A PARALLEL PROGRAMMING METHODOLOGY USING COMMUNICATION PATTERNS NAMED CPANS OR COMPOSITION OF PARALLEL OBJECT

M. Rossainz-López[(a)], M. I. Capel-Tuñón[(b)]

[(a)] Universidad Autónoma de Puebla, Avenida. San Claudio y 14 Sur, San Manuel, Puebla, State of Puebla, 72000, México
[(b)] Departamento de Lenguajes y Sistemas Informáticos, ETS Ingeniería Informática, Universidad de Granada, Periodista Daniel Saucedo Aranda s/n, 18071 Granada, Spain

[(a)]mariorl@siu.buap.mx, [(b)]mcapel@ugr.es

ABSTRACT
Within an environment of Parallel Objects, an approach of Structured Parallel Programming and the paradigm of the Orientation to Objects, shows a programming method based on High Level Parallel Compositions or HLPCs (CPANs in Spanish) by means of classes. The synchronous, asynchronous communication ways and asynchronous future of the pattern of Parallel Objects (Rossainz and Capel 2005-2), the predefined patterns of communication/interaction of the structured approach, the encapsulation and the abstraction of the Orientation to Objects, to provide reusability to this patterns, together with a set of predefined restrictions of synchronization among processes (maxpar, mutex, sync) are used. The implementation of the commonly used communication patterns is explained, by means of the application of the method, which conform a library of susceptible classes of being used in applications within the environment of programming of the C++ and of the standard POSIX of programming with threads.

Keywords: Parallel Objects, Structured Parallel Programming, High Performance Computing, Object Oriented Programming.

1. INTRODUCTION

As it is known, exist infinity of applications that using machines with a single processor tries to obtain the maximum performance from a system when solving a problem; however, when such a system can not provide the performance that is required (Capel and Troya 1994), a possible solution it consists on opting for applications, architectures and structures of parallel or concurrent processing. The parallel processing is therefore, an alternative to the sequential processing when the limit of performance of a system is reached. In the sequential computation a processor only carries out at the same time an operation, on the contrary of what happens in the calculation parallel, where several processors they can cooperate to solve a given problem, which reduces the time of calculation since several operations can be carried out simultaneously. From the practical point of view, today in day is enough justified carrying out compatible investigations within the area of the parallel processing and areas related (Concurrence, Distributed Systems, Systems of Real Time, etc.), since the recent advance in massively parallel systems, communications of great band width, quick processors for the treatment of signs, etc., they allow this way it. Important part of those investigations are the parallel algorithms, methodologies and models of parallel programming that at the moment are developing. The parallel processing includes many topics that include to the architectures, algorithms, languages of programming parallel and different methods of performance analysis, to mention some of the most excellent.

The present investigation centers its attention in the Methods of Structured Parallel Programming, proposing a new implementation with C++ and the library of threads POSIX of the programming method based on the pattern of the High Level Parallel Compositions or CPANs (Corradi 1995; Danelutto), the which it is based on the paradigm of Orientation to Objects to solve problems parallelizable using a class of concurrent active objects. In this work supply a library of classes that provides the programmer the communication/interaction patterns more commonly used in the parallel programming, in particular, the pattern of the pipeline, the pattern denominated farm and the pattern tree of the technique Divide and Conquer of design of algorithms, well-known as it.

2. MOTIVATION

At the moment the construction of concurrent and parallel systems has less conditioners every time, since the existence of systems parallel computation of high performance, or HPC (High Performance Computing), more and more affordable, has made possible obtaining a great efficiency in the processing of data without the cost is shot; even this way, open problems that motivate the investigation in this area still exist; in particular, interest us those that have to do with the parallel applications that use communication patterns

predetermined among their component software. At the moment are identified as important open problems, at least, the following ones:

The lack of acceptance of environments of parallel programming structured to develop applications: the structured parallelism is a type of parallel programming based on the use of communication/interaction patterns (pipelines, farms, trees, etc.) predefined among the processes of user's application. The patterns also encapsulate the parallel parts of this application, of such form that the user only programs the sequential code of this one. Many proposals of environments exist for the development of applications and structured parallel programs, but until the moment, they are only used by a very limited circle of expert programmers. At the moment, in HPC, a great interest exists in the investigation of environments as those previously mentioned ones.

The necessity to have patterns or high level parallel compositions: a high level parallel composition or CPAN, as well as is denominated in (Corradi 1995), it must be been able to define and to use within an infrastructure (language or environment[1] of programming) oriented to objects. The components of a parallel application not interaction in an arbitrary way, but regular basic patterns follow (Hartley 1998). An environment of parallel programming must offer its users a set of components that implement the patterns or CPANs more used in algorithms and parallel and distributed applications, such as trees, farms, pipes, etc. The user, in turn, must can to compose and to nest CPANs to develop programs and applications. The user must be limited to a set of predefined CPANs, but rather, by means of the use of the inheritance mechanism, he must can to adapt them to his necessities. The development environment must contemplate, therefore, the concept of class of parallel objects. Interest exists in exploring the investigation line related with the definition of complete sets of patterns, as well as in its semantic definition, for concrete classes of parallel applications.

Determination of a complete set of patterns as well as of their semantics: in this point, the scientific community doesn't seem to accept in a completely satisfactory way and with the enough generality none of the solutions that have been obtained to solve this problem today. It doesn't seem, therefore, easy the one that can be found a set the sufficiently useful and general thing, for example a library of patterns or set of constructos of a programming language, to be used in the development, in a structured way, of a parallel application not specific.

[1] One talks about the concept of *HPC programming environment* : environment of "friendly" parallel programming to users that provide facilities for the development of applications, abstracting details of low level as the referred ones to the creation, allocation, coordination and communication of the processes in a distributed and parallel system.

Adoption of a approach oriented to objects: Integrating a set of classes within an infrastructure oriented to objects is a possible solution to the problem described in the previous point, since would allow adding new patterns to an incomplete initial set by means of the subclasses definition. Therefore, one of the lines of followed investigation has been finding representations of parallel patterns as classes, starting from which instance parallel objects is been able to (CPANs) that are, in turn, executed as consequence from an external petition of service to this objects and coming from user's application. For example, the derived pattern of the execution for stages of the processes would come defined by pattern's denominated pipeline class; the number of stages and the sequential code of each specific stage would not be established until the creation of a parallel object of this class; the data to process and the results would be obtained of user's application; the internal storage in the stages could adapt in a subclass that inherits of pipeline. Several advantages are obtained when following a approach oriented to objects (Corradi and Leonardi 1991), regarding a approach only based on skeletons algorithmic and programs model (Hartley 1998), it is necessary to point out, for example, the following improvements:

Uniformity: All the entities within the programming environment are objects.

Genericity: The capacity to generate references dynamically, within an environment of software development oriented to objects, makes possible the creation of generic patterns, by means of the definition of its components as generic references to objects.

Reusability: The inheritance mechanism simplifies the definition of specialized parallel patterns. The inheritance applied to the behavior of a concurrent object helps in the specification of the parallel behavior of a pattern.

3. EXPOSITION OF THE PROBLEM

From the work carried out to obtain the investigating sufficiency presented in Julio 1999, redefining and modernizing the investigation, the problem to solve is defining a Parallel Programming Method based on High Level Parallel Compositions (CPANS) (Corradi 1995). For it the following properties have considered as indispensable requirements that should be kept in mind for the good development of this investigation. It is required, in principle, a environment of Programming Oriented to Objects that it provides: Capacity of invocation of methods of the objects that contemplates the asynchronous communication ways and asynchronous future. The asynchronous way doesn't force to wait the client's result that invokes a method of an object. The asynchronous future communication way makes the client to wait only when needs the result in a future instant of her execution. Both communication ways allow a client to continue being executed concurrently with the execution of the method (parallelism inter-objects).

The objects must can to have internal parallelism. A mechanism of threads it must allow to an object to serve several invocations of their methods concurrently (parallelism intra-objects).
Availability of synchronization mechanisms when parallel petitions of service take place. It is necessary so that the objects can negotiate several execution flows concurrently and, at the same time, to guarantee the consistency of their data.
Availability of flexible mechanisms of control of types. The capacity must be had of associating types dynamically to the parameters of the methods of the objects. It is needed that the system can negotiate types of generic data, since the CPANs only defines the parallel part of an interaction pattern, therefore, they must can to adapt to the different classes of possible components of the pattern.
Transparency of distribution of parallel applications. It must provide the transport of the applications from a system centralized to a distributed system without the user's code is affected. The classes must maintain their properties, independently of the environment of execution of the objects of the applications.
Performance. This is always the most important parameter to consider when one makes a new proposal of development environment for parallel applications. A approach based on patterns as classes and parallel objects must solve the denominated problem PPP (Programmability, Portability, Performance) so that it is considered an excellent approach to the search of solutions to the outlined problems.
The environment of programming oriented to Objects that it has been considered as suitable to cover the 6 previously mentioned properties is the programming language C++, together with the use of the standard POSIX Thread, having as base the operating system Linux, in particular the system Red Hat 7.0.

4. SCIENTIFIC OBJETIVES OF INTEREST

The development of a programming method is based on High Level Parallel Compositions or CPANs that implement a library of classes of utility in the Programming Concurrent/Parallel Oriented to Objects (Rossainz 2005). The method must provide the programmer the commonly used parallel patterns of communication, in such a way that this can exploit the generalization mechanisms for inheritance and parametrization to define new patterns according to the pattern of the CPAN. The specific objectives to reach in this work are:

- To develop a programming method based on High Level Parallel Compositions or CPANs
- To develop a library of classes of Parallel Objects (Rossainz and Capel 2005-2) that provides the user the patterns (under the pattern of the CPAN) more commonly used for the parallel programming.
- To offer this library to the programmer so that, with minimum knowledge of Parallelism and Concurrence, it can exploit them, by means of the use of different reusability mechanisms, under the paradigm of the Orientation to Objects and she/he can, also, to define own patterns, adapted to the communication structure among the processes of their applications.
- Transform known algorithms that solve sequential problems (and that they can be easily parallelizable) in their version parallel/concurrent to prove the methodology and the component software developed work presently.

5. HIGH LEVEL PARALLEL COMPOSITIONS OR CPANS

Some of the problems of the environments of parallel programming it is that of their acceptance for the users, which depends that they can offer complete expressions of the behavior of the parallel programs that are built with this environments (Corradi 1995). At the moment in the systems oriented to objects, the programming environments based on parallel objects are only known by the scientific community dedicated to the study of the Concurrence.
A first approach that tries to attack this problem it is to try to make the user to develop his programs according to a style of sequential programming and, helped of a system or specific environment, this can produce his parallel tally. However, intrinsic implementation difficulties exist to the definition of the formal semantics of the programming languages that impede the automatic parallelization without the user's participation, for what the problem of generating parallelism in an automatic way for a general application continues being open.
A promising approach alternative that is the one that is adopted in the present investigation to reach the outlined objectives, is the denominated structured parallelism. In general the parallel applications follow predetermined patterns of execution. These communication patterns are rarely arbitrary and not structured in their logic (Brinch Hansen 1993). The High Level Parallel Compositions or CPANs are patterns parallel defined and logically structured that, once identified in terms of their components and of their communication, they can be taken to the practice and to be available as abstractions of high level in the user's applications within an environment or programming environment, in this case the one of the orientation to objects. The structures of interconnection of more common processors as the pipelines, the farms and the trees can be built using CPANs, within the environment of work of the Parallel Objects that is the one used to detail the structure of the implementation of a CPAN.

5.1. The Structured Parallelism.

A approach structured for the parallel programming is based on the use of communication/interaction patterns (pipelines, farms, trees, etc.) predefined among the processes of user's application. In such a situation, the approach of the structured parallelism provides the interaction pattern's abstraction and it describes

applications through CPANs able to already implement the patterns mentioned.
The encapsulation of a CPAN should follow the modularity principle and it should provide a base to obtain an effective reusability of the parallel behavior that is implemented. When it is possible to make this, a generic parallel pattern is made, which provides a possible implementation of the interaction among the processes, independent of the functionality of these.
The approach structured for the parallel programming in the last years has followed two ways basically:

1. The enrichment of traditional parallel environments with libraries of "skeletons" (Darlington 1999) of programs that concrete communication patterns represent.
2. The definition of restrictive and closed parallel languages that provide communication in terms of the patterns that are already defined in the system (Bacci 1999).

The approach presented here assists to the first way to consider it more generic and more open. What thinks about now is that, instead of programming a concurrent application from the beginning and of controlling the creation of the processes so much as that of the communications among them, the user simply identifies the CPANs that implement the patterns adapted for the necessities of communication of his application and it uses with the sequential code that implements the computations that individually carry out their processes.
They can be identified of way informal several significant parallel patterns of interconnection and reusable in multiple applications and parallel algorithms, but an agreement doesn't exist the sufficiently general thing that allows to define its semantic ones formally (Corradi 1995). For example, the patron farm is a concept that can be understood by most of its general possible users of a formal one, but its concretion in a particular application she/he forces these to choose among different strategies for its implementation.

5.2. The Object-Oriented.

Sometimes the consent lack in the semantics of the parallel patterns makes that its definition is usually complex and that this is only given at a low level of its implementation; therefore, it is forced the users to go into details of the architecture of the system when they are tried to use the patterns in a concrete program. However, in an environment of development of expandable software, as it is it the one from the object-oriented, the programmer can end up defining any parallel pattern that needs, via a language of high level or graphic tool that it supports the paradigm, adapting it, later on to the characteristics of a concrete application by means of the inheritance mechanisms and genericity. The basic characteristic of these systems is the definition of independent modules of the context that can be connected to each other via channels of communication of high level. The obtaining of parallel compositions represents communication patterns then statically certain and that they can be built independently of the context and in modules reusable, providing this way the encapsulation of the parallel behavior and the capacity of anidation of modules. The basic idea is to define to the CPANs as objects in charge of to control and to coordinate the execution of its components interns. Under this premise you can create an environment of expandable development, based on CPANs that provides characteristic as important as they can be: uniformity, generality and reusability[2].

5.3. Definition of the pattern CPAN.

The basic idea is the one of implementing any type of parallel patterns of communication between the processes of an application or distributed/parallel algorithm as classes, following the paradigm from the Orientation to Objects. Starting from this classes, an object can be instanced and the execution of a method of the object in question you can carry out through a petition of service. A CPAN comes from the composition of a set of objects of three types (Rossainz and Capel 2005-2):
An object manager (Figure.1) that it represents the CPAN in itself and makes of him an encapsulated abstraction that it hides their internal structure. The manager controls the references of a set of objects (a denominated object Collector and several denominated objects Stage) that represent the components of the CPAN and whose execution is carried out in parallel and it should be coordinated by the own manager.

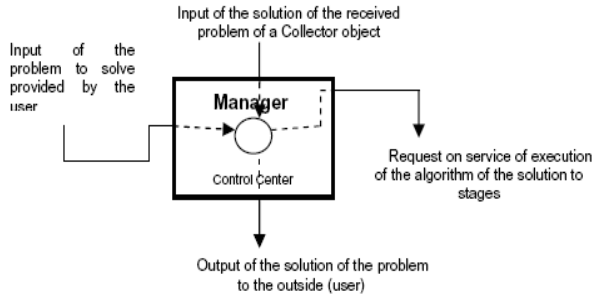

Figure 1: Component MANAGER of model CPAN

The objects Stage (Figure.2) that are objects of specific purpose, in charge of encapsulating an interface type client-server that settles down between the manager and the objects slaves (objects that are not actively participative in the composition of the CPAN, but rather they are considered external entities that contain the sequential algorithm that constitutes the solution of a given problem), as well as providing the necessary connection among them to implement the communication pattern's semantics that seeks to be defined. In other words, each stage should act in parallel as a node of the graph that represents to the pattern. A stage can be directly connected to the manager y/o to

[2] For more details on these characteristics, to see section 2 of the present document

other component stage depending on the pattern peculiar of the implemented CPAN.

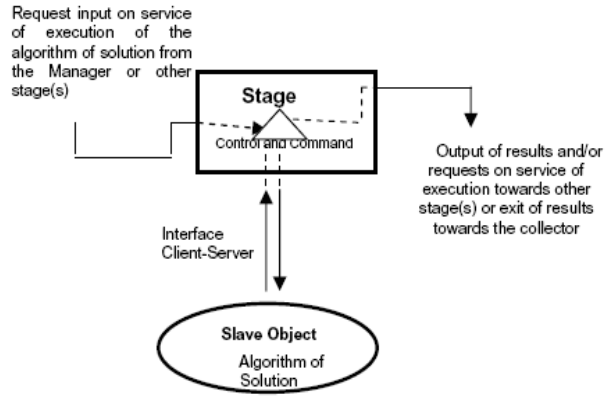

Figure 2: Component Stage of model CPAN and its associated slave object.

And an object Collector (Figure.3) that it is an object in charge of storing in parallel the results that he receives of the objects stage that has connected. That is to say, during the service of a petition, the control flow within the stages of a CPAN depends on the implemented communication pattern. When the composition concludes its execution, the result doesn't return to the manager directly, but rather an instance of the class Collector takes charge of storing this results and of sending them to the manager, which will send to the exterior the results, (that, in turn, send him an object collector), as soon as they go him arriving, without necessity of to wait to that all the results have been obtained.

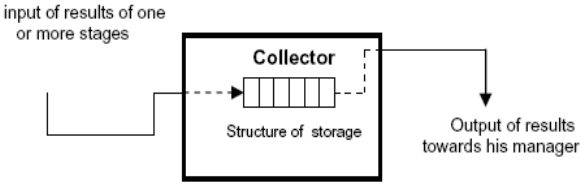

Figure 3: Component Collector of model CPAN

5.3.1. Composition of the CPAN.

If we observe the scheme as a black box, the graphic diagram of the representation of a CPAN would be the one that is shown in Figure.4.

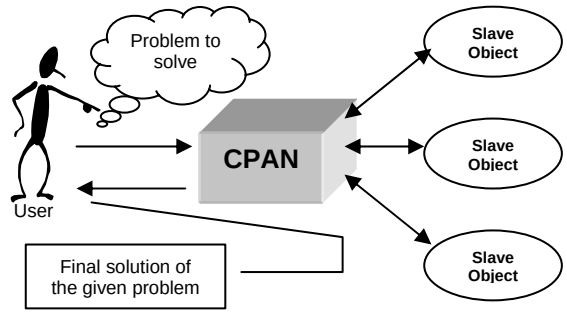

Figure 4: Graphical representation of a CPAN as black box

In summary, a CPAN is composed of an object manager that it represents the CPAN in itself, some objects stage and an object of the class Collector, for each petition that should be treated within the CPAN. Also, for each stage, an object slave will be taken charge of the implementation of the necessary functionalities to solve the sequential version of the problem that you pretend to solve (Figure.5).

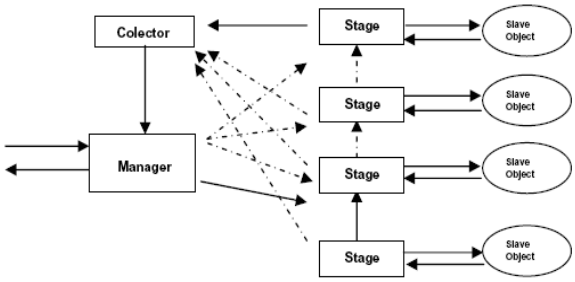

Figure 5: Internal structure of a CPAN. Composition of its components

The Figure.5 shows the pattern CPAN in general, without defining any explicit parallel communication pattern. The box that includes to the components, represents the encapsulated CPAN, the internal boxes represent compound objects (collector, manager and objects stages), as long as the circles are the objects slaves associated to the stages. The continuous lines within the CPAN suppose that at least a connection should exist between the manager and some of the component stage. The same thing happens between the stages and the collector. The dotted lines mean that it can have more than a connection among the components.

5.3.2. The CPAN seen as composition of parallel objects.

The objects manager, collector and stages are included within the definition of Parallel Object (PO) (Corradi 1995).

The Parallel Objects are active objects, that is to say, objects that have execution capacity in themselves. The applications within the pattern PO can exploit the parallelism so much among objects (inter-object) as the internal parallelism of them (intra-object). An object PO has a similar structure to that of an object in Smalltalk[3], but it also includes a politics of scheduling, determined a priori that specifies the form of synchronizing an or more operations of a class in parallel. The synchronization policies are expressed in terms of restrictions; for example, the mutual exclusion in processes readers/writers or the maximum parallelism in processes writers. All the parallel objects derive then of the classic definition of “class” more the incorporation of the synchronization restrictions (mutual exclusion and maximum parallelism). The objects of oneself class shares the same specification contained in the class of

[3] An Object in Smalltalk as in C++ is constituted up of a state and a behavior.

which you/they are instantiates. The inheritance allows deriving a new specification of one that already exists. The parallel objects support multiple inheritances.

5.3.2.1 Types of communication in the parallel objects.

The parallel objects define 3 communication ways: the synchronous communication way, the asynchronous communication way and the asynchronous future communication way.

1. **The synchronous way** stops the client's activity until the object active server gives him the answer. The notation[4]: `ref_obj.name_meth([lista_param])` it facilitates their use in the programming of applications.
2. **The asynchronous way** doesn't force the wait in the client's activity; the client simply sends the petition to the object active server and her execution continues. Its use in the programming of applications is also easy, because it is only necessary to create a thread and to throw it for its execution[5]. We will use the following notation to refer to this communication way: `Thread ref_obj.name_meth([lista_param]);` where `Thread` is thrown to execute the method `name_meth([lista_param])` of an object `ref_obj`.
3. **The asynchronous future way** makes only wait the client's activity when, within its code, the result of the method is needed to evaluate an expression. Their use is also simple, although its implementation requires of a special care to get a constructo with the wanted semantics. The notation used for it will be it: `FutureType futureVar = ref_Obj.name([lista_param])` that expresses the generation and future assignment of the result of a function invoked through a reference to an object. Where `FutureType` is the type that defines the future and `Anytype ResulVar = ref_Obj.futureVar`; it is used for the conversion of type of the future that returns the function when it is executed to a type `AnyType`. The word ANYTYPE is used to suggest the use of "any type", the one that is of interest for the user.

The asynchronous and asynchronous future communication ways carry out the parallelism inter-objects executing the objects client and server at the same time.

5.3.3. Definition of the classes bases of a CPAN anyone.

As it has already been described, a CPAN comes from the composition of a set of objects of three types. In particular, each CPAN this compound for an object manager, some objects stage and an object collector for each petition carried out by the objects clients of the CPAN. Also, for each stage of the CPAN, an object slave will be taken charge of the implementation of the sequential part of the computation that is sought to carry out in the application or in the distributed and parallel algorithm. In PO the necessary basic classes to define objects manager, collector, stages and to compose a CPAN are:

- the abstract **class ComponentManager**
- the abstract **class ComponentStage**
- the concrete **class ComponentCollector**

An instance PO of a concrete class derived of the class ComponentManager represents a CPAN within an application (called manager) programmed according to the pattern of parallel object. The instances (called stages) of a concrete class derived of the class ComponentStage is connected to each other, to implement the stages composition. Each stage commands the execution of an object PO, called slave (slave) that is controlled by the own stage.

The creation of the stages and of the collectors and their later interactions are managed transparently to the code of the application by the manager. From the point of view of an user already interested in reuse the parallel behavior defined in some classes CPAN, the class of interest will be that of the manager. When an user is interested in using a CPAN within an application, he has to create an instance of a class manager specific, it is, one that implements the parallel behavior needed by the application and that it initializes it with the reference to the objects slaves that will be controlled by each stage and the name of the requested method. The following syntactic definitions have been written using the grammar free of context that is in the appendix A of the present document.

5.3.4. The Synchronization restrictions MaxPar, Mutex y Sync:

It is necessary having synchronization mechanisms, when parallel petitions of service take place in a CPAN, so that the objects that conform it can negotiate several execution flows concurrently and, at the same time, guarantee the consistency in the data that are processing. Within any CPAN the restrictions MAXPAR (The maximum parallelism or MaxPar is the maximum number of processes that you/they can be executed at the same time), MUTEX (The restriction of synchronization mutex carries out a mutual exclusion among processes that want to consent to a shared object. The mutex preserves critical sections of code and obtains exclusive access to the resources) and SYNC (The restriction SYNC is not more than a synchronization of the type producer/consumer of utility) can be used for the correct programming of their methods.

---

[4] The notations used in this section are based on the grammar of Parallel Objects described in appendix A.

[5] The POSIX Thread provides the instruction *pthread_create(. . .)* along with the type *pthread_t* for the creation and use of threads.

6. DESIGN AND CONSTRUCTION OF THE CPANS FARM, PIPE Y TREEDV

With the basic set of classes of the model of programming of PO they are possible to be constructed concrete CPANs. To build a CPAN, first it should be had clear the parallel behavior that one needs to implement, so that the CPAN in itself is this pattern. Several parallel patterns of interaction exist as are the farms, the pipes, the trees, the cubes, the meshes, the matrix of processes, etc.

Once identified the parallel behavior, the second step consists on elaborating a graphic of its representation as mere technique of informal design of what will be later on the parallel processing of the objective system; it is also good to illustrate its general characteristics, etc., and it will allow later to define its representation with CPANs, following the pattern proposed in the previous section.

When the model of a CPAN is already had concretized, that defines a specific parallel pattern; say for example, a tree, or some of those previously mentioned ones, the following step would be to carry out its syntactic definition and semantics.

Finally, the syntactic definition previous to a CPAN programmed is translated in the most appropriate programming environment for its parallel implementation. It would be verified that the resulting semantics is the correct one, it would be proven with several different examples to demonstrate its genericity and the performance of the applications would be observed that include it as a component software.

The parallel patterns worked in the present investigation have been the pipeline, the farm and the treeDV[6] to be a significant set of reusable patterns in multiple applications and algorithms. Being used at the moment with different purposes, in different areas and with different applications according to the literature that there is on the topic.

1. **The pipeline**, this compound for a set of interconnected states one after another. The information follows a flow from a state to another.
2. **The farm**, is composed of a set of worker processes and a controller. The workers are executed in parallel until reaching a common objective. The controller is the one in charge of distributing the work and of controlling the progress of the global calculation.
3. **In the treeDV**, the information flows from the root toward the leaves or vice versa. The nodes that are in the same level in the tree are executed in parallel making use of the denominated technique of design of algorithms it Divide and Conquer for the solution of the problem.

These parallel patterns conform the library of classes proposed within the pattern of the CPAN.

[6] The pattern treeDV implements the paradigm of programming of divide and conquer by means of the use of binary trees.

6.1 The Cpan PipeLine.

It is presented the technique of the parallel processing of the pipeline as a High Level Parallel Composition or CPAN, applicable to a wide range of problems that you/they are partially sequential in their nature. The CPAN Pipe guarantees the parallelization of sequential code using the patron PipeLine.

The Figure.6 represent the parallel pattern of communication Pipeline as a CPAN.

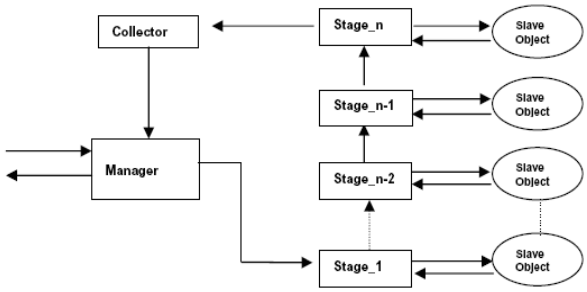

Figure 6: The CPAN of a Pipeline

The objects *stage_i* and *Manager* of the graphic pattern of the CpanPipe are instances of concrete classes that inherit the characteristics of the classes ComponentManager and ComponentStage.

6.2 The CPAN Farm.

It is shown the technique of the parallel processing of the FARM as a High Level Parallel Composition or CPAN.

The representation of parallel pattern FARM as a CPAN is show in Figure. 7.

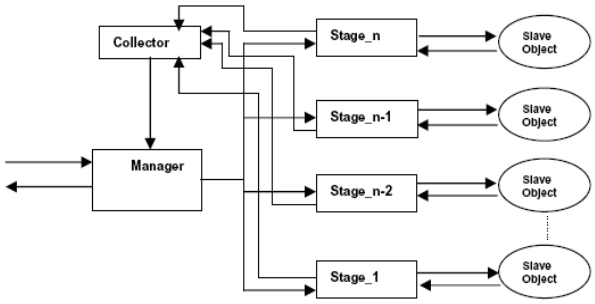

Figure 7: The Cpan of a Farm

The same as in the previous pattern, the objects Manager and *stage_i* are respectively instances of the classes that inherit of the classes base denominated ComponenManager and ComponentStage.

6.3 The Cpan TreeDV.

Finally, the programming technique is presented it Divide and Conquer as a CPAN, applicable to a wide range of problems that can be parallelizable within this scheme.

The representation of the patron tree that defines the technique of it Divide and Conquer as CPAN has their model represented in Figure. 8.

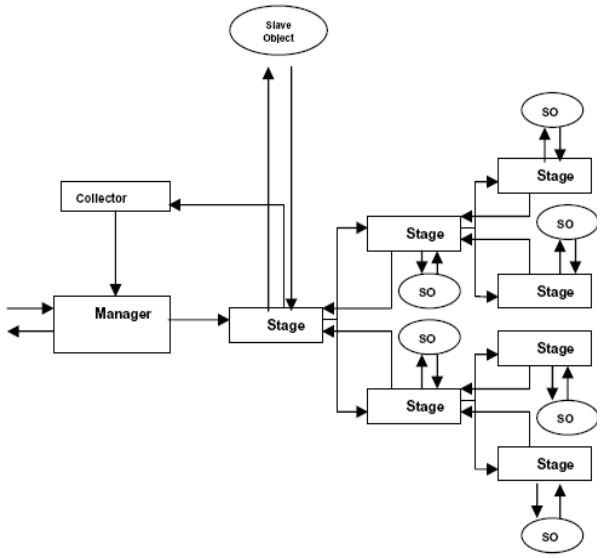

Figure 8: The Cpan of a TreeDV

Contrary to the previous models, where the objects slaves were predetermined outside of the pattern CPAN, in this model an object slave is only predefined statically and associated to the first stage of the tree. The following objects slaves will be created internally by the own stages in a dynamic way, because the levels of the tree depend from the problem to solve and a priori the number of nodes that can have the tree is not known, neither its level of depth.

These constitute a significant set of reusable communication patterns in multiple parallel applications and algorithms. See (Capel and Rossainz 2004; Rossainz 2005) for details.

7. USE OF A CPAN WITHIN AN APPLICATION

Once implemented the CPANs of interest, the way in that you/they are used in user's application is the following one:

1. It will be necessary to create an instance of the class manager of interest, that is to say, one that implements the required parallel behavior in agreement with the following steps:
   1.1. To initialize the instance with the reference to the objects slaves that will be controlled by each stage and the name of the method requested as an association of even (slave_obj, associated_method).
   1.2. The internal stages is created (using the operation *init()*) and they are passed each one an association (slave_obj, associated_method) that will use invoking the associated_method on their slave object.
2. The user asks the manager to begin a calculation through the execution within the CPAN of the method execution(). This execution is carried out as it continues:
   2.1. The object collector is created with respect to the petition.
   2.2. They are passed to the stages the input data (without verification of types) and the reference to the collector.
   2.3. The results are obtained from the object collector.
   2.4. The collector returns the results again to the exterior without verification of types.
3. An object manager has been created and initialized and some execution petitions can be dispatched in parallel.

8. RESULTS OBTAINED

Some CPANs adapt better to the communication structure of a given algorithm than others, therefore yielding different speedups of the whole parallel application. The way in which it must be used to build a complete parallel application is detailed below.

1. It is necessary to create an instance of the adequate class manager, that is to say, a specialized instance (this involves the use of inheritance and generic instantiation) implementing the required parallel behavior of the final manager object. This is performed by following the steps:
   1.1. Instance initialization from the class manager, including the information, given as associations of pairs (slave_obj, associated_method); the first element is a reference to the slave object being controlled by each stage and the second one is the name of its callable method.
   1.2. The internal stages are created (by using the operation *init()*) and, for each one, the association (slave_obj, associated_method) is passed to. The second element is needed to invoke the associated_method on the slave object.
2. The user asks the manager to start a calculation by invoking the *execution ()* method of a given CPAN. This execution is carried out as it follows:
   2.1. a collector object is created for satisfying this petition;
   2.2. input data are passed to the stages (without any verification of types) and a reference to the collector;
   2.3. results are obtained from the object collector;
   2.4. The collector returns the results to the exterior without type verification.
3. An object manager will have been created and initialized and some execution petitions can then start to be dispatched in parallel.

We carried out a Speedup analysis of the Farm, Pipe and TreeDV CPANs for several algorithms in an Origin 2000 Silicon Graphics Parallel System (with 64 processors) located at the European Center for Parallelism in Barcelona (Spain) this analysis is discussed below.

Assuming that we want to sort an array of data, some CPANs will adapt better to communication structure of a Quicksort algorithm than others. These different parallel implementations of the same sequential algorithm will therefore yield different speedups. The program is structured of six set of classes instantiated

from the CPANs in the library High Level Parallel Compositions, which constitute the implementation of the parallel patterns named Farm, Pipe and TreeDV. The sets of classes are listed below:

1. *The set of the classes base*, necessary to build a given CPAN.
2. The set of the classes that define the abstract data types needed in the sorting.
3. *The set of classes that define the slave objects*, which will be generically instantiated before being used by the CPANs.
4. The set of classes that define the Cpan Farm.
5. The set of classes that define the Cpan Pipe.
6. The set of classes that define the Cpan TreeDV.

This analysis of speedup of the CPANs appears in Figures 9, 10 and 11. In all cases the implementation and test of the CPANs Farm, Pipe and TreeDV 50000 integer numbers were randomly generated to load each CPAN.

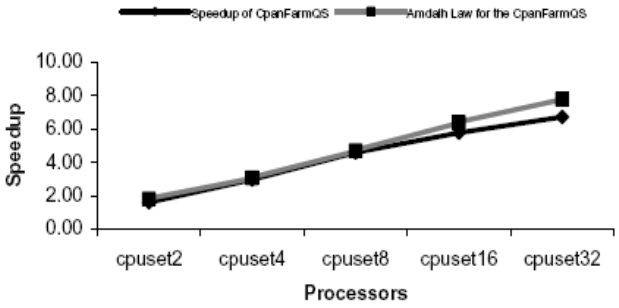

Figure 9: Scalability of the Speedup found for the CpanFarm in 2, 4, 8, 16 and 32 processors

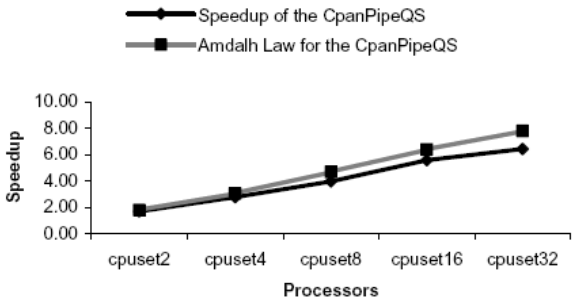

Figure 10: Scalability of the Speedup found for the CpanPipe in 2, 4, 8, 16 and 32 processors

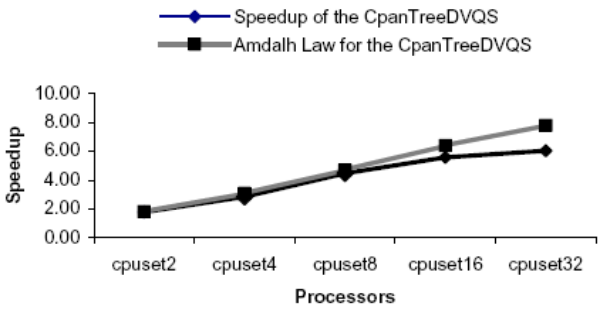

Figure 11: Scalability of the Speedup found for the CpanTreeDV in 2, 4, 8, 16 and 32 processors

9. CONCLUSIONS

1. A method of original programming has been developed based on High Level Parallel Compositions or CPANs
2. Patterns of communication/interaction have implemented themselves within the model of the CPAN commonly used in the parallel and distributed programming: the Cpan Pipe, the Cpan Farm and the Cpan TreeDV.
3. The implemented CPANs can be exploited, thanks to the adoption of the approach oriented to objects using the different mechanisms of reusability of the paradigm to define new patterns already using those built.
4. Well-known algorithms that solve sequential problems in algorithms parallelizable have transformed and with them the utility of the method has been proven and of the component software developed in the investigation.
5. The CPANs Pipe, Farm and TreeDV conform the beginning of the library of classes that intends in this work.
6. The restrictions of synchronization have been programmed of original form suggested by the model of the CPAN for their parallel and concurrent operation: the maximum parallelism (MaxPar), the mutual exclusion (Mutex) and the synchronization of communication of processes readers/writers (Sync).
7. Of equal it forms the programming in the asynchronous future communication way for results “futures” within the Cpans it has been carried out in an original way by means of classes.

REFERENCES

Bacci, Danelutto, Pelagatti, Vaneschi, 1999, SklE: A Heterogeneous Environment for HPC Applications. *Parallel Computing*, Volume 25, pp. 1827-52.

Brassard, G., Bratley, P., 1997, *Fundamentos de Algoritmia*, Spain, Prentice-Hall.

Brinch Hansen, 1993, Model Programs for Computational Science: A programming methodology for multicomputers. *Concurrency: Practice and Experience*. Volume 5, Number 5, 407-423.

Brinch Hansen, 1994, SuperPascal- a publication language for parallel scientific computing.

Concurrency: Practice and Experience, Volume 6, Number 5, 461-483.

Capel, M., Troya, J., M., 1994, An Object-Based Tool and Methodological Approach for Distributed Programming. *Software Concepts and Tools*, Volume 15, pp. 177-195.

Corradi, A., Leonardi, L., 1991, PO Constraints as tools to synchronize active objects. *Journal Object Oriented Programming*, Volume 10, pp. 42-53.

Corradi, A., Leonardo, L., Zambonelli, F., 1995, *Experiences toward an Object-Oriented Approach to Structured Parallel Programming*. Technical report no. DEIS-LIA-95-007. Italy.

Danelutto, M., Orlando, S., et al. *Parallel Programming Models Based on Restricted Computation Structure Approach*. Technical Report. Universitá de Pisa.

Darlington et al, 1993, Parallel Programming Using Skeleton Functions. *PARLE'93*, Munich.

Hartley, Stephen J., 1998, *Concurrent Programming. The JAVA Programming Language*. New York, Oxford University Press.

Lavander, Greg R., Kafura, Dennis G., *A Polymorphic Future and First-class Function Type for Concurrent Object-Oriented Programming*. Journal of Object-Oriented Systems. Available from: http://www.cs.utewxas.edu-users/lavender/papers/futures.pdf.

Roosta, Séller, 1999, Parallel Processing and Parallel Algorithms. Theory and Computation. *Springer*.

Rossainz, M., 1999, *Una Metodología de Programación Paralela en Java*. Technical Report. Universidad de Granada.

Capel, M., Rossainz, M., 2004. A parallel programming methodology based on high level parallel compositions. *14th International Conference on Electronics, Communications and Computers IEEE CS press*. México.

Rossainz, M., 2005, *Una Metodología de Programación Basada en Composiciones Paralelas de Alto Nivel (CPANs)*, PhD dissertation, Universidad de Granada.

Rossainz, M., Capel, M., 2005-2, An Approach to Structured Parallel Programming Based on a Composition of Parallel Objects. *XVI Jornadas de Paralelismo*, Granada, Spain, Thomson.

AUTHORS BIOGRAPHY

MARIO ROSSAINZ LOPEZ was born in Puebla, México and went to the University of Puebla, where he studied Sciences of Computation and obtained his degree in 1994. He works in the Faculty of Sciences of Computation of the University of Puebla from the year of 1995 where he is now in the research group of Software Engineering in the field of distributed systems. His e-mail address is: `mariorl@siu.buap.mx` and his Web-page can be found at `http://www.cs.buap.mx/~mrossainz`.

MANUEL I. CAPEL TUÑON was born in Spain and went to the University of Granada, where he studied Physics and obtained the MSC degree in 1982. He worked in the University of Murcia before moving in 1989 to the University of Granada where he is now leading a research group in the field of Concurrent Systems. His e-mail address is: `mcapel@ugr.es` and his Web-page can be found at `http://lsi.ugr.es/~mcapel`.