SPIN: Scientific Programing on the InterNet

M. González[1,2]
e-mail: mgonzale@asci.fr
http://www.erablis.com/martha
and
S. Petiton[2,3]
e-mail: petiton@lifl.fr, petiton@asci.fr
http://www.lifl.fr/~petiton

[1] Escuela de Computación. Universidad Central de Venezuela. Caracas - Venezuela
[2] Laboratoire d'Applications Scientifiques du Calcul Intensif (ASCI) CNRS-CPR 9029. Université Paris-Sud. Orsay - France
[3] Laboratoire d'Informatique Fondamentale de Lille (LIFL). Université des Sciences et Technologies de Lille. Lille - France

Abstract. *The aim of this paper is to present the general architecture of SPIN, an interactive and integrated environment for distributed scientific computing. SPIN's main goal is to allow the access to remote computational resources in an easy to use and transparent way from the user's point of view. This environment, based on a client-server architecture, is being developed following a software engineering process which incorporates the CORBA standard for interoperability and integration; the PAC-Amodeus model, a multi-agent based model for interactive systems architecture and the OOSE: Object-Oriented Software Engineering approach, based on the use cases notion, to build the different models of the application domain component.*

1 Introduction

The growing needs of more reusable, portable and efficient software applications in several domains have motivated many researches on hardware and software technology, one of the main goals is to provide tools that can operate across diverse platforms, and that can interoperate with legacy systems.

The main mechanism to facilitate the portability of existing codes is to use libraries which define a set of stable interfaces. Among these libraries we can mention: *BLAS: Basic Linear Algebra Subprograms*, a set of basic codes to perform operations on vectors and matrices [4], *PBLAS* a parallel version of the BLAS for distributed memory computers [9], *Sparskit* which focuses on sparse matrices computations [20]. There are also some developments trying to incorporate the Object-Oriented approach: *LAPACK++: Linear Algebra Package* [10], a C++ extension of the LAPACK library for solving common problems of numerical linear algebra: linear systems, linear least squares, and eigenvalue problems on high-performance computer architectures. *PETSc: Portable, Extensible Toolkit for Scientific computation* [3], a suite of data structures and routines that provide the building blocks for the implementation of large-scale application codes on parallel (and serial) computers [3]. *POOMA* a C++ class library to provide a flexible environment for data-parallel programing [2].

Nevertheless these libraries can be inexistent on some platforms and their use can in some cases demand considerable programming efforts to their users, who may not have the time to learn the programming details of a given library or language. Moreover, the use of scalable parallel architecture conduces to many program adaptations and their related problems.

On the other hand, tools like MATLAB [12] which tries to deal with such difficulties are available on a limited number of platforms.

The considerations of these problems have motivated the development of a scalable, integrated and interactive environment for distributed scientific computing on high-performance computers.

Our environment provides the following advantages:

1. Platform independence. Operations call will be independent of the library and programming languages implementing the computing capabilities of the servers.
2. Easier computations parallelization. The program source code will not depend upon the number of available processors to execute it.
3. Flexible parameterization. The user can configure many parameters for optimization purposes. For instance the user can choose the best server to execute an operation thanks to the possibility in obtaining the computational ressources of each server from a list of available servers.
4. Incremental capacity growth. In any time it is possible to connect and add new servers with new computational capabilities which will permit to enhance the execution of existing programs without need to operate any change on the existing code.
5. Portability and interoperability by the use of the Java and CORBA/IIOP technologies. This choice will permit us to adapt our system easily on many heteregeneous platforms (super-computers, workstations, etc.).
6. Preserves technology investments by means of standards based technologies, software components, sustainable middleware.
7. Leverage investments in legacy code developing software components, wrapping legacy code. As some of libraries available on certain platforms are highly optimized, our objective has been to permit their use each time it would be possible.
8. Web integration, allowing the use of our system interface from everywhere.

The environment has been designed and developed using different object-oriented technologies, among them we can mention:

– *CORBA: Common Object Request Broker Architecture* a standard defined by the *OMG (Object Management Group)* for interoperability and portability for distributed object-oriented applications [19], [18], [17].
– *PAC-Amodeus model* based on the *multi-agent* notion, it is used to define the architecture of the interactive system [7] [8].
– *OOSE (Object-Oriented Software Engineering)* approach to model the application domain functionalities [14].

- *UML (Unified Modeling Language)* for the domain component model specification [21], [22].
- $Java^{TM}$ programming language, to develop the user interface, we take advantage of its facilities to define, implement and access CORBA objects [5], [16], [23], and also of its faclilities to operate with libraries written in other languages.
- $JDBC^{TM}$ *API* defines $Java^{TM}$ classes to represent database connections, SQL statements, result sets, database metadata, etc. It is used to store the users information (default parameters, source code, etc.).

Besides this introduction and the conclusions, this paper contains the following sections: section 2 describes the general architecture of the environment, the section 3 describes the client-side architecture and the section 4 is a brief description of the server-side architecture.

2 General Architecture

For the general architecture definition we have chosen CORBA (Common Object Request Broker), a set of specifications published by the OMG (Object Management Group). These specifications define the way objects are defined, created, dispatched, invoked and how they communicate with each other.

The figure 1 presents the general architecture of the SPIN environment, based on the CORBA standard.

The ORB (Object Request Broker) establishes the client-server relationships among objects. Using the ORB, a client can transparently invoke a method on a server object, which can be on the same machine or not. The ORB intercepts the call and is responsible for finding the refered object, pass the method's parameters, invoke the method and return the results. The client does not have to be aware of where the object is located, its programming language, its operating system, or any other system aspects that are not part of an object's interface. In doing so, the ORB provides interoperability between applications on a same or different machines in a heterogeneous distributed environment thanks to the IIOP (Internet Inter-ORB Protocol) protocol.

The ORB used for our implementation is provided by Java IDL. Java IDL is an ORB provided with the JDK 1.2. Together with the idltojava compiler it can be used to define, implement, and access CORBA objects from the Java programming language. Java IDL is compliant with the CORBA/IIOP 2.0 specification and the IDL-to-Java Language Mapping [16], [19], [24].

The client side provides a graphical user interface. This interface wrote in Java can be executed on several platforms (generally a personal computer or a workstation which possesses an JVM). The client side does not need any computing power as it only acts as a controller of supercomputing servers which provide high performances (computing and mass storage).

The client application can be seen as a centralized instruction controller of an MIMD machine whose processors are super computers loosely connected (we think in using basically the Internet World Wide network). This vision will be used in the near future to provide to our system adapted strategies for load balancing.

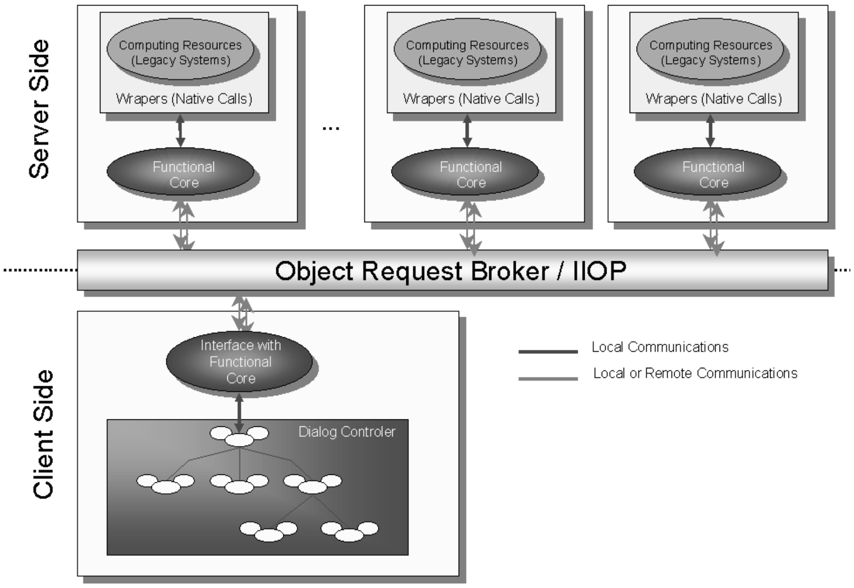

Fig. 1. SPIN's General Architecture

In the server side computing resources are located. These resources are specific applications for scientific computing, the legacy systems and their corresponding object wrappers. The components of this side conforms the Functional Core (FC) corresponding to the domain specific component. For the analysis and design of this component, an object-oriented method based on the uses cases approach has been used [14]. The resulting models have been specified using the UML notation [21].

The figure 2 presents how the distributed application operates in the defined architecture.

Our system makes use of the HTTP protocol to permit the user to download the client application using a $Java^{TM}$ enabled web browser and to retrieve from a database the information needed to deal with the computing servers. This information basically includes:

1. The list of servers, the user will be able to connect to.
2. The principal characteristics of the servers (available resources such as processors, memories, libraries).
3. The available data and methods available on each server.
4. The source code of the user's programs.

As the client application is running, the database is updated accordingly to the manipulations realized by the user (new programs, modification of the server list, etc.). As the user executes one of its programs on a list of servers, the IFC translates each basic instructions in an invocation stream and send it to the servers via the the ORB, thanks

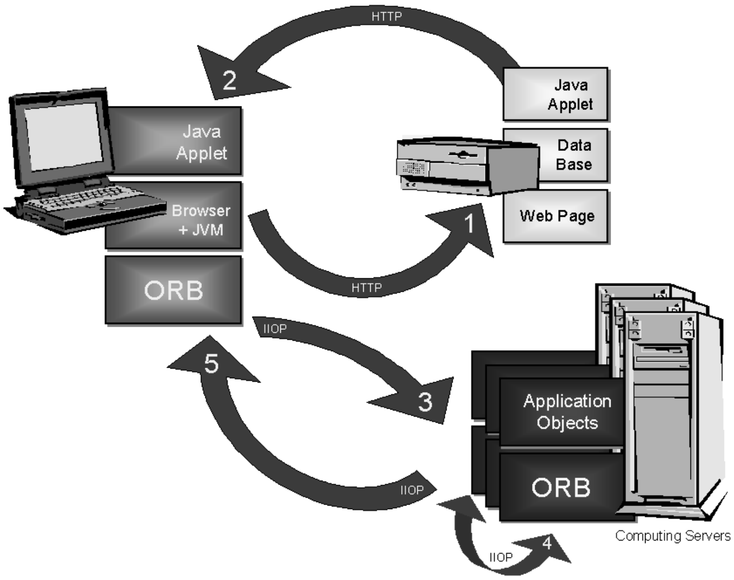

Fig. 2. Client Application Operation

to the IIOP protocol. These invocations are treated by the servers and the execution status returned to the client application. The IFC's (Interface with the Functional Core) functionalities will be explained later.

Based on the previously described approach, a typical scenario in using our system is shown by the figure 2:

1. From a Java enabled Web browser, the user enters an URL that points to a logon page on the Web server.
2. The Web server receives the user request and returns a HTML (HyperText Markup Language) page to the client browser. The Web page makes reference to the Java applet which is loaded and executed by the browser's JVM, making up the client component of the distributed application. Thanks to the user authentication, the associated database is retrieved from the Web server.
3. As working (program writing, etc.), the user's database on the web server is updated. The user write programs by means of an easy to use graphic interface based on object manipulations. The user constructs its program managing its data (vectors, matrices, etc.) like objects on which it applies a set of methods (product, sum, etc.). These methods will be translated at the execution time in remote invocations to the servers by the IFC (Interface with the Functional Core) which will map them on functions of the specific library available on the server (legacy code). The scheduling of the invocations to each server is realized thanks to the information provided by the user indicating the server on which a particular instruction must be

executed. In future works the assignment of a treatment to a particular server will be managed automatically by the IFC.
4. As computations required by a server can involve data stored on another one, the communications to retrieve the data are made via IIOP.
5. The server sends the results to the client application.

3 Client Architecture

The client-side architecture has been developed using the *PAC-Amodeus model*. PAC-Amodeus is an hybrid multiagent model that reuses the main components advocated by Arch/Slinky: Functional Core, Interface with the Functional Core, Dialogue Controller, Presentation Techniques and Low Level Interaction [8]. Besides that, PAC-Amodeus refines the Dialog Controller in terms of PAC agents [7]. In the following we describe the components belonging to the client side:

– The Interface with the Functional Core (IFC): serves as a domain adaptor between the Functional Core and the Dialog Controller, because of the use of different formalisms. The IFC translates the requirements captured by the Dialogue Controller in a form that is understand by the Functional Core. The communications between both components is made via the ORB.
– The Dialog Controller: has the responsibility of task sequencing. Each task correspond to a user's goal. PAC-Amodeus structures the Dialog Controller as a hierarchy of specialized computational units called agents. An agent has a state, possesses an expertise, and is capable of initiating and reacting to events [6]. In our environment this level is composed by the agents responsible of the interaction with the user. Each of these agents have different but complementary facets, which define their external aspect, their internal structure and the bridge between external and internal aspects. These facets are: Presentation, Abstraction and Control[7].
  • *Presentation*: is the perceivable behavior.
  • *Abstraction*: implements the competence of the agent in a media independent way.
  • *Control*: links the abstraction to the presentation, controls the behavior of the two facets. It serves and maintains relationships with other agents in the hierarchy.
– The Presentation Component: serves as an adaptor between the Dialog Controller and the Low Level Interaction Techniques. It translates the abstract presentation objects used by the Dialog Controller in terms of interaction objects which are basic elements platform-dependents.
– The Low Level Interaction: denotes the underlying platform, includes basic interaction facilities such as windowing systems, a spoken language shell and toolkits. It manages the physical interaction with the user. In our case this level corresponds to the facilities provided by Java Swing [1], [13].

The PAC-Amodeus model is characterized by a strong modular organization, communications by events and the possibility for concurrent or parallel processing. The events can occur simultaneously and can involve different agents, who can process them in parallel. This model encourage the practice of multi-threaded dialogs where each agent is specialized in the management of a thread.

We have chosen the PAC-Amodeus model for the following reasons:

- The effective separation of concerns, is an advantage with respect to MVC model, where the functionalities of the agents are dispersed among the different perspectives [15], [7].
- it explains explicitly how to realize the communications among agents.
- the control notion facilitates the communications.

The figure 3 represents the SPIN user interface with a scheme of a program in construction. Each presentation element such as menu, tool bar and windows are modeled as PAC agents.

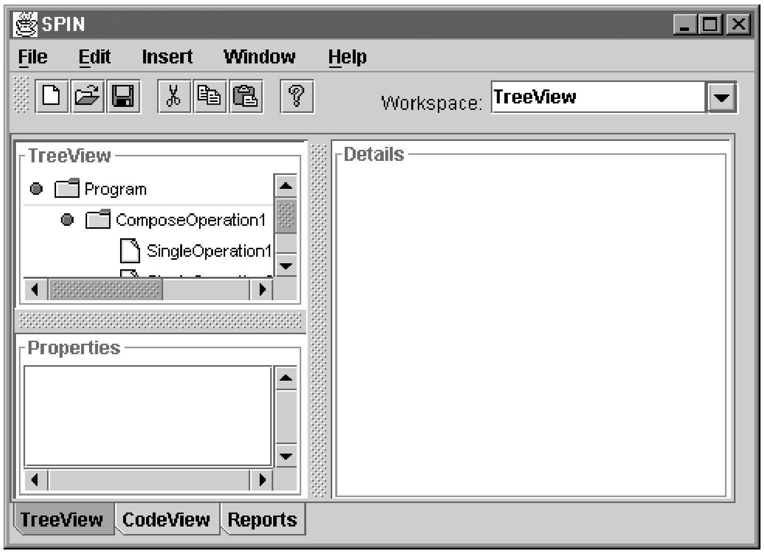

Fig. 3. The main SPIN User Interface: The current window contains three frames to represent the different aspects of a program; its tree view, the properties of each node and the details.

The figure 4 presents the PAC agents of the user interface shown by the figure 3. Each agent is represented by a three-facet oval corresponding to Presentation, Control and Abstraction perspectives.

4 Server-Side Architecture

In this side we have located the Functional Core (FC) of the distributed application, the legacy systems and their corresponding object wrappers.