski,S., Object Interconnections: *Comparing Alternative Client-side Distributed Programming Techniques (Column3)*, SIGS C++ Report Magazine, May 95.

http://siesta.cs.wustl.edu/~schmidt/publications.html.

[ObjInt4]. Schmidt,D.C., Vinoski,S., Object Interconnections: *Comparing Alternative Server Programming Techniques (Column4)*, SIGS C++ Report Magazine, October 95.

http://siesta.cs.wustl.edu/~schmidt/publications.html.

[ObjInt5]. Schmidt,D.C., Vinoski,S., Object Interconnections: *Comparing Alternative Programming Techniques for Multi-threaded Servers (Column5-6-7)*, SIGS C++ Report Magazine, February-April-July/August 96. http://siesta.cs.wustl.edu/~schmidt/publications.html.

[ObjInt8]. Schmidt,D.C., Vinoski,S., Object Interconnections: *Distributed Callbacks and Decoupled Communication in CORBA (Column8)*, SIGS C++ Report Magazine, October 96, http://siesta.cs.wustl.edu/~schmidt/publications.html.

[Orbix96]. IONA Technologies, Ltd., *OrbixTalk: Management Overview*, April 96, http://www.iona.com/Orbix/Talk/MO/CorbaIntro.html.

[OSS95]. OMG, *Object Startup Service*, IBM submission, OMG TC Document 95.10.7.

[Russel95]. Russel, S., Norvig, P., *Artificial Intelligence, A Modern Approach*, Prentice Hall Series in Artificial Intelligence, New Jersey, USA, 1995, page 33.

[Sichman95]. Sichman, J. S., Demazeau, Y., *Exploiting Social Reasoning to Enhance Adaptation in Open Multi-Agent Systems*, Lecture Notes in Artificial Intelligence, #991 Springer, October, 95, pp 253-263.

[Schmidt95]. Schmidt,D.C., Harrison,T., Al-Shaer,E., *Object Oriented Components for High-speed Network Programming*, Proc.of the USENIX Conf. on Object-Oriented Technologies, Monterey, CA, June 95, http://siesta.cs.wustl.edu/~schmidt/publications.html.

[Trader95]. ODP, *Trading Functions*, ISO/IEC JTC1/SC 21, 20 June, 95, ftp://ftp.dstc.edu.au/pub/arch/RM-ODP.

[Vinoski93]. Vinoski, S., *Distributed Object Computing With CORBA*, C++ Report Magazine, july/august 93.