STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **XLV**, Number 1, 2000

A JAVA-BASED OBJECT-ORIENTED INFRASTRUCTURE FOR HPCC

MARIN IUGA AND BAZIL PÂRV

Abstract. This paper present a Java-based object-oriented infrastructure for an High Performance Computing Center (HPCC). This infrastructure has several functional levels: user- and server-interaction (at client level), and identificating and getting all the relevant information (at communication protocol level). The main functionality of the server is to establish the link between the algorithms requested by the client and their storage environment, by offering additional assistance to clients while browsing through the algorithms collection.

Keywords: High performance computing, Java technologies, client-server architecture, object-oriented infrastructure.

1. The general structure of HPCC

This work is based especially on [2], trying to concretize the abstract specification of a High Performance Computing Center (HPCC) given there. It continues other works on the same topic (see [1] and [3]).

The HPCC application has several functional levels. Figure 1 below presents the way these levels are structured, taking into account the functional needs for data manipulation, and the functional dependencies between them.

As we see in Figure 1, there are five significant functional levels, each level using extensively services exposed by the previous ones. On its turn, each functional level has several sections, each with its specific services.

The first level (starting from top to bottom), **AD user level**, is user interface one; it allows the user to navigate, visualize, or search data contained in the algorithm store. Usually, this level will be an applet running on client machine. This applet will communicate with the data server either using a specific network protocol, or RMI. At this moment, there is no final decision concerning this issue. The main task of this applet is to capture user's needs and to generate queries for the data server. On its turn this server will process these queries by using the

2000 *Mathematics Subject Classification.* 68N19.

1998 *CR Categories and Descriptors.* D.2.2 [**Software**]: Software Engineering – *Design Tools and Techniques.*

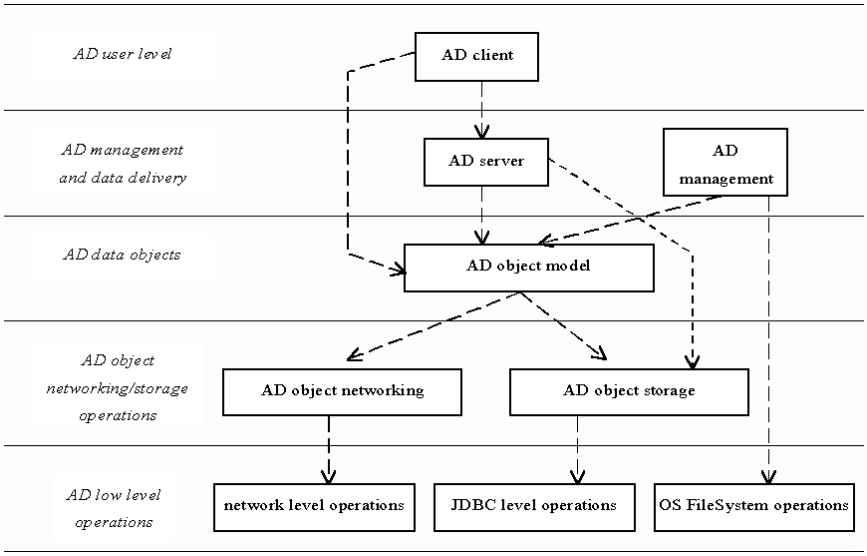

FIGURE 1. HPCC - Functional levels

services offered by the object model. Also, the client level will use the services exposed by the AD data objects level in order to manipulate those objects.

The next level, **AD management and data delivery**, has two different sections: AD server and AD management. The second section, **AD management**, is designed as a separate JAVA application. By using the services exposed by AD data objects level, its functionality covers maintenance of data about algorithms and classes of algorithms.

The first section, **AD server**, deals with data transmission to client applet. If a specific network protocol is used for client communication, AD server needs to be a daemon JAVA application running on server machine. In this case, the protocol for data transmission needs to be defined and implemented. In the second case, which uses RMI for data transmission, there will be a set of interfaces for ensuring communication between client and server. In fact, AD server will be a collection of such interfaces and some additional classes used to implement queries for algorithm store. This second approach for the AD server has the advantage of a simpler implementation, and the drawback of working with JAVA clients only.

The third level, **AD data objects**, is the core part of all applications which constitute HPCC. This level define the structure of objects which manipulate data referring to algorithms and groups of algorithms and implements a series of useful operations on them. These issues are discussed in detail in Section 2 and 3.

The fourth level, **AD object networking/storage operations**, is responsible with storing and transmitting the objects across the network. It has two sections: AD networking and AD object storage. The upper level will use the services of **AD object storage** in order to store/retrieve objects, as we discuss in the third section. **AD networking** exposes services for packing-transmission across the network-unpacking operations.

The basic level of the HPCC application, **AD low level operations**, defines some primitive operations. Functionality of this level has to be fulfilled by using some standard JAVA packages, including JDBC, and the usual functions of the operating system.

Note the pyramidal structure of the application, in which each level is using extensively only the operations exposed by the level below. This structure was designed keeping in mind the functional decomposition of the task and using a stepwise approach for abstractions.

2. Algorithm Store data scheme

Data about algorithms and groups of algorithms are modeled via two objectual counterparts, which use the relational paradigm for assuring their persistence. There are two levels of storing information:

- identification: map each algorithm/group to its specific main folder; this information is kept in two relational tables;
- data: data are stored in standardized structure of different files and folders in the main folder (description, source code Pascal and C++, JAVA applets).

The information contained in **Algorithm Store** is structured in two organizational levels: the algorithm level, **AlgorithmTable**, and the algorithm group level, **GroupTable**. Also, there are some internal tables.

All the tables are managed by the **AD object storage** section from **AD object networking/storage operations level**, which use **JDBC** and **OS FileSystem** operations, located on lowest level.

This section discusses the data scheme, while the next section is discussing the corresponding objects.

2.1. **AlgorithmTable.** Algorithm Store contains data referring to algorithms and groups of algorithms. Each algorithm is characterized by the following attributes: a name, a description file containing its goal, its parameters, and, in the case of functions, the result type. Other attributes include algorithm implementation, using a common programming language (C++, Pascal), and/or the corresponding applet, which is executed on client machine (see Table 1).

As we see in Table 1, the information contained in each line is a kind of directory information. The way this information is used to store all the data referring to an algorithm is as follows:

TABLE 1. The structure of **AlgorithmTable**

Attribute	**Description**
AlgorithmIndex	integer representing the algorithm id in the algorithm table
GroupIndex	integer representing the group id in the GroupTable (id of the class the algorithm belongs)
AlgorithmName	string representing algorithm name
HasDescription	boolean value: True if the algorithm has a description file and False otherwise
HasPascalCode	boolean value: True if the algorithm has a Pascal implementation and False otherwise
HasCPPCode	boolean value: True if the algorithm has a C++ implementation and False otherwise
HasApplet	boolean value: True if the algorithm has a corresponding JAVA applet and False otherwise
KeyWords	string, a list of keywords, separated by commas; the keywords are used in queries

- all the information referring to an algorithm is stored in a folder, called *algorithm main folder* and named `00-AlgorithmIndex`; its sub- folders are discussed below
- `description`: contains the file `description.html`, containing the algorithm description (this subfolder exists only if `HasDescription = True`)
- `pascal`: contains the file `pascal.html` containing the pascal source code (this subfolder exists only if `HasPascalCode = True`)
- `cpp`: contains the file `cpp.html` containing the C++ source code (this subfolder exists only if `HasCPPCode = True`)
- `applet`: contains the start file `applet.html` and all the necessary JAVA files for this applet (this subfolder exists only if `HasApplet = True`).

Note that the fields `HasDescription`, `HasPascalCode`, `HasCPPCode`, and `HasApplet` are redundant in the **AlgorithmTable**, because one can test the existing subfolders in the algorithm main folder. The reason is increasing the speed of the applications which use this table.

2.2. **GroupTable.** Each algorithm belongs to a unique algorithm group (e.g. sorting algorithms, searching algorithms, string pattern-matching algorithms, numerical analysis algorithms and so on). On its turn, a group of algorithms can be divided into subgroups in a tree fashion. The usual attributes for algorithm group are its name and a description file which contains the common features of its algorithms.

The structure of **GroupTable**, which contains data referring to algorithm grouping is detalied in Table 2.

TABLE 2. The structure of **GroupTable**

Attribute	**Description**
GroupIndex	integer representing the group id in the group table. Root group has the index 1
UpperGroupIndex	integer representing the id of the parent group in the group table. For the root group this index equals 0
GroupName	string representing group name
HasDescription	boolean value: True if the group has a description file and False otherwise (usually this flag is True)
KeyWords	string, a list of keywords, separated by commas; keywords are using in queries

All the information referring to a group of algorithms is stored in a folder, called *group main folder* and named `00-GroupIndex`; it contains the file `description.html`, i.e. the group description (this file exists only if `HasDescription = True`).

2.3. **Internal tables.** Algorithm Store also contains several internal tables, designed for a better implementation of its functionality. By using these tables, Algorithm Store server builds several URLs and then sends them to the client applet. On its turn, the client applet displays these URLs in the browser window. Thes internal tables are:

UnusedAlgorithmIds: unused algorithm ids (due to algorithm delete operations)

UnusedGroupIds: unused group ids (due to group delete operations)

GlobalData: contains context information: the root path for the file and folder structure and the prefix used for building URLs.

3. Algorithm Store Object Model

Both data and operations concerning algorithms and groups are modeled using objects. Both algorithms and groups are considered objects, which are manipulated by using a specific manager object. The designed classes are:

AlgorithmInfo: models the algorithm object

GroupInfo: models the algorithm group object

ObjectDBManager: models the object manager, which performs load/store operations on objects. Because all objects are stored in a relational database, store and load operations need some specific transformations (i.e. linearization).

The object model also contains some support classes, needed for object propagation across network. These classes are not full implemented. `AlgorithmInfo` and `GroupInfo` classes belong to **AD object model** level, while `ObjectDBManager`

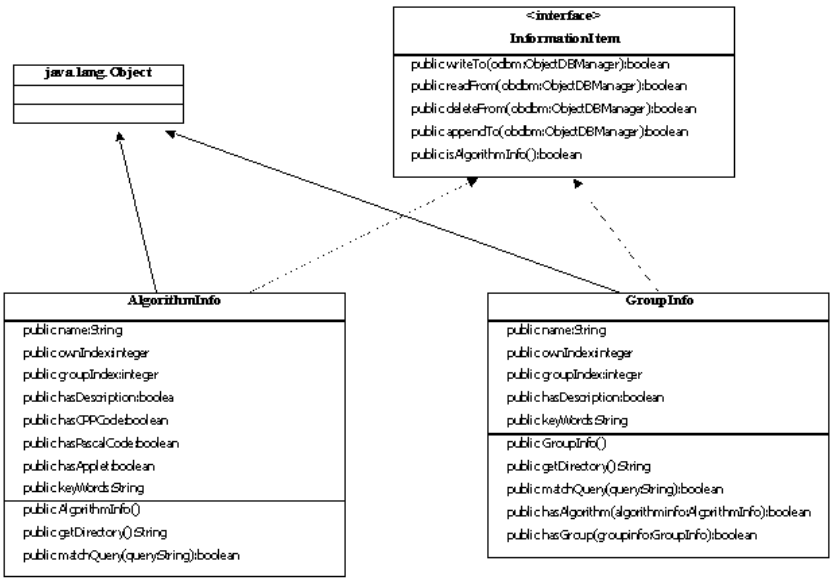

FIGURE 2. Class diagrams – AlgorithmInfo and GroupInfo

is the core of AD object storage section of **AD object networking/storage operations** level.

3.1. `AlgorithmInfo` **and** `GroupInfo` **classes.** Figure 2 presents class diagrams for `AlgorithmInfo` and `GroupInfo`. Both classes are derived from `java.lang.object` and implement the interface `InformationItem`.

Note the 1:1 mapping between their attributes and the structure of corresponding tables (**AlgorithmTable** and **GroupTable**). In order to speed up data manipulation and to decrease memory usage, all attributes are considered `public` (instead of declaring them `private` and using `get/set` methods). The `InformationItem` interface contains usual data manipulation operations: write, read, delete, and append. All these operations use a reference to an `ObjectManager` object. The method `isAlgorithmInfo` is used in dynamic identification of the receptor type.

3.2. `ObjectManager` **class.** `ObjectManager`'s object main task is to make persistent `AlgorithmInfo` and `GroupInfo` objects. The roles of `ObjectManager` support class are:

- to provide the infrastructure for storing/retrievig `AlgorithmInfo` and `GroupInfo` objects in/from a relational database (which contains the tables **AlgorithmTable**, **GroupTable**, **UnusedAlgorithmIds**, **UnusedGroupIds**, and **GlobalData**)
- to help `AlgorithmInfo` and `GroupInfo` objects in managing their own persistence
- to support queries referring to an algorithm or group of algorithms.

Figure 3 presents the diagram for `ObjectDBManager` class.

The `ObjectDBManager` object does not interact directly with the files in the main folders. It is used by the server in order to know if these files exist in the folder structure. **AD Management** component is responsible with creation and updating of these files.

4. How HPCC works

The AD management section of AD management and data delivery level is responsible with creating the standard structure of folders and files. First, the AD server section of the same level, by using AD object model, creates URLs for the root group which are sent to the client. On his behalf, client displays in a tree control the structure of HPCC Algorithm Store. When the user selects a specific algorithm/group, the client applet sends the algorithm/group id to the server, and the server builds the corresponding URLs, which are sent back to the client applet, which displays them in a window.

The user can specify queries by using keywords or algorithm/group names. The parameters are directed to the AD server, which builds the query string and uses ObjectDBManager::doQuery to retrieve the results, considered as a heterogeneous collection of AlgorithmInfo and GroupInfo objects. These results are sent back to the client applet, which displays them in a window.

5. Conclusions and future work

In this moment, the core part of HPCC application is already in place. The remaining components (as client presentation, networking, AD management) will be implemented soon.

References

[1] Avram, D., M. Iurian, B Pârv, *A High Performance Computing Center Based On A Local Network*, in SYNASC 2000, The Second International Workshop on Symbolic and Numeric Algorithms for Scientific Computation, West University, Timisoara, 4-6 Oct. 2000, 87-90.

[2] Pârv, B., *A Component-Based Model for Algorithms*, Babeş- Bolyai Univ., Fac. Math. Comp. Sci. Res. Sem, Seminar on Computer Science, 20 (1998), No. 2, 53-60.

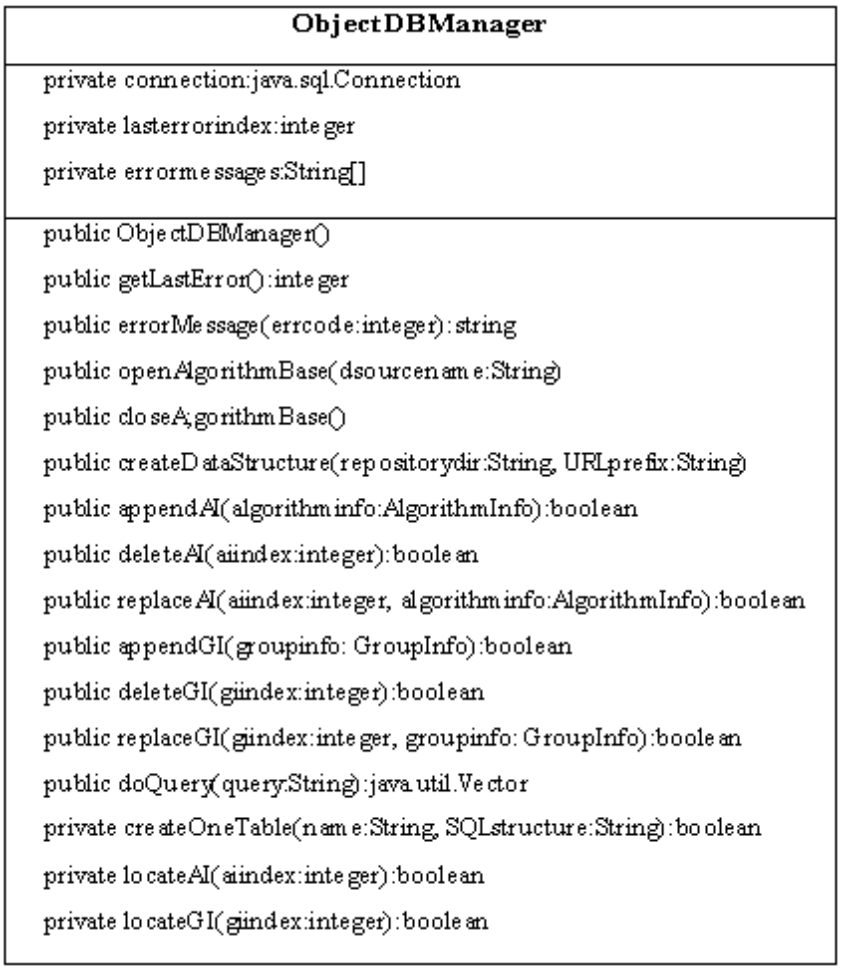

FIGURE 3. Class diagram – ObjectDBManager

[3] Pop D., S. Iurian, M. Iurian, B. Pârv, C. Mihoc, *Objectual Interfaces for Algorithm Databases*, Babeş-Bolyai Univ., Fac. Math. Comp. Sci. Res. Sem, Seminar on Computer Science, 21 (1999), No. 2, 35-42.

FACULTY OF MATHEMATICS AND COMPUTER SCIENCE, "BABEŞ-BOLYAI" UNIVERSITY, CLUJ-NAPOCA, ROMANIA
E-mail address: marin|parv@cs.ubbcluj.ro