A Distributed Environment for Solving Optimization Problems[1]

Fernando L. Dotti, Maurício de O. Cristal, Celso M. da Costa, Maurício L. Nunes
{ fldotti , mcristal, celso, mnunes } @inf.pucrs.br
Pontifícia Universidade Católica do Rio Grande do Sul
Faculdade de Informática - Programa de Pós-Graduação em Ciência da Computação
Av. Ipiranga, 6681 - Prédio 16 - CEP 90619-900 - Porto Alegre - RS - Brazil
Fone / fax: +55 51 320 36 11

Felipe Martins Müllcr
felipe@inf.ufsm.br
Universidade Federal de Santa Maria - Departamento de Eletrônica e Computação
Programa de Pós-Graduação em Engenharia de Produção
UFSM / CT / DELC - CEP 97105-900 - Santa Maria - RS - Brazil
Fone: + 55 55 220 83 69; Fax: +55 55 220 80 30

Abstract

This paper presents a distributed environment to support cooperation among institutions researching in the fleld of Operations Research, more specifically Optimization Algorithms. The proposed architecture for this environment respects the autonomy and heerogeneity of the cooperating institutions, promoting the organized reuse of solutions and enabling the workload to be shared by the cooperating institutions. The five functional blocks that build the structure of the architecture have their interfaces discussed. An implementation in Java/RMI is described. This implementation is currently being tested in a scenario involving two universities. Algorithms implementing classical heuristics are supported.

Key words: *Distributed Systems, Optimization Techniques, Cooperation, Autonomy, Heterogeneity, Java/RMI*

1 Introduction

Due to the growing competition in various sectors of the economy (industry, commerce, or services), there is a well defined trend towards the Optimization of current processes in order to achieve lower costs, better efficiency, and higher flexibility. Optimization techniques are thus becoming almost mandatory and widely used. Optimization problems are no rare complex, involving lots of variables, and having a combinatorial structure.

Therefore, the Operations Research scientific community is investing research efforts in this area. Better and new algorithms for solving combinatorial problems are under investigation in various research centers. Very often, these algorithms have similar structure, could reuse parts of the solutions, as well as use same input data sets to be tested. Metaheuristic methods (see

[1] Project partially supported by FAPERGS and CNPq/ProTem - Brazil.

Besides the extensions above mentioned, in the next phase of the project the Plan Manager should be developed.

7 References

[1] FUGGETA, A., PICCO, G. P., and VIGNA, G. Understanding Code Mobility. *Transactions on Software Engineering. IEEE* , v.24, 1998.
[2] CARZANINGA, A., PICCO, G. P., and VIGNA, G. Designing Distributed Applications *With Mobile Paradigms. [s.l.], [199-].*
[3] KNUDSEN, P.. Comparing Two Distributed Computing Paradigms - A Performance Case Study. M. S. Thesis, University of Troms0, 1995.
[4] OMG-Document. Common Object Request Broker Architecture. 1997. *http://www.omg.org. corba.*
[5]OMG-Document. CORBA Services: Common Object Services Specification. 1997. *http://www.omg. org. corba.*
[6] SIEGEL, J. CORBA fundamentais and programming. New York, NY : J. Wiley, 1996. 693p.
[7] HENING, Michi. Binding, Migration and Scalability in CORBA. Communications of the ACM, vol 41, no 10, october 1998.
[8] VINOSKI, S. New Features for CORBA 3.0. *Communications of the ACM,* vol 41, no 10, October 1998.
[9] VINOSKI, S. CORBA: Integrating Diverse Applications Within Distributed Heterogeneous Environments, *IEEE Communications Magazine,* 1997.
[10] Open Distributed Processing: Annex A - Tutorial on Trading Function. pp. 28. 1998.
[11] ORFALI, R. and Harkey, D. Client/server Programming with Java and CORBA Second Edition; New York, NY: John Wiley and Sons Inc
[12] FARLEY, J. Java Distributed Computing. The Java Series. Califórnia: O'Reilly Associates. 1998.
[13] CORNELL, G. and HORSTMANN, C. S. Core Java. Makron Books, São Paulo, 1997. 807 p.
[14] GAREY, M. R. and JOHNSON, D. S., Computers and Intractibility: a Guide to the Theory of NP-Completeness, Freeman, San Francisco, 1979.
[15] ZANAKIS, S. H. and EVANS, J. R., Heuristic "optimization", why, when, and how to use it, Interfaces, 11(5), 1981.
[16] BAL, H. E., KAASHOEK, M. F., and TANEMBAUM, A. S., Orca: a language for parallel programming of distributed systems. IEEE Transactions on Software Engineering, Vol. 18(3), March 1992, pp. 190-205.
[17] AUSTIN, S. An introduction to genetic algorithms, AI Expert, March, pp. 49-53, 1990.
[18] DAVIS, L. (Ed.) Handbook of Genetic Algorithms, Van Nostrand Reinhold, New York, NY, 1991.
[19] EGLESE, R.W. Simulated annealing: a tool for operational research, European Journal of Operational Research, 46, pp. 271-281, 1990.
[20] FRANÇA, P. M., GENDREAU, M., LAPORTE, G., and MÜLLER, F. M. A tabu search heuristic for multiprocessor scheduling with sequence dependent setup times, International Journal of Production Research, Elsevier Science Publishers, Vol. 43(1), pp. 79-89, 1996.
[21] GLOVER, F., Tabu search: part I, ORSA Journal on Computing, Vol. l, pp. 190-206, 1989.

Manager as it is. Our choice for a Java only implementation using RMI, is because CORBA revealed to have some limitations and sometimes to be cumbersome to use. The programmer has to deal with two languages (e.g. IDL and the implementation language); no object by value passing is allowed, which sometimes leads to problems while implementing; code portability is not always possible in the practice because some products do not yet implement the Portable Object Adapter. Also due to the plans to implement some parts of the architecture with mobile code, the option for a Java only implementation seemed to be more realistic since migration and mobility of CORBA objects is more complicated than with Java/RMI.

One of the advantages in the integration of legacy optimization algorithms is that their interface is simple, they receive input data, process and returns the output data. Input and output data have the form of byte strings. It is simple to implement Java modules to launch (*Runtime.exec()*) optimization processes and connect to the input and output streams to provide input and retrieve output data.

- Java support: The experience using Java is satisfactory, but not free of problems. The learning curve is acceptable and prototyping is quite fast. For distributed computing, there are some advantages if compared to CORBA, as listed above. However, the lack of a formal definition of the language semantics lets the programmer sometimes without the needed support. This is true specially when the programmer comes into detail in the use of the threads library. Group communication support was also felt important although not available.
- Trader Service: With respect to the CORBA-Services [5], we have made an attempt to use a free version of the Trader Service [10][5]. Using the Trader, we would have to either have each optimization algorithm running and achievable through a CORBA interface awaiting for requests, or have an Implementation Repository to launch the appropriate implementation whenever a request happens.

In the first case, processes would be awaiting for requests, resulting in processing costs. Furthermore, transient interfaces [7] would have to be used and this would impact in the administration of the Centers, since references to a service could become invalid. We have made the attempt to use the Trader with persistent references [7] and use an Implementation Repository. However, since we have not found a ORB with an Implementation Repository and a Trader, we would have to use two ORBs (two manufacturers), one supporting the Trader and another with the Implementation Repository. This was not possible because at that time the manufacturers did not grant that the references obtained in a ORB could be exported to the Trader written for the other ORB. Therefore we have implemented in Java our own way to find and launch optimization algorithms in a distributed context.

- Policies for load balancing: currently, the decision of where to execute an algorithm, given that several nodes are candidate, is based on the number of processes already running in each node. Extensions are planned to consider a load indicator that considers the heterogeneity of the nodes. Also, we do not consider the costs of transferring input and output data to and from the place where the process executes because the volume of input and output data of optimization algorithms is easily handled by current networks.
- Mobile code: work is in progress to identify the possibility of using mobile code paradigms [1][2][3] in the architecture proposed. Until now it turned out that mobile code can be a well suited approach to implement the search of a better place to execute, request the execution in the chosen place, and deliver the results back to the originator. This is because for this case design is more intuitive and programming seems to be more intuitive than the classical approach.

To build this metacomputing environment, a key module is the metaNEOS resource broker. This module is responsible for the global resource allocation. It manages schedulers defined by metaNEOS, interacts with schedulers local to the nodes non exclusively dedicated to metaNEOS processing, as well as deal with heterogeneity aspects in the environment. metaNEOS plans to support local resource allocation using techniques defined in CONDOR [31] and GLOBUS [34].

The table below compare important aspects of the projects commented with the environment described in this paper.

	NetSolve	NEOS	metaNEOS	This paper
Supports	Numerical libraries	Optimization algorithms	Optimization algorithms	Optimization algorithms
Programming languages	Only C and Fortran	All languages	All languages	All languages
Administrator	Not mentioned	Not centralized, each administrator is responsible for one solver.	Not centralized, each administrator is responsible for one solver.	Centralized, each administrator is responsible for a group of nodes (center)
Request **submission**	By compiled programs	By email, WWW and sockets through an interface application	By email, WWW and sockets through an interface application	By WWW
Resource **allocation** / **infrastructure**	Nodes dedicated to the project	Nodes dedicated to the project	Nodes dedicated and non-dedicated to the project	Nodes dedicated to the project(*)
Prototype/ implementation	Available	Available	Not available yet	Available

(*) Our architecture runs also on nodes non-dedicated to the project. However, the load balancing mechanism does not yet take into consideration other sources of load.

Table 1: NetSolve, NEOS and metaNEOS aspects compared to this paper

6 Conclusions and Future Work

This paper presents the architecture and implementation of a distributed environment to promote the cooperation in the research area of Operations Research, more specifically Optimization Algorithms. The current implementation is under test in a distributed environment involving two universities (Pontifícia Universidade Católica do Rio Grande do Sul and Universidade Federal de Santa Maria) both in Brazil, 500km far from each other. Below, we organize our concluding remarks in 5 main parts.

- CORBA: CORBA [4][8] is currently pointed as platform for integration of legacy applications [9]. We have not used it in our implementation. However, as Optimization Centers are autonomous, it could be used within a center, keeping the cooperative interface of the Center

Figure 8 represents the processing of a request after the center with lowest load has been selected. If the user request is synchronous, thread 1 blocks and waits for the end of the execution notified through the returnResult. If not (asynchronous) the user interface is unblocked and thread 1 sends an electronic mail to the user with the result.

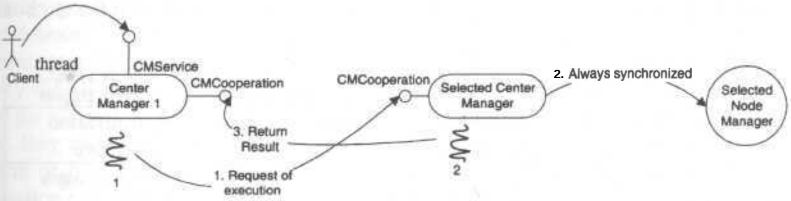

Figure 8: Request processing

5 Related Work

Concerning related work, three research projects can be mentioned: *NetSolve* [30], *NEOS* [32] and *MetaNEOS* [33]. NetSolve was developed at Tennessee University together with the Oak Ridge National Laboratory. NEOS (acronym for Network-Enabled Optimization System Server) and MetaNEOS were developed by Argonne National Laboratory. Below, architectural and functional aspects of these projects are briefly commented.

NetSolve builds a distributed environment where users can submit requests to remote servers supporting numerical libraries. Communication modules called agents are responsible for receiving requests from the user and submitting them to the appropriate server. This environment eases the access to those libraries supporting their location, download and installation. After [30] these are time consuming tasks for the academic community.

NetSolve agents support communication among servers and are responsible to find out the better server for a given requisition through a load balancing policy. The architecture developed for NetSolve allows for servers or agents to be dynamically started and killed without ompromising the integrity of the system. Fault tolerance aspects were also considered in the rchitecture.

NEOS is an environment developed to allow the resolution of optimization problems using he internet. It works following a standardized structure for input data and has a list of registered optimization problem solvers. Through the internet, the user chooses the problem solver, informs the input data set, and issues a request. At the end of the execution, the users receives back the results and run time statistics. To each registered problem solver a manager is assigned which is responsible for the computational resources needed by the server and for answering possible questions coming from users. To cope with resource allocation management (e.g. load balancing) in a distributed environment, NEOS uses the CONDOR environment [31].

The *metaNEOS* project [33] is a proposal to add functionality to the already existing environment supported by NEOS. It aims at solving optimization requests in a metacomputing environment. A metacomputing environment is an abstraction to represent the computational resources existing in an infrastructure of loosely coupled processor nodes (e.g. the Internet). The user transparently uses the computational power of the environment without being aware of distribution aspects.

that supports it. This query results in a node reservation in the consulted center associated to the `processId`;

- `executeSelectedCenter`: after a center discovers the federating center with lowest load for an algorithm, this method is used to ask the selected center to execute its request;
- `release`: the other consulted centers (non-selected) are asked to release the reservation;
- `returnResult`: when the selected center ends the execution, it returns the result to the requester center by calling this method.

Detailed Request Execution

The user may issue synchronous or asynchronous execution requests. In the synchronous request, the user waits for the end of the operation, while in the asynchronous request the user receives the result by electronic mail. Both methods involve the search for the node with lowest load in the federation.

When a request is issued to a Center, it consults all federating Centers to discover where the request should be processed. Two policies are explicit here: a) exhaustive search in the federation for the best location, and b) search performed every time a request is issued. These policies were chosen because optimization algorithms are CPU bound and the communication and computing costs to find a place to execute is worthy if the best place is found in the federation.

The second policy grants that the decision is taken based on the freshest state of the federation in terms of load (since when the search is performed, a Center blocks/reserves the best option until the originating center either submits the request or releases the reservation with the center). The reservation with a center follows a soft-state approach, i.e., if the originating center does not answer, either submitting the request or releasing the reservation, the reservation expires and the resources are released. This policy is to avoid invalid reservations in the case of failure of the originating center or in the communication with it.

The search for the node with lowest load is done through synchronous communication. During this process, for each center in the table of federating centers, a thread is started to communicate with that center asking for the load of the node with lowest load in that center supporting the wanted algorithm (using lookForLowestLoad). Each center receiving the request searches in its nodes for the node with lowest load for that algorithm. The originating center compares all loads returned and selects the center with the lowest one. Figure 7 represents this process.

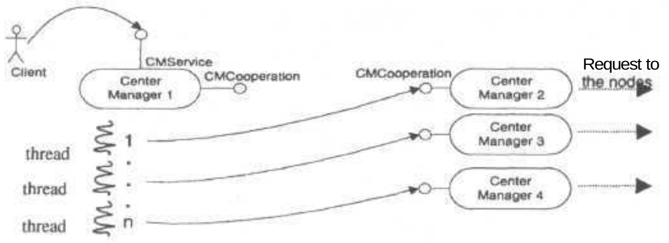

Figure 7: Load Balancing

- getCenterName: returns the name of a center;
- addCenter: adds a center identiflcation (an URL) in the Center Table. This operation enables the remote center to federate with this one receiving service requests issued by this center as well as issuing service requests to this center;
- dropCenter: deletes the center identiflcation of the Center Table. Ends the cooperation between this center and the remote center;
- addNode: adds a node identiflcation (an URL) in the Node Table. With this operation a new node is inserted in an optimization center;
- dropNode: deletes the node identiflcation of the Node Table;
- listNodes: returns all nodes contained in the Node Table.

The service interface of the Center Manager is defined as follows:

```
public String execute(String algorithmId, String urlDataSetName,
                           boolean SyncComm, String userEmail);
public String addDataSet(String dataSetName);
public Vector queryAllAlgorithms();
public Vector queryAllDataSets();
```

Figure 5: CMService interface

- execute: used by the user interface to submit an optimization request to a center manager. The process involved in executing a request is described bellow, in subsection "Detailed Request Execution";
- addDataSet: the inclusion of input data sets is a task that the user may start through the CMService interface. The data set included by the user is kept by the center in a data base (as presented in Figure 2.B), it remains in that base for a default time configured by the administrator and after that it is excluded from the center. The parameter DataSetName reports the name that will be used to store the data set in the center. This function returns the complete URL for the stored data set. The URL returned is the data set identifier in the system;
- queryAllAlgorithms: returns all available algorithms in the federation. The user interface needs this method to present to the client all available algorithms, such that the client may select one of them;
- queryAllDataSets: returns all available data sets in the federation.

The cooperation interface of the Center Manager is defined as follows:

```
public int lookForLowestLoad(String algorithmId, String processId);
public void executeSelectedCenter(Request objRequest,
                                  String returnToCenter);
public void release(String processId);
public void returnResult(String result, String processId);
```

Figure 6: CMCooperation interface

- lookForLowestLoad: through this method a federating center may ask if the algorithm with algorithmld is supported and, in case affirmative, the load associated to the node

More specifically, this section discusses the setup of the whole environment, the interfaces supported by each module, and shows the steps involved in the processing of an optimization request in the federation.

Setup of the environment

The system administrator is responsible for choosing the nodes in a network that will participate in an Optimization Center. In each of those nodes, a Node Manager is installed, offering the respective service and management interfaces. The control of the Node Managers is made through their management interface. The administrator updates the Algorithms Table of each node in the Center.

The administrator configures then the Center Manager, offering service and management interfaces. The configuration of this module includes setting Unified Resource Locators to identify the Node Managers (i.e. the URL of their service interfaces) of the nodes in the Center, and the Center Managers of federating Optimization Centers (i.e. the URL of their cooperation interface).

Interface Definitions

The management interface of the Node Manager is defined as follows:

```
public void addAlgorithm(String algorithmId, String algorithmPath,
                          String algorithmParameters);
public void dropAlgorithm(String algorithmId);
public int queryLoad(String algorithmId);
public Vector listAlgorithm();
```

Figure 3: NMManagement interface

- `addAlgorithm`: adds an optimization algorithm in the Algorithms Table of a node. The `algorithmld` must be unique in the node and the executable files already downloaded to the `algorithmPath` directory in the node;
- `dropAlgorithm`: withdraw the algorithm from the Algorithms Table of a node.
- `queryLoad`: returns the number of optimization processes running in this node, it answers at the same time if the node supports a given algorithm;
- `listAlgorithm`: returns a list of all available algorithms in this node.

The management interface of the Center Manager is defined as follows:

```
public String getCenterName();
public void addCenter(String urlCenterManager);
public void dropCenter(String urlCenterManager);
public void addNode(String urlNodeManager);
public void dropNode(String urlNodeManager);
public Vector listNodes();
```

Figure 4: CMManagement interface

heuristic will serve as input for the next one. Parameters to control this execution like number of repetition of a given cycle in the graph are also specified by the metaheuristic as well.

Metaheuristics specify what we have called an "execution Plan". The Plan Manager, showed in Figure 2.C, is responsible to manage the execution of a metaheuristic specified or chosen by the user. This module is a user of the Center Manager and issues optimization requests to execute classical heuristics in the federation, corresponding to single execution steps of the Plan. The results of a single execution answered by the Center Manager will serve as parameter for another request. The intermediate results are stored in containers, the Plan Manager deals only with references to these storage places.

Before executing, the specification of a metaheuristic has to pass through a verification phase where it is checked if each single step (optimization algorithm used) of the Plan can be supported by the federation. The choice of where to execute each single classical heuristic is transparent to the Plan Manager, being a responsibility of the Center Manager. Studies are to be carried out about how to choose the best node to perform an execution step taking into consideration the whole execution Plan.

The Plan Manager keeps information about already submitted metaheuristics, such that users may choose among an available one or submit a new one. The Plan Manager offers the following interfaces:

- A *service interface*, called PMService, offering the users the possibility of submitting new metaheuristic specifications or choosing among already submitted ones. This interface offers also the possibility of Consulting heuristics, metaheuristics, and input data sets available in the context of the federation;
- A *management interface*, called PMManagement, through which the human manager of the Optimization Center is able to control the base of metaheuristics and to modify other properties internai to the Plan Management Module.

User Interface

The User Interface is the module responsible for interacting with human users of an Optimization Center. While the interface communicates with a center, the service perceived by the user is relative to the whole federation. The User Interface may communicate with the Plan Management Module for submitting metaheuristics or with the Center Management Module to use classical heuristics. As optimization algorithms may have long execution times, the interface offers the possibility for the user to receive the results in an asynchronous manner, via e-mail for instance. With this, the user can launch a request and disconnect from the center.

4 Implementation

The implementation described here is limited to 4 of the 5 modules: User Interface, Center Manager, Node Manager and Executor. With these modules, users may publish, share and solve classical heuristics. The Plane Management Module is left for a second phase, i.e. metaheuristics are not supported in this implementation.

With exception of the Executor Modules, which represent legacy algorithms already implemented in various languages, the other modules were implemented in Java. The interfaces described in the architecture are mapped to Java/RMI interfaces [11][12][13].

Center Management

The Center Manager, showed in Figure 2.B, controls an Optimization Center which is comprised of nodes devoted to the execution of optimization algorithms and may cooperate with other Optimization Centers. For such cooperation, the Center Management Module interfaces with its counterpart in another Optimization Center. This module accepts optimization requests from user or cooperating modules and delegates it to the most appropriate internai node, using the Node Management Modules. The Center Manager keeps three main structures:

- A *table ofnodes* or machines in the Optimization Center: this structure keeps information about the nodes working in the Center;
- A *table of federatingOptimization Centers:* this structure keeps information about other Optimization Centers that a Center may federate with;
- A *table ofrequisitions*: offers information relative to each ongoing optimization process in the Center, keeping a process identification and the current state of the process (consulting load, executing, ready).

The Center Management Module offers the following interfaces:

- The *service interface,* called CMService, allows user modules to request optimization services. For the user modules, the Center Management Module represents the whole federation because the requests may be passed to any other federating Optimization Center through the cooperation interface. This interface supports also user modules to query about supported algorithms and input data sets in the federation, as well as to publish new input data sets in the federation;
- The *cooperation interface,* called CMCooperation, allows an Optimization Center to pass optimization requests to federating Optimization Centers (registered in the table of centers). Before asking a federating Center to perform an optimization request, there is a phase in which centers consult the availability of the algorithm in the other centers and the existing load in the nodes where the algorithm is supported. More details are described in the implementation section. The cooperation interface allows also to consult for algorithms and input data sets available in a center;
- The *management interface,* called CMManagement, allows a human manager in the Optimization Center to modify the properties of the center. Nodes can be added and deleted from the center, federating centers can be registered and excluded.

Plan Management

The structure described up to the Center Management Module supports user modules to request execution of a single execution step, i.e. supports requests to a given optimization algorithm with a given input data set. The Optimization Center that receives this request from the user modules finds out through the federation the best center and node to execute that request. The request is then processed and the results are returned to the originating center, which then returns it to the user module that issued the request. This whole structure is enough for "heuristic" optimization algorithms.

"Metaheuristic" algorithms base on classical algorithms. They specify data dependencies between classical heuristics in a chained execution, which can be seen as a graph where nodes represent the execution of a classical heuristic and branches represent that the output from a

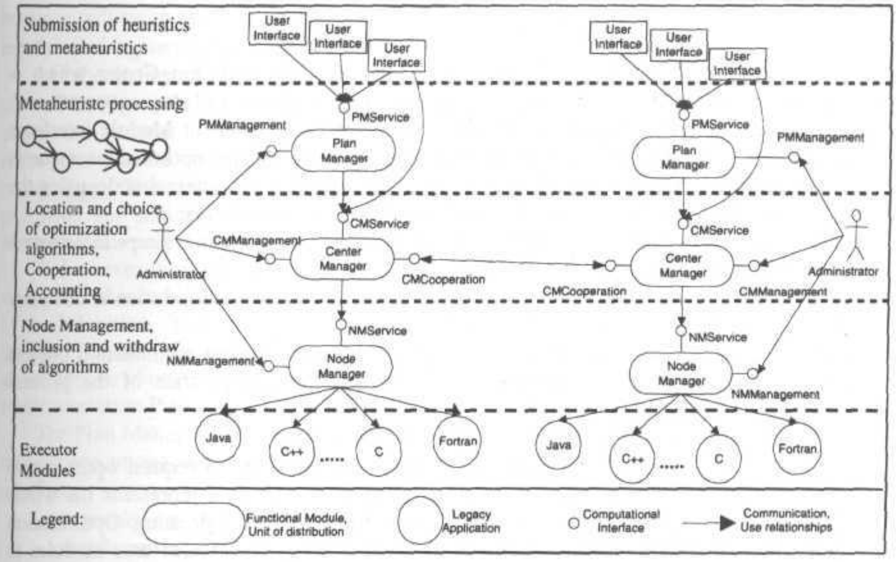

Figure 1: Proposed Architecture

The Node Manager offers two interfaces:

- The *service interface*, called NMService, allows user modules (the Center Management Module) to ask about the load in the node and to require Optimization algorithms to be performed in that node.
- The *management interface*, called NMManagement, allows a human manager in the Optimization Center to consult and modify the properties of the node by Consulting, inserting and deleting existing Execution Modules.

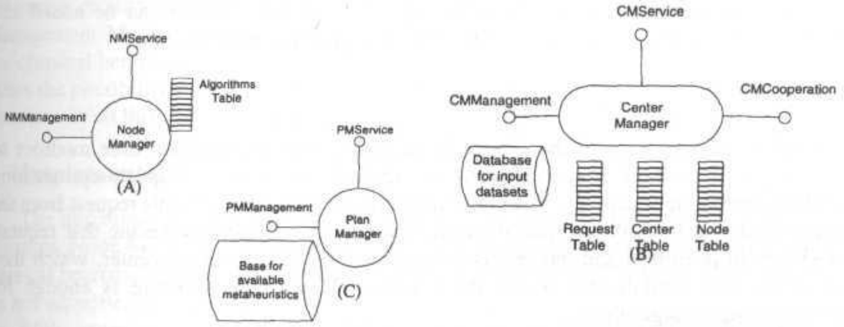

Figure 2: (A) The Node Manager, (B) Center Manager, (C) Plan Manager

[19][24][25], Genetic [18][17][23][26] and Memetic algorithms. Optimization algorithms are heavily CPU bound, they receive input data, process it, and give an output. Input and output are normally not volumous.

3 The Architecture

The main objective of the architecture described is to offer an environment to support the cooperation between persons and institutions researching in the area of combinatorial Optimization. The following features are identified:

- Distribution: the scenario is inherently distributed, as various institutions physically dispersed may want to cooperate;
- Autonomy and Cooperation: due to the autonomy of the institutions, a hierarchical approach is regreted, and a cooperative one is followed. Each institution is able to provide a support infrastructure configured according to the local decisions, called an "Optimization Center". Optimization Centers may cooperate with each other to solve Optimization problems issued by their users. In this case a group of Optimization Centers will build a "Federation";
- Heterogeneity: as a consequence of autonomy, the heterogeneity of computational infrastructures must be taken into consideration;
- Reusability: due to the similar structure of combinatorial algorithms, reuse of Solutions is to be promoted. Reuse of available input data sets is to be supported too.

The architecture here described considers these features. The implementation described in the next section brings more detailed aspects about this architecture. Each Optimization Center has five kind of modules: Executors, Node Managers, a Center Manager, a Plan Manager, and User Interfaces. Figure 1 depicts the whole architecture.

The functional modules may have various computational interfaces. Each module is a unit of distribution, i.e. the various modules can be allocated to different nodes in a computer network. The computational interfaces are defined following an object oriented methodology, as well as the information passed between the modules. Each module of the architecture is discussed next.

Executor Module

Represents available Optimization algorithms. These algorithms may be available for different computational platforms, and may be written in different programming languages. Coupling them to the architecture is a case of integration of legacy applications.

Node Management

The Node Manager, showed in Figure 2.A, manages a physical node which participates in an Optimization Center. It controls the execution of Executor Modules registered with that node. Upon requisition for an Optimization algorithm, it searches the appropriate Executor Module, fires the execution and returns the results back to the requiring module. The node manager thus keeps a table with information about supported algorithms and associated Executor Modules.

Section 2) combine various solutions in a higher levei one, whereby it is important to easily combine (chain) executions of existing algorithms.

In order to promote the research work in this area (Operations Research - Optimization Algorithms), this paper proposes a distributed environment supporting a structured way to exchange results among researchers. Due to the intrinsic characteristics of distribution and autonomy of cooperating parts (research centers, universities, enterprises, etc.), this environment is organized in the form of a federation of cooperating optimization centers. Each optimization center is capable of supporting the life-cycle of optimization algorithms, that can be published both in the form of source code as well as running processes solving optimization requests. A center supports also a base of ínput data sets to test optimization algorithms. This base can be extended with contributions from the users. In the context of a federation, optimization centers share processing power cooperating to better solve the requests issued by users of that federation.

This text is organized as follows. In Section 2 we give a brief overview of optimization techniques. In Section 3 a distributed architecture to support the cooperative environment is proposed. In Section 4 an implementation of this architecture is described. A distributed environment based on this implementation is being tested involving two universities. In Section 5 conclusions and future developments are outlined.

2 Optimization Techniques - A Historical Perspective

The class of problems we focus here is the one with combinatorial structure. It is known that most combinatorial problems of practical interest can be classified as NP-Complete, for which there are no known efficient algorithms, i.e. algorithms that achieve an optimal solution in a polynomial time [14] (Garey e Johnson -1979).

As exact polynomial algorithms probably do not exist, research efforts were directed to investigate methods to achieve good solutions to combinatorial problems in reasonable times. So, approximated or heuristic methods gained in importance as they allow to treat real problems of considerable dimensions in reasonable processing times. However, the quality of the solutions obtained through classical heuristics was not satisfactory for all the problems in the class considered. Therefore, the discussion of which heuristic is better suited to what problem emerged. This could lead to an exhaustive analysis of all of them, comparing the complexity of the implementation, quality of solutions, computational efficiency, robustness and so on [15] (Zanakis & Evans - 1981) [16] (Ball & Magazine - 1981). Such a detailed study could point to the use of one heuristic for a given problem. However, using only one heuristic the good qualities of the others could be wasted. The idea of organizing these heuristics in a team comes up to use the better characteristics of each one and achieve better solutions than using isolated heuristics.

In addition to this, research efforts have been also directed towards metaheuristics [27]. Metaheuristics are coupled to the classical heuristics resulting in better solutions mainly because they can transcend local optimality and add knowledge to search structures.

So, optimization algorithms can be classified in heuristic and metaheuristic. In the class of heuristic algorithms, there are "construction" and "improvement" algorithms. Constructive algorithms generate a first solution to a problem (e.g. Nearest Neighbor, Nearest Insertion, Farest Insertion, Cheapest Insertion). Improvement algorithms start from a given solution and modify it to make enhancement (e.g. Lin-Kernigham). Metaheuristic algorithms may manage a chain or flow of executions of classical heuristics, e.g. Tabu Search [20][21][22], Simulated Annealing

[22] GLOVER, F., Tabu search: part II, ORSA Journal on Computing, Vol. 2, pp. 4-32, 1990.

[23]GOLDBERG, D. E., Genetic Algorithms in Search, Optimization, and Machine Learning, Addison-Wesley, Reading, MA, 1989.

[24] KIRKPATRICK, S., GELATT Jr., C. D., and VECCHI, M. P. Optimization by simulated annealing, Science, Vol. 220, pp. 671-680, 1983.

[25] van LAARHOVEN, P. J. M., and AARTS, E. H. L. Simulated Annealing: Theory and Applications, Reidel, 1987.

[26] MICHALEWICZ, Z., Genetic Algorithms + Data Structures = Evolution Programs, Springer-Verlag, 1992.

[27] TSUBAKITANI, S. e EVANS, J. R., An empirical study of a new metaheuristic for the traveling salesman problem, Q A-1992-05 Working Paper, Dept. of Quantitative Analysis, College of Business Administration, University of Cincinnati, 1992.

[28] WIRFS-BROCK, R. J. e JOHNSON, R. E. Surveying current research in object oriented design. Communications of ACM, 32(9): 104-124, Sept. 1990.

[29] COULORIS, G., DOLLIMORE, J. e KINDBERG, T. Distributed Systems: Concepts and Design, Addison-Wesley, 1994.

[30] CASANOVA, H., AND DONGARRA, J., NetSolve: A Network server for solving computacional science problems, Technical Report CS-95-313, University of Tennessee, Knoxville, Tennessee, 1995.

[31] LITZKOW, M. J., LIVNY, M., AND MUTKA, M. W., Condor - A hunter for idle workstations, in Proceedings of the 8th International Conference on Distributed Computing Systems, Whashington, District of Columbia, 1988, IEEE Computer Society Press, pp. 108-111.

[32] CZYZYK, J., MESNIER, M.P. AND MORE, J.J., The Network-Enabled Optimization System (NEOS) Server, Preprint MCS-P615-1096, Argonne National Laboratory, Argonne, Illinois, 1997.

[33] METANEOS: METACOMPUTING ENVIRONMENTS FOR OPTIMIZATION, propose available at http://www.mcs.anl.gov/metaneos, 1998.

[34] GLOBUS RESOURCE MANAGER SPECIFICATION. Globus working note, available at http://www.globus.org, 1997.