Improving POP-C++ for HPC

diploma project

Barras Frédéric

Responsible interns :	Pierre Kuonen François Kilchoer Jean-François Roche Guilherme Peretti Pezzi
Responsible externs :	Barney Maccabe (UNM) Rolf Riesen (Sandia Labs) Tuan Anh Nguyen (HCMUT)
Expert :	Peter Kropf (UNI-NE)
Student :	Frédéric Barras
Date :	09/06/2007 to 11/14/2007

SUMMARY

General Introduction

1. Introduction {common}

In this final project, four different tasks figure in our goal specification. The work has to be split in two distinct projects between the two students. The goal of this project is to improve the language POP-C++ for **H**igh **P**erformance **C**omputing (HPC). For more details, please refer to section *1.2 Objectives.* This project lasts 9 weeks, and is realized at the *University Of New Mexico (UNM)* in Albuquerque, USA.

1.1.Technologies used

This is a brief description of the different technologies used during the project.

1.1.1. Ubuntu

Ubuntu is a distribution of the free operation system Linux and is based on *Debian Linux*. We use the version 6.10, called *Edgy Eft*, on our notebooks. At this time (September 2007), the most recent version is 7.04. This version is actually not compatible with our notebooks. After research on the internet, it seems that it is a common problem that *Ubuntu 7.04* has difficulties with hardware detection.

1.1.2. POP-C++

POP-C++ is a programming language which is derived from C++. It has been developed by the GridGroup at the EIA-FR. Its main objective is to allow programmers to write object oriented applications which are able to run objects on different workstations connected by a network. These objects are distributed automatically during the execution of an application. At the time of this project, POP-C++ is running only on workstations with a Unix/Linux operating system. The POP-C++ environment includes two main parts. First of all, a precompiler which is able to parse and read a POP-C++ code to generate pure C++ code. This code is now ready to be compiled with a standard C++ compiler. Secondly, the runtime: an environment which is necessary to run the parallel distributed objects.

1.1.3. MPI

MPI (Message passing interface) is a standard, describing message passing in parallel computing on a distributed computer system. The programming interface specifies a pool of operations and their semantic.

The implementation of MPI used during this project is *MPICH2* which implements the MPI-2.1 standard. It is delivered with C/C++ and Fortran 77 resp. Fortran 90 compilers. The programmer can choose to write programs in one of these languages. Thorough this document when talking about MPI, it always means this standard and this implementation.

1.1.4. Computer cluster / GRID

A computer cluster is a group of connected computers working together. They are commonly connected to each other by a fast local area network, allowing them to share tasks though they can be viewed as a single computer. Clusters are used for different tasks (e.g. Load-balancing). We will use a cluster for high performance computing. Programs which are designed to run on such a cluster (e.g. MPI-programs) split the main problem in different smaller tasks to distribute them on different nodes (processors) on the cluster. These tasks are computed in parallel which increases the performance.

The main differences between a cluster and a GRID:

- GRID computing is dedicated to work on computers that can be geographically separated
- Computers on a GRID are quite autonomous and are not dedicated to accomplish only subtasks on this net

1.2.Objectives

This project will help improve the POP-C++ programming language in high performance computing. The first step will be the comparison with the MPI programming language. This will help us to find the weak points of POP-C++. The second step is to add global communication to POP-C++. Global communication makes it possible for remote objects to communicate by broadcasting (point-to-multipoint message passing). The next step will be the development of a convenient method to create arrays of parallel objects using different personalized constructors. Until now, only the default constructor is used. The last step in the project is the validation of the asynchronous object creation in the most recent version of POP-C++.

In our planning, we must include the details about the two first parts. If we manage to finish them before the end of the project, the rest of the time is used for the two last tasks.

1.3.Presentation of the document

This document is divided in two different sub reports, one for each task we have to achieve. The first part is about the comparison between POP-C++ and MPI (starting at page 9), and the second is about collective methods in POP-C++ (starting at page 35).

Each common part contains *{common}* in its title.

- **General introduction:**	Explanation of the objectives of this project and different technologies
- **Part 1, Comparison between POP-C++ and MPI:**	First sub report
○ **Introduction:**	Explanation of the subject
○ **Analysis:**	Analysis of the initialization of POP-C++ and MPI
○ **Implementation:**	Method used to measure the initialization
○ **Test:**	Prediction model and result of the tests on the cluster, comparison
○ **Conclusion:**	Conclusion about the measurements, explanation about the differences
- **Part2 , collective communications in POP-C++:**	Second sub report
○ **Introduction:**	Explanation of the subject
○ **Analysis:**	Analysis of the possibility to add collective

	communication to POP-C++ without changing the parser
○ **Conception:**	Explanation about the way to use reflection and the functionment of the collective methods
○ **Implementation:**	Current state of the project, explanation of the implemented parts
○ **Conclusion:**	Conclusion about the library, encountered problems
- **General conclusion:**	Personal conclusion, planning discussion, thanks
- **Appendix:**	Figure table, links, source codes.

1.4.Deliverables

At the end of our project, we will provide a CD with all documentation, including:

- a slide-show
- external project source
- source code
- the report
- the website

During this project, all documentation is stored on our website [8].

Comparison between POP-C++ and MPI

1. Introduction {common}

In this part of the project, we have to compare POP-C++ to another distributed programming interface called MPI implemented in the library OpenMPI. We have to define a test scenario to measure the differences between POP-C++ and MPI in terms of performance. To do this, we will have to write equal programs in both languages and run them on a cluster under identical conditions. The result will permit us to determine which language is slower in which part of the program and where their weaknesses are. This kind of comparison is called **Benchmarking**.

2. Analysis

2.1.POP-C++ Runtime {common}

This is only a short description of the POP-C++ runtime. Complete description is available in: "WSDL with POP-C++". [2]

The POP-C++ Runtime contains the following classes:

a) *Interfaces* : the local representation of a remote object
b) *Brokers* : make the translation between network messages and method call on the remote object
c) *Comboxes* : contain the socket for the network communication
d) *JobManagers* : manage the resources and placement of remote objects
e) *Buffers* : pack/unpack the data which have to be sent/received
f) *Objects*: are the real remote object, always bound with a buffer on the remote side, and an interface on the local side.

Every remote object has an interface on the local runtime, which has the same methods with the same signatures. Seen from the local runtime, calling a method on a remote object or on a local object makes no difference. When the interface receives the method call, it sends this call, using the combox and the buffer, through the network to the remote object. The broker attached to the remote object receives the packet, and translates it into a method call for the object. It then sends back the result, if necessary.

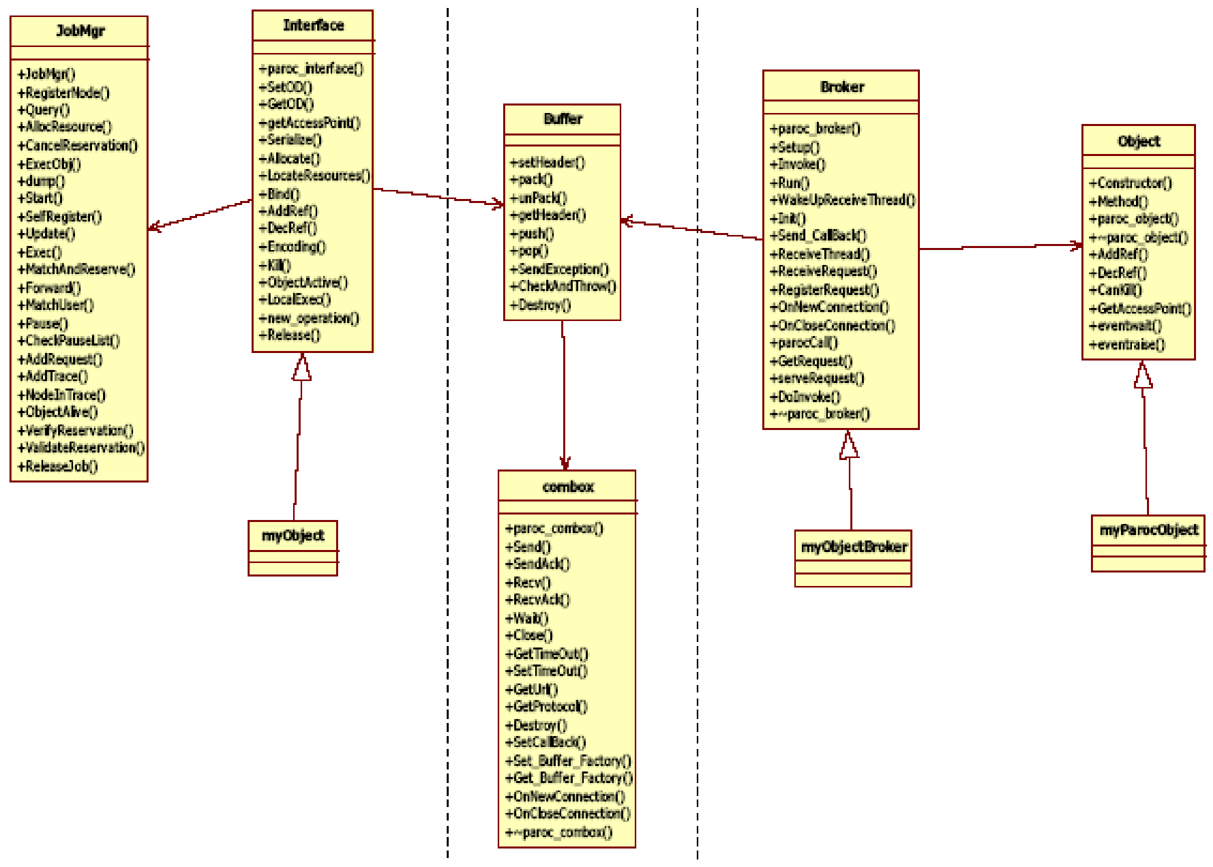


Figure 1 – POP-C++ Class Diagram

This diagram shows the different classes of the POP-C++ runtime, on left, are the classes used exclusively on the local side, on the right the classes used by the remote side, and in the middle, the classes used by both local and remote part. *MyObject* is the object which has to be used as remote, it also inherits from *Interface*. *MyObjectBroker* and *MyParocObject* are the remote part of our object. The real implementation of the object is in *MyParocObject.*

2.2.POP-C++ Object creation

This is only a short description of the POP-C++ Object creation. A complete description is available in: "WSDL with POP-C++". [2]

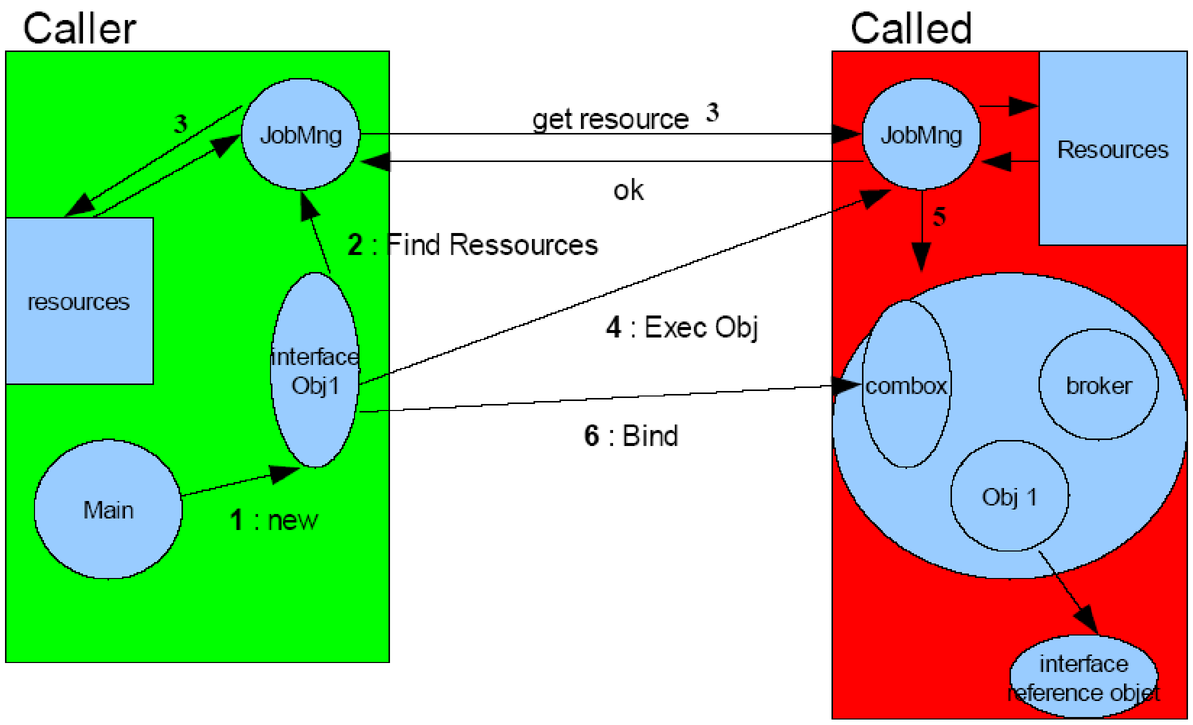


Figure 2 – Object creation in POP-C++

When a remote object is created, the following steps are made:

1) The Main creates a new interface for the desired object.
2) The interface creates its buffer and combox, and then will search for a machine for the object, depending on the parameters provided with the Object Descriptor (the methods added after the object constructor, for example *Integer(int power)@{od.power(power);}*; The JobManager will try to find resources for those requirements.
3) The Job Manager will first ask the local POPC++ runtime, to see if it has the required resources available, and, when the local system is not able to host the object, will seek on other machines if the resources are available.
4) The JobManager returns the reference of the remote machine to the Interface which uses a system call to start the executable of the object on the remote machine.
5) The remote JobManager will get the executable of the object from the described location. It contains a combox, the object and its broker.
6) The interface is bound, through the comboxes, with the remote object (connect sockets), and can then call methods on the remote object. The first method called is always the constructor call.

If the job manager is not used, and the address from the remote host is determined by the programmer, the interface itself will start the object on the remote side, and then bind it.

The method call of the constructor could already be taken as a method call, and is not really a part of the object creation.

2.3.The MPI standard {common}

MPI uses the message passing paradigm to communicate in a distributed memory system (Figure 3 - Distributed Memory Sytem[2]). Each memory entity is accessible only via its corresponding processor (CPU). To communicate, processors use a network which connects them to each other. Messages can be sent and received in different manners. For example, it can be passed in a blocking way, which is very safe but slow, or in a non-blocking way, which can increase the performance of a program by eliminating active waiting of the CPU.

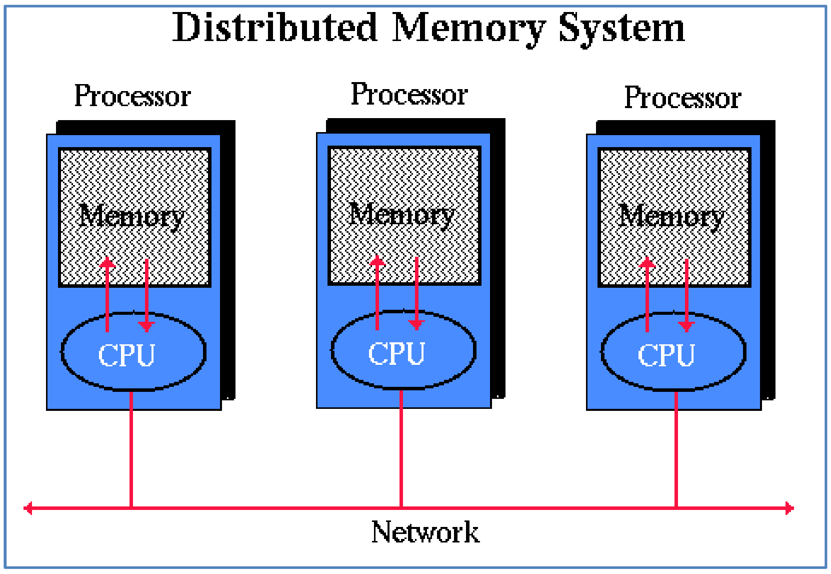


Figure 3 - Distributed Memory Sytem[2]

Message passing is used to exchange data between processes. Process A **sends** a memory buffer (data) of its application memory through the network to process B, which **receives** the message and stores it into his application memory.

2.3.1. MPI-processes

It is important to understand the difference between a **process** (set of executable instructions) and a **processor** (hardware)! More than one process can execute on the same processor but will nevertheless communicate via message passing. MPI differentiates processes by their **rank**. Every process is identified by a number going from 0 to N-1, where N is the number of processes running.

Every MPI-programmer has to be aware about issues related to concurrent computing. In an MPI-application, there is usually more than one process running at the same time. They communicate between each others in either a blocking or non blocking way. In case of inappropriate use of the message passing routines provided by MPI, unexpected consequences may show up (e.g. Dead-lock or data loss). Different **communication events** can take place during message passing:

- copying a message from application memory to system memory (send buffer)
- arrival of a message

2.3.2. SPMD

Programming in MPI applies the SPMD (**S**ingle **P**rogram, **M**ultiple **D**ata) mechanism. This means that the same program is running everywhere. To avoid that every process executes exactly the same instructions, they have to be differentiated in the program by their rank. A model which is very often used is to differentiate only process 0 from the rest of the processes.

```
if rank==0
      send(message)
else
      receive(message)
```

2.4. MPI Initialization

This function must be called before any other MPI routine is called. It initializes the MPI environment. The C version of the function requires the *argc* and *argv* arguments provided to `main()`. This function initializes the system able to create MPI error messages.

When a process calls *MPI_Init()*, the following operations occur:

- *MPI_Init()* opens a local socket with a port number, and sends this information to the parallel operating environment (POE).
- It receives then from the POE the list of addresses and ports already registered, and establishes a connection with each of them.

2.4.1. MPI_COMM_SPAWN

```
int MPI_Comm_spawn(char *command, char *argv[], int maxprocs,
    MPI_Info info, int root, MPI_Comm comm,
    MPI_Comm *intercomm, int array_of_errcodes[])
```

MPI_COMM_SPAWN is described in version 2.0 of MPI. This command permits the user to dynamically generate processes after the *MPI_Init()* has been called. Here is a description of the parameters of *MPI_Comm_Spawn()*:

Command :	Name of program to be spawned; must be an MPI program, which calls *MPI_Init().*
Argv :	Arguments given to the command
Maxprocs :	Maximum number of processes to start. When less than *Maxprocs* processes are started, an error code is returned
Info :	set of key-value pairs telling the runtime system where and how to start the processes
Root :	Rank of the process which will spawn these new processes
Comm :	Intracommunicator which will contain these new processes. (For example, we can create a MPI_COMM_NEWWORLD)
Intercom :	Output parameter. Intercommunicator between the Intracommunicator *Comm* and the one from the current process
Array_of_errcodes :	Output parameter. One code per process created. Indicate whether the spawn is successful or not for each process.

The spawned processes are referred to as children. They have their own MPI_COMM_WORLD, different from their parents'. *MPI_Comm_spawn()* doesn't return until all child have called *MPI_Init*. And *MPI_Init* in the children does not return until all the parents have called *MPI_Comm_spawn()*. Once all have made their calls, they receive the *Intercommunicator*, to communicate between children and parent processes.

2.5.Scenario Test

I will compare the creation of objects on POP-C++ and the distribution of processes in MPI. The programs will not be complicated, they will just create processes/objects, and measure the time used to do it.

The following parameters will change during the test:

- Number of nodes: 1,2,4,6,8,10,12,14,16.
- Number of objects/processes created: 1 to 16, one per node each time.

The size of the data in the objects / processes will not change during this test; we don't want to test data sending between the local object /first process and their children.

I will use the following programs

- A POP-C++ program which creates objects without JobManager
- A POP-C++ program which creates objects with JobManager
- An MPI program which is launched with a parameter to determine the number of processes
- An MPI program which dynamically spawns processes on the different nodes

2.5.1. POP-C++ scenario test

Two implementations of POP-C++ are tested, one with the JobManager, and one with a static URL in the constructor parameters. It will help me to determine the weight of the JobManager in the POP-C++ runtime by comparing both programs. The version without Job Manager will also be used to be compared with MPI. Different timestamps are used during the object creation of POP-C++.

Here are some light versions of the sequence diagram from the creation of an object, with and without the Job manager. The local object is in **green**, the remote one in **red**, and the timestamps in **blue.**

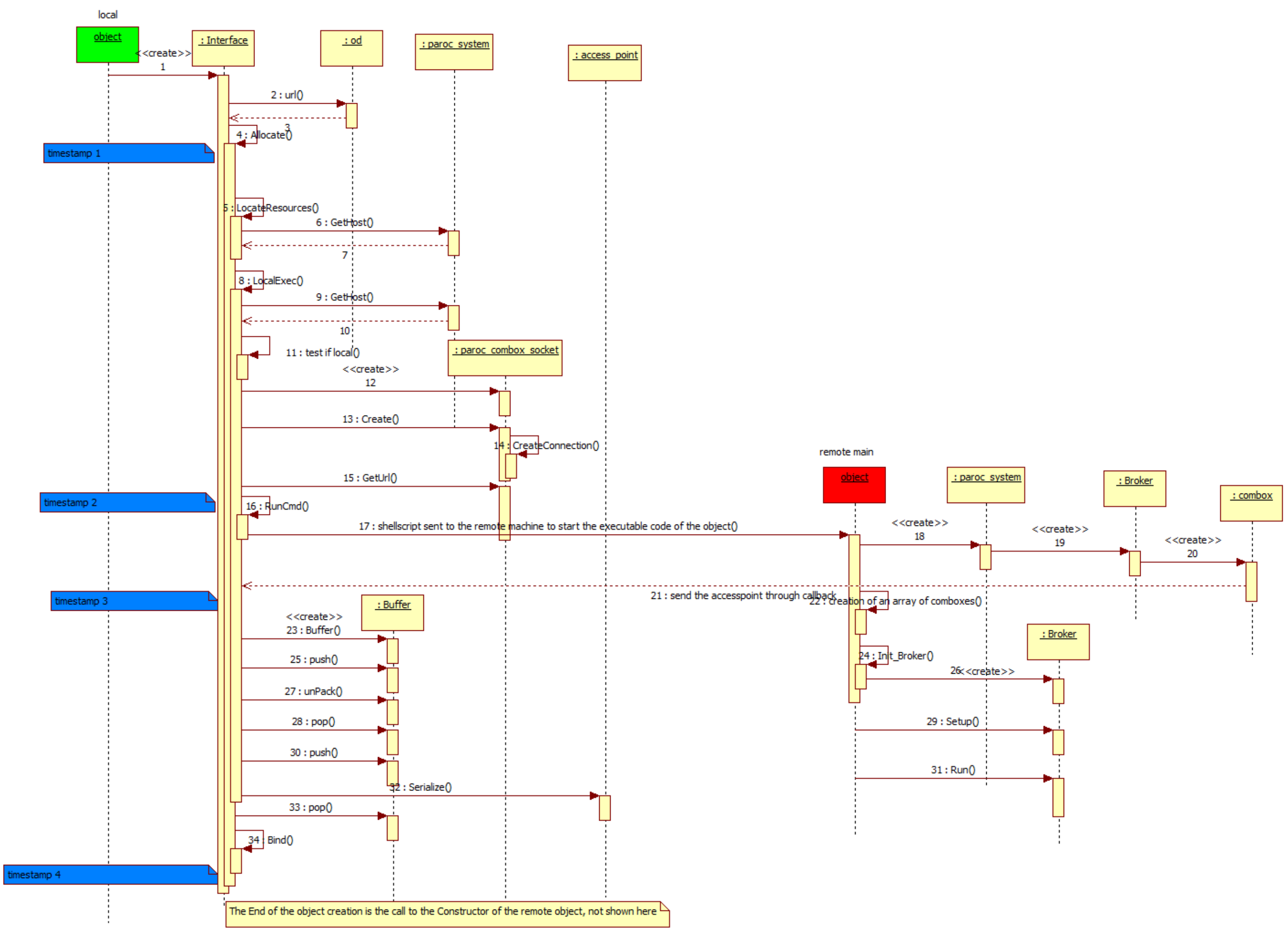


Figure 4 - simplified sequence diagram of object creation without Job Manager

Here are the positions of the timestamps:

- *Timestamp 1:* at the beginning of the Allocate method, will be considered as time t0=0.
- Timestamp 2: End of the local first-part initialization, just before sending the command to the remote node.
- *Timestamp* 3: Just after having received the response from the remote node, that the necessary objects were created.
- *Timestamp 4:* End of the Allocate method, just before sending the *_Constructor* method message.

This is the simplified diagram of the object creation with the JobManager.

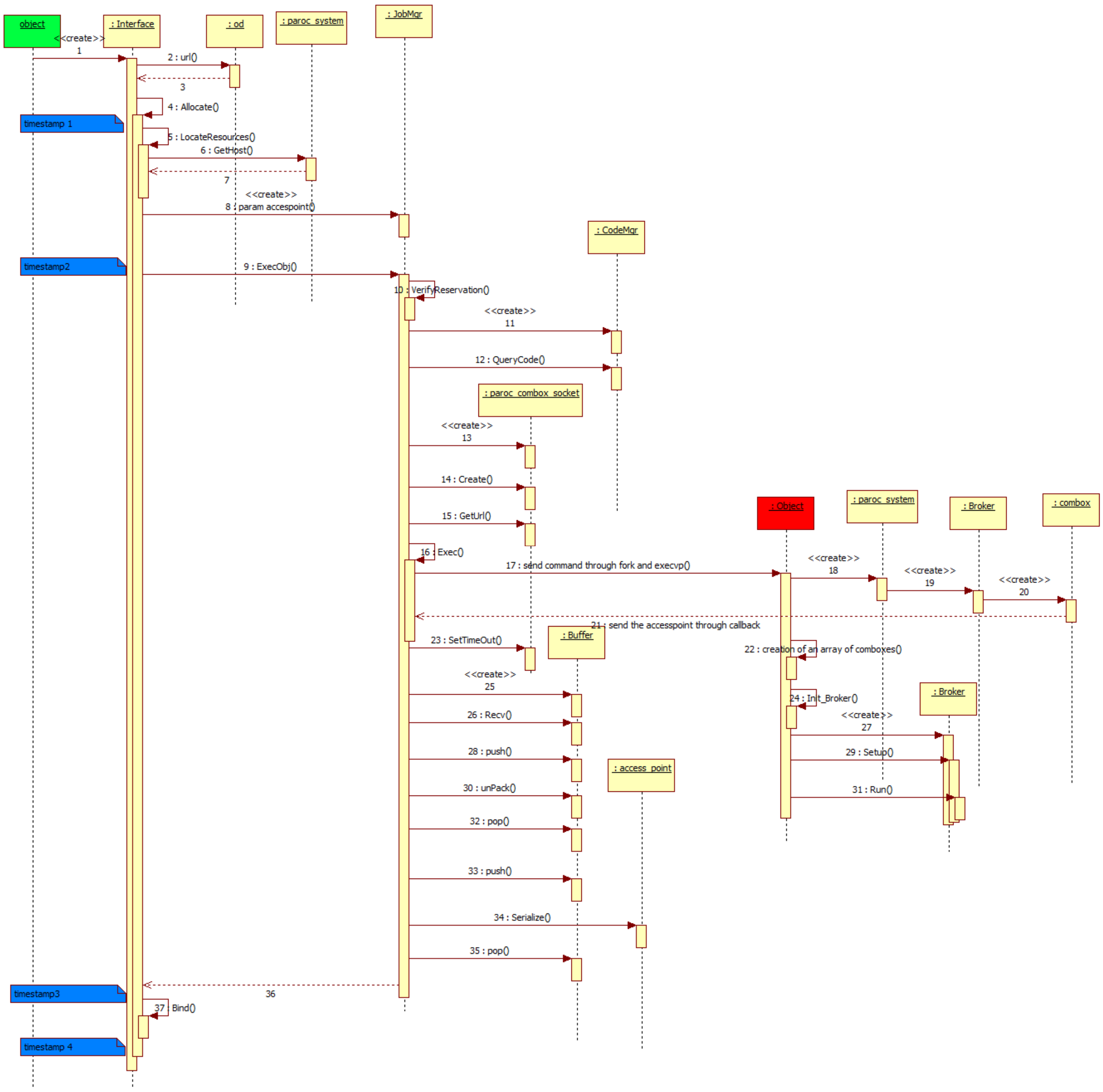


Figure 5 - simplified sequence diagram of object creation with Job Manager

Here are the positions of the timestamps:

- *Timestamp 1:* at the beginning of the Allocate method, will be considered as time t0=0.
- Timestamp 2: Begin of the *ExecOjb* method in the JobManager, where it is doing its job: find a place which has the right properties to create the object.
- *Timestamp* 3: End of the *ExecOjb* method, to determine the time used by the JobManager.

- *Timestamp 4:* End of the Allocate method, just before sending the *_Constructor* method message.

I need to have the timestamp 1 and 4 in the same method, to allow me to compare their values. The biggest usable method was *Allocate()* in Interface.cc.

The biggest difference between these two versions of the POP-C++ program will be at timestamp 3. With the job manager, it will take a different time to initialize the objects and find an appropriate remote location, which corresponds to the constructor conditions of the object.

The POP-C++ Programs will follow this sequence of methods:

```
Main{
   For i=0, i<nb of objects to create, i++
      Start Timer ti
      Create Object i
      End Timer ti
   Add all timers
}
```

The timer in this snippet will be an “outside view” timer. The other timers described in the preceding schemas will be implemented in the runtime and they will write their results on the screen after the object is created. The time used to print the different values will be counted by the “outside view” timer. This “outside view” timer is not usable for precise comparison with MPI, because it will contain the work from the other timers, and the call method to the constructor of the object, what is not really a part of the initialization. It is just here to give an idea of the total time needed for the creation of a certain amount of objects.

The object used will be as small as possible; they will only have a class, a constructor and a destructor method to make them as similar to processes as possible.

2.5.2. MPI Scenario test

The initialization of distributed programs in MPI follows this sequence:

1. *Node allocation* : first part of the *mpirun* command
2. *Job Launch* : second part of the *mpirun* command
3. *Initialization*: *MPI_Init()* method call
4. *Dynamic processes creation*: *MPI_Comm_Spawn()* method call

I don’t have any possibility to influence the *mpirun* command, so the measurements will be made on the *MPI_Init()* method. The problem is that every process has its own clock, and they are not able to synchronize them to do commands exactly at the same time.

The interesting time in this test is between the moments when the first process call *MPI_Init()*, until the slowest one is ready to use a next function.

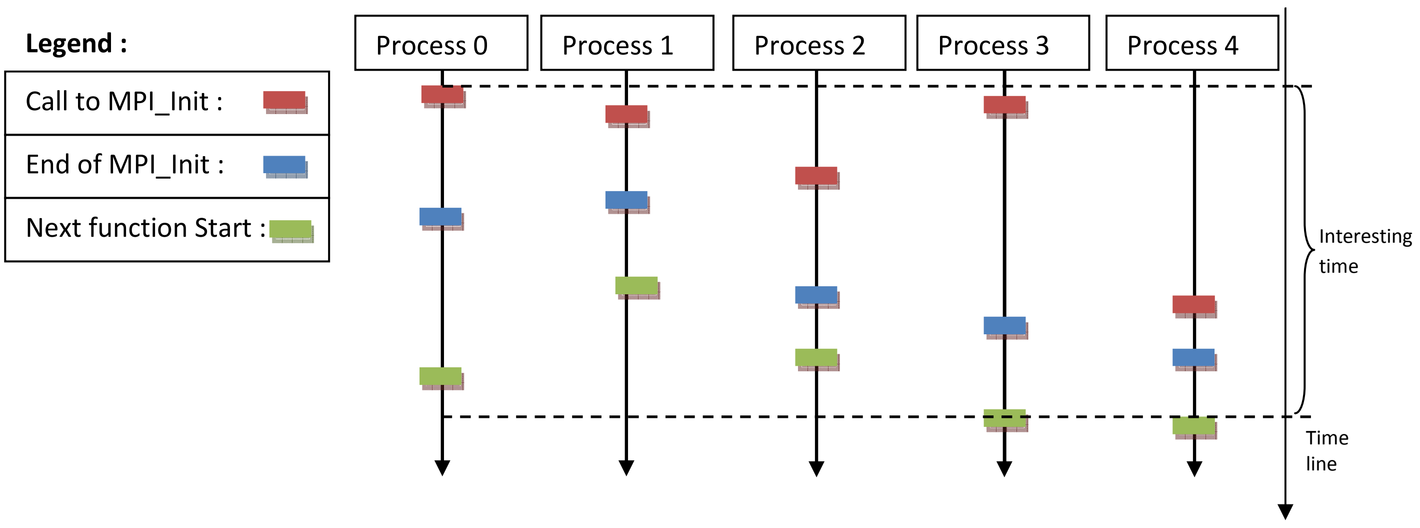


Figure 6 - *MPI_Init()* with several processes

This drawing shows the difficulty to have a precise value of the total time of all calls to *MPI_Init()*. Each process does not start exactly at the same time, and does not finish the call at the same time. Thus, they could all have different laps of time until they can call their next method. To get the best approaching time, I will use a program with the following structure:

```
Main static {
   Start timer
   MPI_Init()
   Stop timer
   Get the highest timer value between the processes
}
```

I will also take the "longest" call to *MPI_Init()*, which will be comparable to the real time I want to obtain (*In the picture, for example, it will be the time from the process 3)*.

The dynamic process creation is measured after the *MPI_Init()*.

```
Main dynamic {
   MPI_Init()
   Start timer
   MPI_Comm_Spawn()
   Stop timer
}
```

This will be measured on the node 0, where the application is launched. We will first create only one process. Each new process spawned will have a different node for location. This can be comparable with POP-C++ object creation without JobManager more than the static MPI initialization, because the nodes are not already known at the start of the application.

2.5.3. Differences between the scenarios

These two scenarios are not really equal, due to the structure of the runtimes, which are different.

The *MPI_Init()* call is made in a parallel way between the processes, and the remote object invocation is made in a sequential way with the actual version of POP-C++. Because of this fact, POP-C++ will surely take more time to finish its initialization.

POP-C++ is an object oriented programming language, and MPI is a lower level language. An object has more functionalities than a process. All the functions can be called with particular arguments; their execution order is described with the keywords in the *.ph* file. For a process, the programmer has to code all the logic and take care of the order of the sends and receives calls.

When calling the *MPI_Init()* function, a part of the remote process creation is already done by the command *mpiexec* (or *mpirun)*, for example the distribution of the processes on the different nodes. The POP-C++ call has to find a node to store the remote object, and communicate with this node.

3. Implementation

I implemented a bash shell script to make the compilation, run and write the results automatically in a file passed as parameter. It is *runTests* (appendix 8.1) and *startBenchmarks* (appendix 8.1). You have to run *runTests*, which will call several times *startBenchmarks* with the right parameters.

It first compiles all the needed files, and then does the tests in this order:

- Empty main time for MPI
- Empty main time for POP-C++
- MPI static process creation
- MPI dynamic process creation
- POP-C++ object creation without JobManager
- POP-C++ object creation with JobManager
- Time Measure

The last test, Time Measure, is used to give a "human" value for the unit used to compare the different programs. The units used are ticks; this is the number of instruction cycle used by the processor. It varies from processor to processor. I took the number of ticks in 10 seconds, to have an approximate value of 1 tick. The time in seconds which is then calculated cannot be taken as a precise one, this is just an approximation. Another solution is to use a C function to get the "human" time of a cycle of a processor, and then use it to translate the ticks. The file *cycle.h* contains the declaration of these ticks, and the method to get the number of processor instruction cycles.

3.1.POP-C++

The modifications of the runtime are not shown here; I added timestamps which print the current number of ticks elapsed on the screen.

MyObject.ph (appendix 8.3) file contains the minimum methods needed to create an object with the JobManager (constructor asking for a power), and without JobManager (string indicating the location where the object have to be created).

MyObject.cc (appendix 8.4) shows the body of the objects. The body could be empty, it prints just here where the object was created, to verify that every object is correctly distributed. For the measurements, this print is not compiled, they are commented out.

Main.cc (appendix 8.5) has to be called with two arguments: the number of objects which are to be created and a Boolean to indicate whether we want to use the JobManager or not.

The test of the total initialization will be made with *mainpop.cc* (appendix 8.9.3) , which will use *objectpop.cc* (appendix 8.9.2) and *objectpop.ph* (appendix 8.9.1). This application will just distribute objects using JobManager or not, depending on the parameters given. The time to execute this application will be measured with the shell command *time*.

3.2.MPI

mainstatic.c (appendix 8.6) uses the methods *getticks()* and *elapsed()* defined in *cycle.h*, which get the clock ticks from the processor. The *MPI_Reduce()* method stores in *globalMax* the max value between all *localMax* values from each process.

At the beginning of *maindynamic.c* (appendix 8.7), only process 0 is created. It then spawns the other processes on different nodes of the cluster. The function *MPI_Comm_spawn()* has to be called with arguments which describe the processes to be launched, the place where they have to be launched, and a communicator, to make them reachable to the “parent” process.

Child.c (appendix 8.8) contains the code of the children processes of *maindynamic.c* (appendix 8.7). They are as simple as possible, to avoid having time measured for anything else than their creation.

The test of the total initialization will be made with *mainstatic.c* (appendix 8.9.4) for the static initialization with *MPI_Init()* and *maindynamic.c* (appendix 8.9.5) which will call *childdynamic.c* (appendix 8.9.6). The time to execute these applications will be measured with the shell command *time*.

4. Tests

The part which I have to test is the following: MPI initialization and object creation in POP-C++. The tests were run on the PHOENIX Cluster from the UNM. **Phoenix** is a 16 node cluster for parallel processing and InfiniBand research. Each node contains two 2.4 GHz AMD Opteron processors, 2 GB of ram, an InfiniBand card, and gigabit ethernet networking. Only the head node, phoenix.cs.unm.edu, contains disks. The other 15 nodes must be booted from the network. The Phoenix rack also contains a 24 port InfiniBand switch, a remotely accessible console/terminal switch, and a gigabit ethernet switch. Phoenix is using Fedora Core 4 as operating system.

We installed POP-C++ v1.1.1 and OpenMPI1.2.3 which support *MPI_comm_spawn()*. Both implementations run on ethernet, to avoid having differences due to different underlying protocols.

The initialization tests are divided into three parts:

- The runtime launch: launch of the application until the program is in the main method.
- The objects/processes initialization: entry in the main method until end of creation of the objects/processes.
- The total initialization: launch of the application until end of the creation of the objects/processes.

4.1.Results

These results got obtained on the PHOENIX Cluster. The following things are to keep in mind when observing the results:

- OS noise: the different O.S. operations, which took some CPU time.
- Hardware differences: some nodes do sometimes answer later than others.
- Other processes: we tried to have the most of the CPU time, but some processes are also running, by the O.S. for example.
- OpenMPI is not well implemented on TCP, this results in slower results than on InfiniBand.

Here is the color code for the measurements and prediction graphs.

Subject	**Color**
Prediction MPI runtime	Violet - - -
Prediction MPI static	Light Red - - -
Prediction MPI Dynamic	Light Orange - - -
Prediction POP-C++ runtime	Gray - - -
Prediction POP-C++ without JobManager	Light Green - - -
Prediction POP-C++ with JobManager	Light Blue - - -
Measure MPI runtime	Dark Violet —
Measure MPI static	Red —
Measure MPI Dynamic	Orange —
Measure POP-C++ runtime	Black —
Measure POP-C++ without JobManager	Green —
Measure POP-C++ with JobManager	Blue —

Tableau 1 - color code of the graphs

4.1.1. Expected

The MPI results will be the best ones, due to the fact that it's a lower level language. The increase of the number of processes will increase time in a logarithmic way. The distribution of processes on nodes follows the logic of a binary tree. The first node starts two other nodes, which then start four other nodes, which then start eight other nodes…

POP-C++ results will be much slower, due to the fact that POP-C++ doesn't just start a process on a remote node, it launches an object with all its methods ready to use, and the associated objects, like the buffer, combox, and broker. It is a higher level language, and allows us to make more actions on it than just send data. It has to pack the data, analyze the received frames and translate them into method calls.

4.1.1.1. Prediction models

The following prediction models are based on first small tests made on the cluster. I will use the values in the following table to create the formulas to determine the comportment of POP-C++ and MPI. Y will be the necessary time of execution of the task; X will be the number of nodes. In this prediction model, I will assume that a tick is $1/(2,4*10^9)$ second (the processors on the cluster are *2,4Ghz AMD Opterons*).

Phase	Time [s]
MPI Runtime initialization	Y=0.25+0.035*X
MPI static process creation	Y = 4’000’000*log2 (X) + 10’000’000 (ticks) = 0,016666 * log2(X) + 0,0416666 [s]
MPI dynamic process creation	50’000’000+680’0000*X (ticks) = 0,2833 + 0,028333 *X [s]
POP-C++ Runtime initialization	Y=0.05
POP-C++ without JobManager object creation	30’000’000*X+9’000’000 (ticks) = 0,125 *X + 0,0375 [s]
POP-C++ with JobManager object creation	25’000’000+10’000’000*X (ticks) = 0,10466 + 0,041666 [s]

Tableau 2 - prediction equations

4.1.1.2. Runtime Launch

This graph shows the runtime launch prediction model:

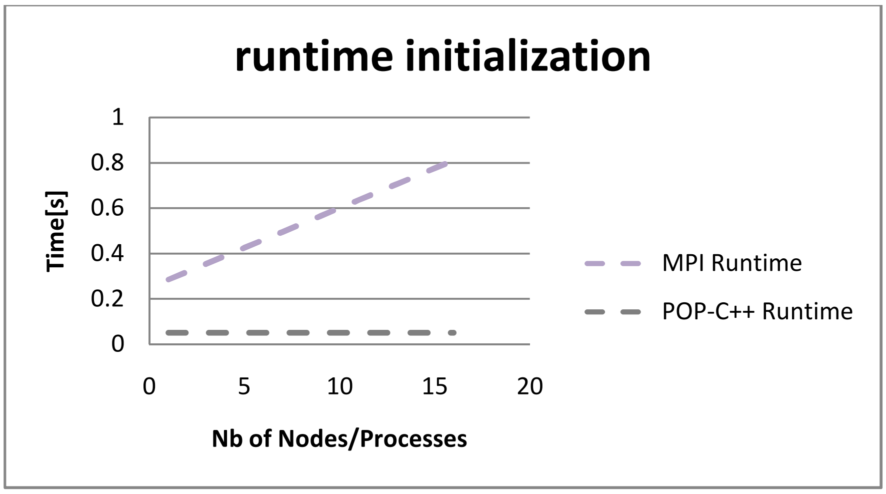


Figure 7 - runtime initialization prediction model

The time to launch POP-C++ Runtime shouldn’t change, because the number of nodes doesn’t affect it. It will be at 0.05 seconds.

MPI will increase with each process, because a part of the initialization of each process is made during the runtime launch. This is the node allocation. It has to be made for each node; it is also a linear increase. MPI will start with 0.25 seconds, and goes until 0.81 seconds.

4.1.1.3. Objects/processes initialization

This graph shows the object/processes creation prediction model:

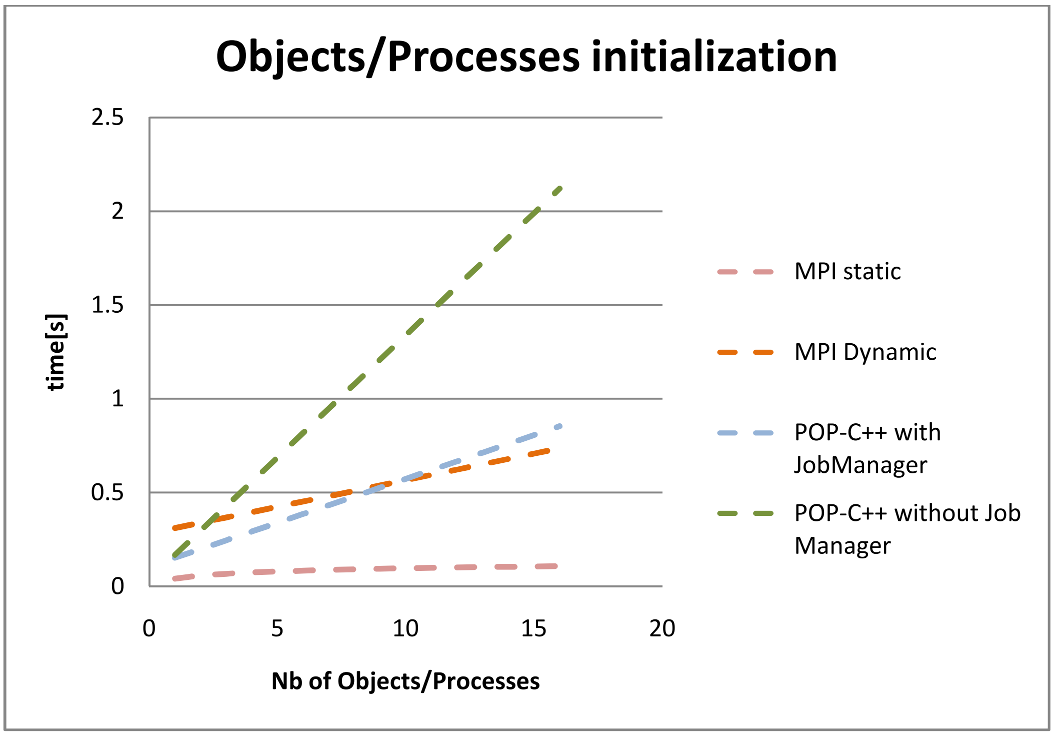


Figure 8 - Processes/Objects initialization prediction model

MPI Static should stay very low, and increase in a logarithmic way. The processes are created in parallel, as described in the section 2.5.2. With 16 nodes, it takes 0.108 seconds to create all the processes.

POP-C++ will increase in a linear way, due to the fact that the objects are created sequentially. POP-C++ without JobManager will be the slower one, due to the fact that the ssh connection between nodes using the user account asks for an authentication. This authentication is done with a pair of public and private keys. The JobManager, as a service, doesn't need to make an authentication each time, that's why it will be quicker on this diagram. This is due to the configuration on the cluster, and we have no right to change it. Without JobManager, it spends 2.12 seconds to create 16 objects, and with JobManager, it spends 0.854 seconds.

MPI dynamic will also increase in a linear way, because it has to start the runtime for each new dynamic process. For 16 processes, MPI dynamic needs 0.736 seconds.

4.1.1.4. Total initialization

This graph shows the total time initialization prediction model:

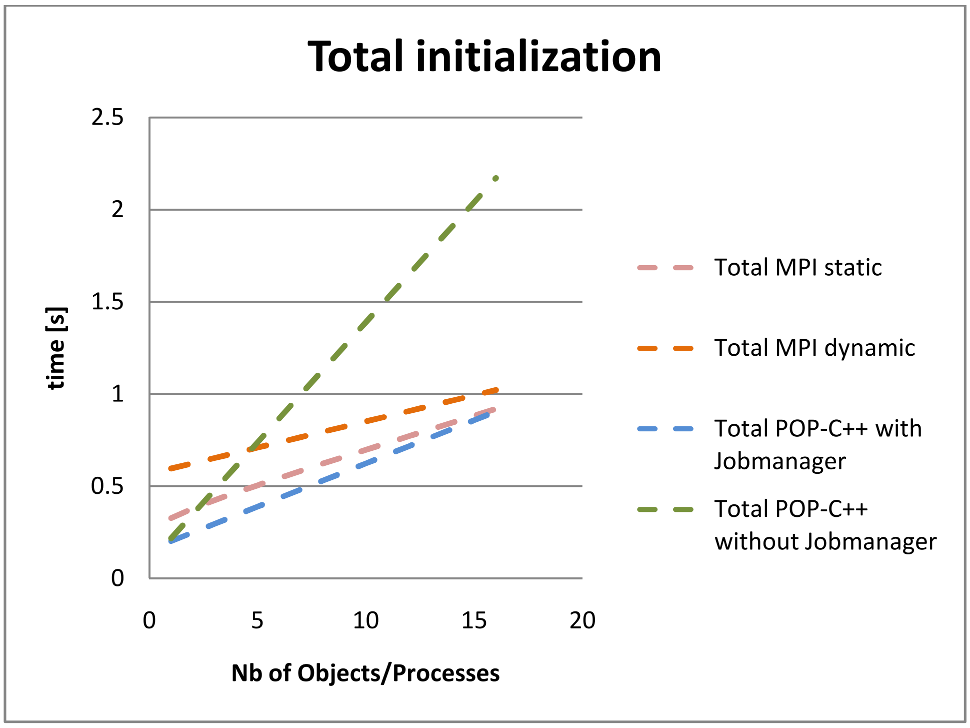


Figure 9 - time initialization prediction model

The total time is the addition of the runtime launch and the object/processes initialization. MPI static is clearly the best solution due to the fact that it has a LOG(N) complexity, compared to all the others, which have an N complexity (linear complexity).

MPI static needs 0.918 seconds to start 16 processes, MPI dynamic 1.021seconds, POP-C++ without JobManager 2.17 seconds and with JobManager 0.904 seconds.

The next part will be the comparison between these prediction models and the results obtained on the PHOENIX cluster. POP-C++ and MPI will also be compared on runtime launch, objects/processes creation and total initialization.

4.1.2. Measured

4.1.2.1. Runtime Launch

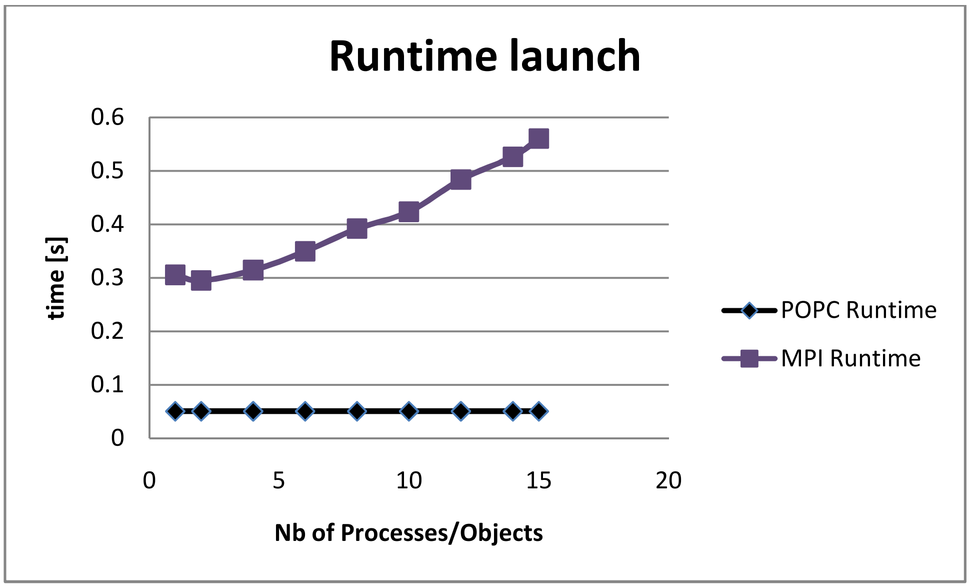


Figure 10 - Launch of the runtimes

This graph shows the launching of the runtime of POP-C++ and MPI. There are differences between them:

- MPI prepares the runtime on every node, 0.28 seconds for one node, and 0.56 for 15 nodes. POP-C++ only starts the runtime on the node where the application is launched. It explains why MPI takes more and more time for each node, and POPC-++ stays at 0.0504 seconds without taking care of the number of nodes potentially usable.
- At this point of the initialization, MPI is not ready to use processes, and POP-C++ doesn't have any knowledge about the remote nodes. The allocation of the nodes is done only for MPI.
- An advantage for POP-C++ is that the number of nodes isn't fixed; it could change during the program execution. It's a more adaptive solution than MPI, for example for programs running on a grid of computers through the internet.

We can say that this comparison is not fair, because MPI has already a contact with other nodes, and POP-C++ not. So MPI is after the runtime launch more advanced in the initialization than POP-C++.

This graph shows the comparison between prediction model and measures:

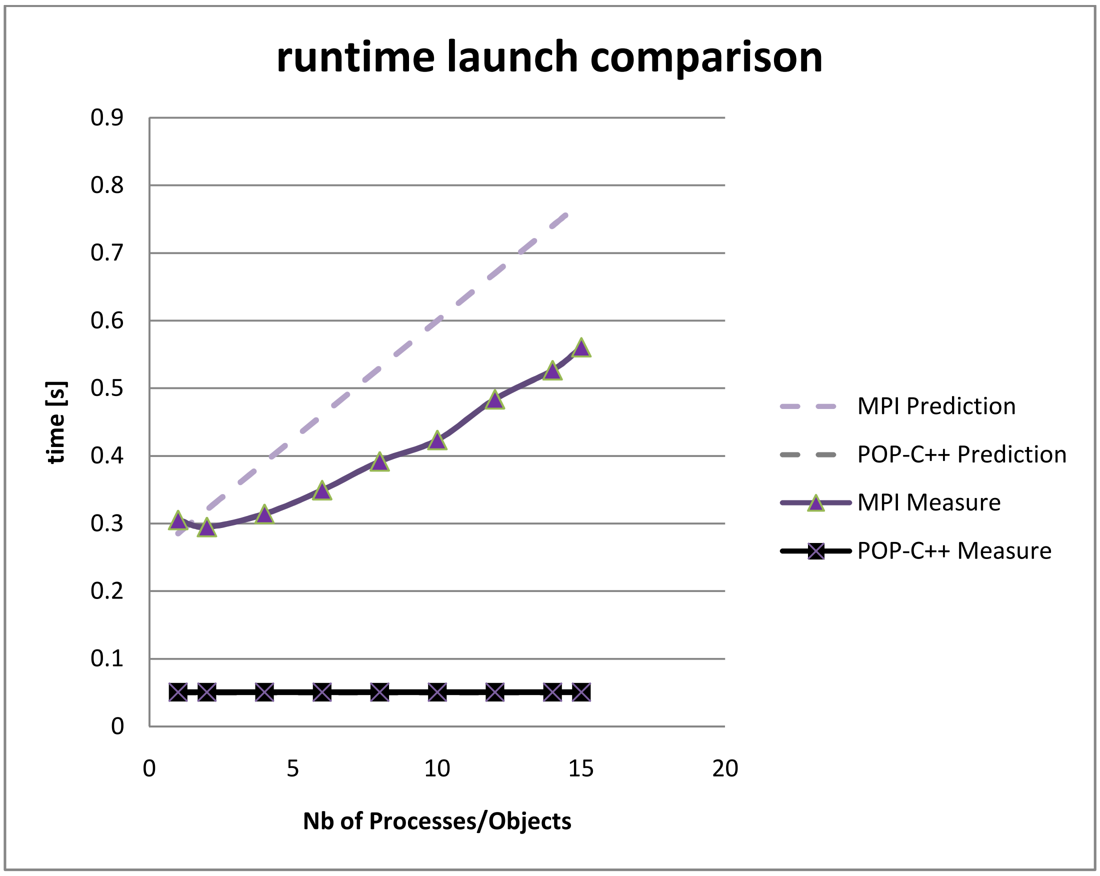


Figure 11 - runtime launch comparison

The launch of the runtime of POP-C++ really follows the prediction model. The difference between them is 0.4 µs. And the measurements stay stable: the number of nodes available doesn't have any effect on it. (The line of "POP-C++ Prediction" stays under "POP-C++ Measure").

The launch of MPI is quicker than the prediction model. It stays linear, with small differences due to the variation of the time used to allocate the remote nodes.

4.1.2.2. Objects/Processes initialization

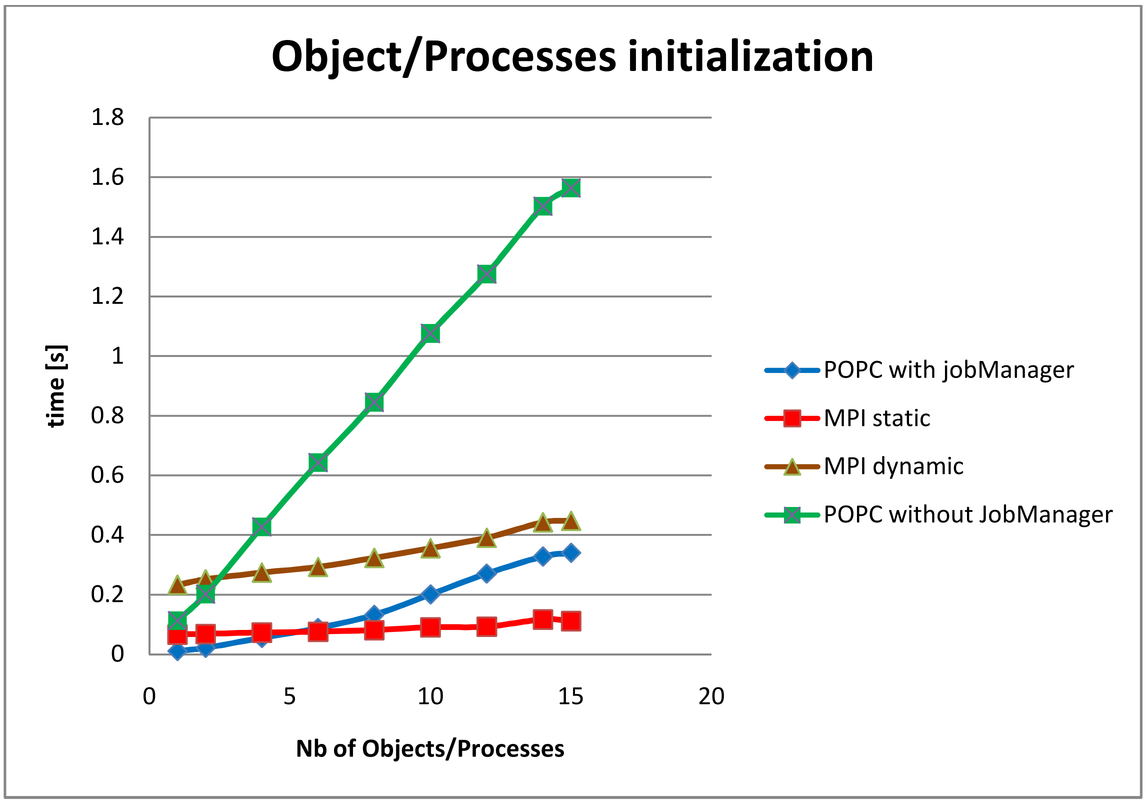


Figure 12 - initialization of the objects/processes

This graph shows the initializations of the objects/processes. We are in the program execution, the runtimes are ready. For MPI static, that's the call to *MPI_Init(),* for MPI dynamic, that's the call to *MPI_Comm_spawn(),* and for POP-C++ that's the creation of the remote objects (the call to the constructor of each object is not counted in this diagram).

- Both POP-C++ follow a linear progression. They create their objects one after another.
- MPI dynamic has to launch the child process on all the different nodes each time (in this case all the remote nodes already have a MPI runtime ready), This is done in a linear algorithm. A better implementation could do it logarithmically, by using the same technique as the static initialization.
- MPI static creates the processes in a logarithmic way; it's the best solution for creating a lot of processes (in this example, 6 and more) in a minimum amount of time.

We see that in the future, the lines from the dynamic MPI initialization and the line of POP-C++ will cross each other. It will be around the node 23 on this cluster.

- For POP-C++ with JobMgr : y=0.024x
- For MPI dynamic : y=0.015x+0.21

(The approximation linear lines were used to determine these equations.)

We can also say that MPI dynamic will be more efficient than POP-C++ with JobManager with 24 nodes or more. This only for the initialization part, the runtime is not counted in this equation.

POP-C++ without Job manager is clearly the slower one, due to the need to authenticate the user on every access on a different node. If this issue could be corrected, POP-C++ will be faster. The time to access to a node is constant (0,02s) for every test I made. I can also say that POP-C++ without JobManager will have a linear increase with every object added.

This table shows the difference between each application for 15 nodes:

Application	**Time [s]**	**Percentage [%]**	**remark**
MPI static	0.11	100	Fastest one
MPI dynamic	0.446	405.45	Acceptable value for a dynamic process creation.
POP-C++ without JobManager	1.563	1420	High value due to the authentication issue
POP-C++ with JobManager	0.340	309.09	Acceptable value, the initialization part of a program is not the most important one. You can use this value also as reference for POP-C++ without JobManager, if it hadn't the authentication issue.

Tableau 3 - percentage comparison for objects/processes creation

This graph shows the comparison between the prediction model and the measures for MPI:

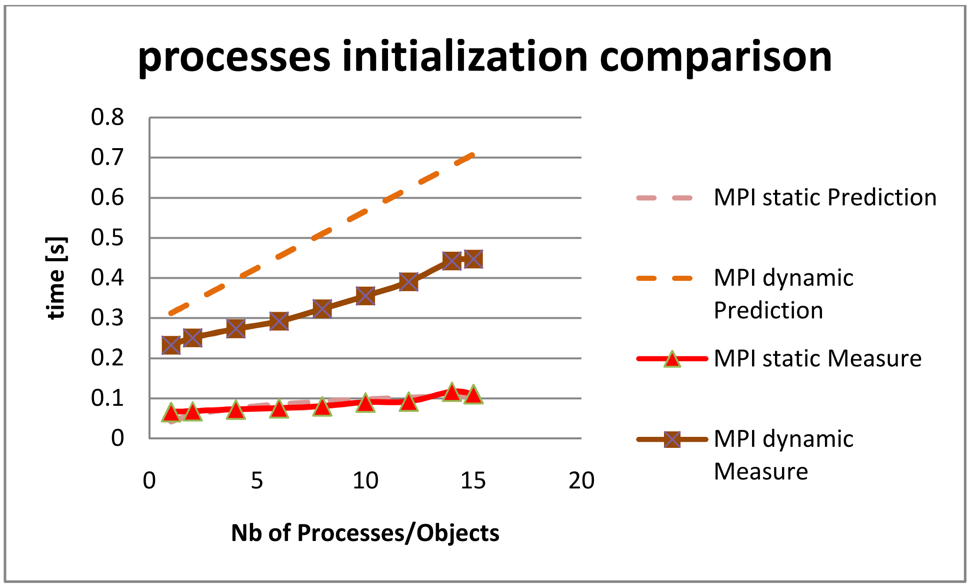


Figure 13 - processes initialization comparison

For MPI static, the differences between the prediction model and the measured values are only due to the OS noise, and the network. The complexity is not clearly visible here, but with more nodes, we can see the LOG(N) complexity appears on the line (the line of "MPI static Prediction" stays under "MPI static Measure").

For MPI dynamic, the prediction model was too pessimistic. An explanation is that the first measurements used to determine the behavior of MPI were taken with another version of MPI than the one used to take the final measures. Between these two tests, the runtime of MPI was changed to allow Manuel Schrag to do some tests between InfiniBand and TCP (see Manuel Schrag's report [4] for more details). But the complexity of the equation doesn't change between the prediction model and the measures.

This graph shows the comparison between the prediction model and the measures for POP-C++:

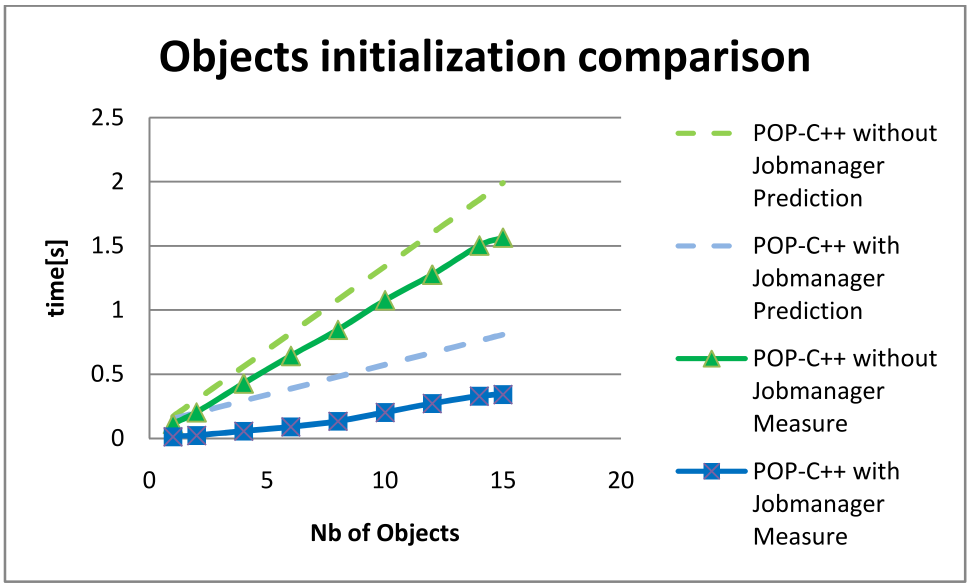


Figure 14 - Objects initialization comparison

The predictions for POP-C++ are all too pessimistic, but they stay in the right complexity (linear for both cases). This diagram proves that POP-C++ without JobManager is slower on this cluster, due to the authentication problem. Here is an example of the different values obtained for the timestamps.

With JobManager:

```
diff between t1 and t2Job :  8661891.000000
diff between t1 and t3Job : 25924708.000000
diff between t1 and t4    : 26960248.000000
```

Without JobManager :

```
diff between t1 and t2NoJob :   2688238.000000
diff between t1 and t3NoJob : 259319593.000000
diff between t1 and t4      : 260870956.000000
```

The difference is between the timestamp 2 and 3, during the communication with the remote node. For POP-C++ with JobManager, it's 17262817 ticks, also 0.0072 seconds, and for POP-C++ without JobManager, it's 256631355 ticks, also 0.11 seconds. It is 15.27 times slower. For the other timestamps, POP-C++ without JobManager is faster. If this issue will be corrected, POP-C++ without JobManager will be the fastest one.

4.1.2.3. Total Initialization

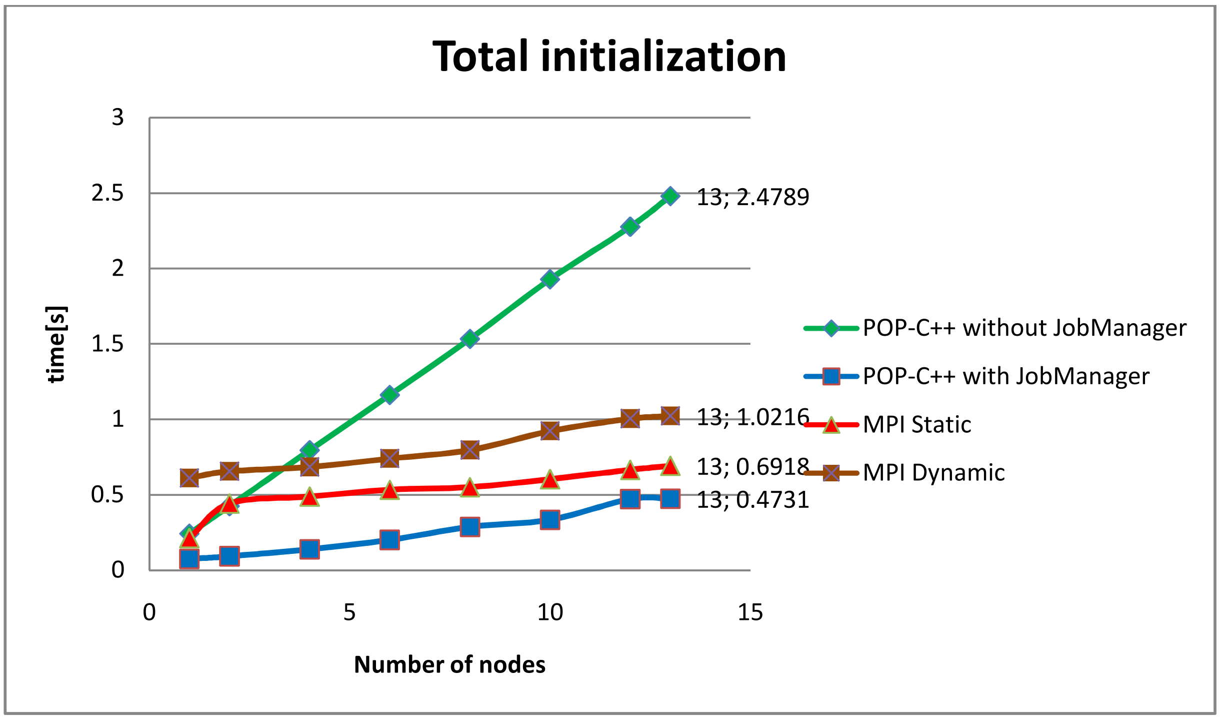


Figure 15 - launch of the runtime + initialization of the objects/processes

On this graph, we can see a more complete comparison between the runtimes. This measurement includes the launch of the runtime, and the initialization of the processes/objects. For POP-C++, the constructor call is also taken here. The goal of this diagram is to show the needed time to launch an application, until the program is ready to use the processes/objects and make them communicate.

- MPI Static starts with a bigger time than POP-C++, but then continues in a logarithmic way. For programs with more than 24 objects, it's definitely the most rapid one. For 13 processes, it needs 0.691 seconds.
- Both POP-C++ and MPI dynamic continue in a linear way. If MPI stays later on the same protocol (TCP), POP-C++ with JobManager will be a better solution; it increases less than MPI per object.
- This table shows the difference between each application for 13 nodes (total initialization):

Application	Time [s]	Percentage [%]	remark
MPI static	0.691	146.08	46% slower until 13 nodes, but will become faster with 24 processes and more
MPI dynamic	1.021	215.85	Will stay slower than POP-C++ with JobManager
POP-C++ without JobManager	2.478	524.31	High value due to the authentication issue. Without this issue, could be faster as POP-C++ with JobManager
POP-C++ with JobManager	0.473	100	Fastest one

Tableau 4 - percentage comparison for total initialization

This graph shows the comparison between the prediction model and the measures:

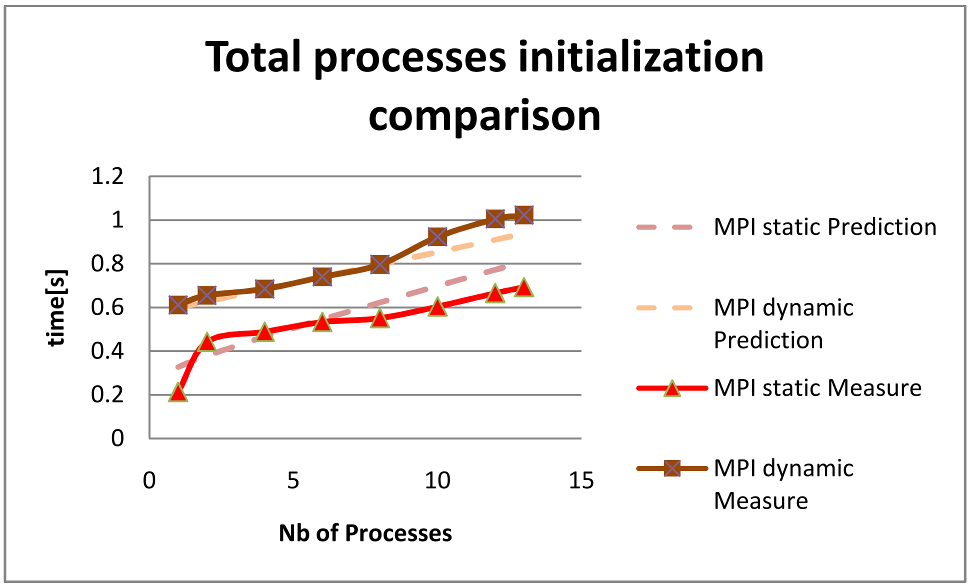


Figure 16 - Total processes initialization comparison

The difference between the predictions and the measurements for MPI dynamic can come from the network communications. The node 10 was occupied with other applications, and took more time to finish its initialization of its MPI process.

For MPI static, the measurement shows a difference of 0.22 s between the initialization of one and two processes. This difference comes from the fact that when two processes are created, they are called one after another. For more processes, MPI can then use the binary tree logic: both processes 1 and 2 wake one process each, which makes four processes. The logic is followed until all the needed processes are ready.

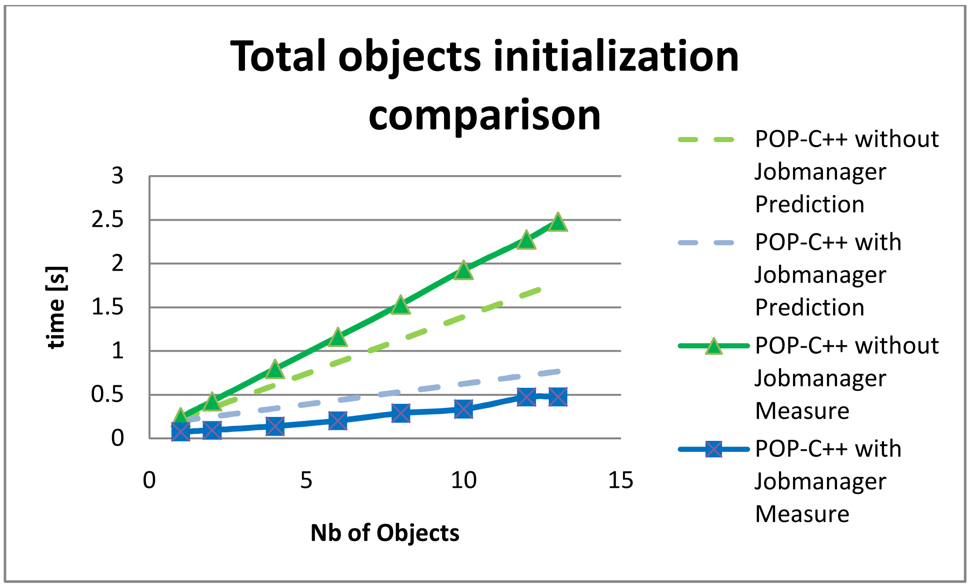


Figure 17 - Total objects initialization comparison

Between the prediction model and the measure of POP-C++ with JobManager, the difference is in the time needed to create an object. The prediction was too pessimistic.

For POP-C++ without JobManager, the time to initialize an object was also too long in the prediction. We can also see that the prediction wasn't false, both lines are linear.

The next chapter will conclude this first part of my diploma project, by talking about the problems, the measurements, and a personal conclusion.

5. Conclusion

5.1.Encountered problems

5.1.1. Cluster configuration

Now ok, bug resolved by An-Thuan. It is possible to use POP-C++ without job manager. The problem was that the runtime from POP-C++ used rsh to communicate between the different nodes, which was not usable on the cluster. Now, by adding `export PAROC_RSH=/usr/bin/ssh` to the user bash file, the communication between nodes without Jobmanager is possible. It is until now very slow (0,2s), as described during the analysis of the measurements.

5.1.2. Cluster hours

This is not a real problem, but more an organizational need. We had access to the cluster on Mondays and Wednesdays. So we had to adapt our planning to fit to these days. It explains also why we needed more time to finish the interpretation of the results than first planned.

5.1.3. POP-C++ static object array creation

Until now, it is not possible to select the constructor we want to use in POP-C++, when creating arrays of objects. For the runtime launch, with a static array of objects, I had to use the JobManager, to avoid having all the objects on the same node. I put `@{power(20)}` after the default constructor, to make the JobManager distribute the objects. A solution for this problem would be the development of the third part of our project: Arrays in POP-C++.

Another problem with this object creation is that POP-C++ doesn't allow us to declare a paroc object before the main. It always gives the following exception:

```
Exception thrown by: paroc_exception*
```

At this point, no solution has been found. Create a test for POP-C++ where an array of objects is created before the main isn't useful, because it will be equal to the total initialization of POP-C++ (in this case, with JobManager).

5.2.About the measured tests

5.2.1. POP-C++ with and without Jobmanager

About the initialization, POP-C++ is comparable to MPI. The tests show that the initialization of a few objects (in this example, 1-4) can be made in less time than MPI. For more objects, or for objects on a

Grid without an NFS, where the time to get the executable of the object will be longer, the next version of POP-C++ with asynchronous object creation will increase a lot the performances of the runtime.

The measures were quite regular; they all have a small standard deviation. An interesting one is POP-C++ with JobManager compared to POP-C++ without JobManager: the time used by the JobManager to find an appropriate place always depends on the current resources, and of the availability of the nodes.

Example of the standard deviation of POP-C++ for 14 nodes (initialization of the objects only, the runtime has no interest here). The percentage is calculated with the following formula: $\frac{standard\ deviation*100}{average\ time}$.

Standard deviation	Time[s]	Percentage [%]
POP-C++ with JobManager	0.171	73.1483852
POP-C++ without JobManager	0.044	4.174313199

Tableau 5 - standard deviation

We see that the standard deviation with JobManager is really more important, and explains why the measurements do not completely follow the prediction model. It always depends on the number of nodes the JobManager has to ask before it finds a node which has the requested requirement.

As a comparison, POP-C++ without JobManager stays more stable, because the time to ask a particular node is always the same. It proves that without JobManager, when the issue about authentication will be corrected, it will be more stable than POP-C++ with JobManager.

5.2.2. Standard deviation of the total initialization

The following chart shows the different standard deviation of MPI and POP-C++.

Standard deviation	Percentage[%]
MPI static	2.43
MPI dynamic	5.61
POP-C++ without JobManager	1.45
POP-C++ with JobManager	3.23

Tableau 6 - standard deviation of the total initialization

The standard deviation of POP-C++ with JobManager is more important than the one without JobManager. It proves that the previous comparison was right, that the JobManager has a more random time to find a node than the version without.

POP-C++ is more constant for the creation of objects than MPI static. This can be considered as an advantage to determine the time needed for the initialization of an application.

5.3.Personal

POP-C++ doesn’t really take more time than MPI to be ready to be used, but the number of objects affects it more than the number of processes for MPI. This is not a complete comparison of the runtimes, due to the time that we available. The PHOENIX cluster had 16 nodes, I think that with a bigger cluster (64 nodes, for example) the results could be more interesting. It was a good introduction to MPI and to the world of parallel computing. Working with a cluster was very interesting, and permitted me to see different environments. This first part of the project was a good introduction to the HPC and it permitted me to have an overview of the issues which can happen in this field.

The next Chapter contains the Task 2: Global communication in POP-C++. In this chapter, it will be shown how we added the possibility of calling broadcasts and reduce methods with POP-C++ Objects. This will be developed in a library for POP-C++.

Collective Communication in POP-C++

1. Introduction {common}

This part of the project consists in adding collective communication to POP-C++. POP-C++ already provides a feature to use collective communication by using MPI underneath (see [4]). What we want to do is add collective communication independent from MPI. Collective communication means that more than 2 entities are involved in a communication process (point to multipoint, multipoint to multipoint).

1.1. Introduction to collective communication in POP-C++ without changing the parser

I will focus on adding collective communication without changing the actual POP-C++ parser. This restriction implies that I have to find a way to call methods on every type of object, without knowing their structure before, and I have to be able to handle every type of parameter and return. For the version which changes the actual POP-C++ parser, please refer to the project report from Manuel Schrag [4].

I will need to use templates in C++ to develop this library. The next chapter gives a small introduction to templates. For more precise information, please refer to the web pages in the references. The chapter Analysis (chapter 0) explains the different functionalities that the library has to offer. The chapter Conception (chapter 3) gives a more precise view of the library, thinking of the fact that it will be written in C++. The Implementation (chapter 4) indicates the actual state of the code and the limitations of the library.

1.2. Templates in C++

Templates in C++ permit to make classes, methods and variable independent of a type. A template can be understood as a generic type, which takes a real type at the compilation time. This snippet shows the way to create a method using templates.

```
template <class T>
T max (T a, T b) {
  return (a>b)? a : b;
}
```

This snippet shows how to use the template class created in a C++ code

```
void main()
{
       cout << "max(10, 15) = " << max(10, 15) << endl ;
       cout << "max('k', 's') = " << max('k', 's') << endl ;
       cout << "max(10.1, 15.2) = " << max(10.1, 15.2) << endl ;
}
```

This snippet shows the output of the main.

```
max(10, 15) = 15
max('k', 's') = s
max(10.1, 15.2) = 15.2
```

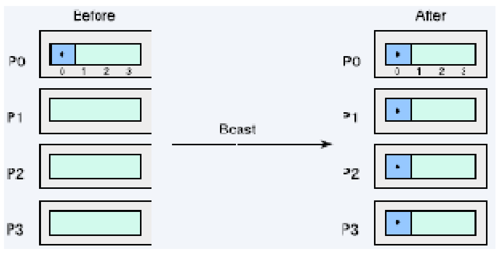
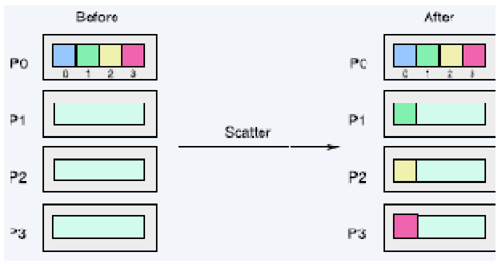
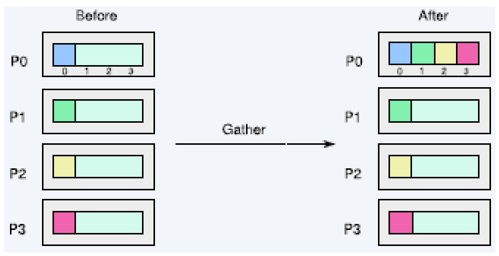
The method *max(T a, T b)* takes as parameter every type of data possible. You can even give objects to it, if the operator > is implemented. At compilation time, *T* is replaced by the type given when calling the method. In this example, we will have a m*ax* for *int*, *char* and *double*.

The snippets and examples are from [3], a web page which has a complete explanation of the function and class templates.

1.3.Collective communication in MPI{common}

Understanding point-to-point communication in MPI is recommended before reading this section (see Manuel Schrag's report [4]).

In parallel computing we can find communication models where more than 2 processes are involved at the same time. The MPI standard specifies a bunch of functions which are defined under the term *Collective communication* and which correspond to such communication models. Often, one process is distinguished from the others (by its rank) and is commonly called *root* (for 1-N / N-1 functions). But in some functions, every process does the same (N-N functions).

Method	Description	Visualization
Broadcast	The *root* process sends identical data to any other process of the same group.	 **Figure 18 - MPI broadcast[9]**
Scatter	*root* sends different data of same size to any other process of the same group.	 Before After P0 P1 P2 P3 0 1 2 3 Scatter **Figure 19 MPI Scatter** Erreur ! Source du renvoi introuvable.
Gather	*root* gathers the data from all involved processes and places them, sorted by rank, into its reception buffer.	 Before After P0 P1 P2 P3 0 1 2 3 Gather **Figure 20 MPI Gather** Erreur ! Source du renvoi introuvable.

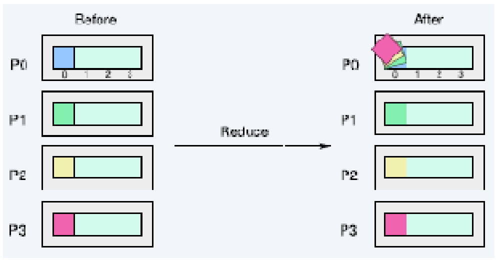
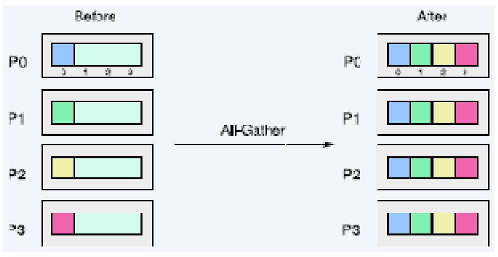
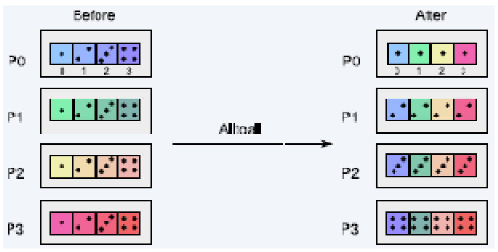
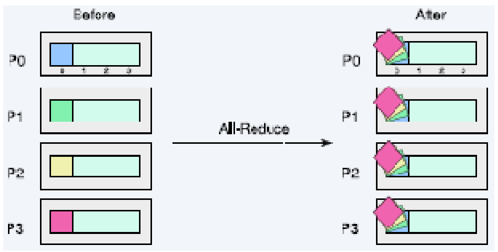
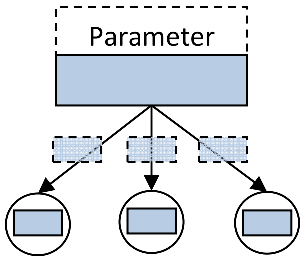
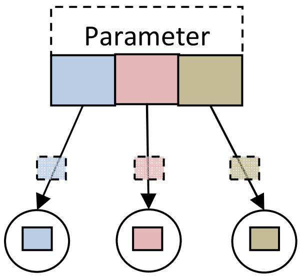
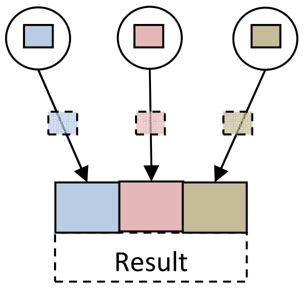
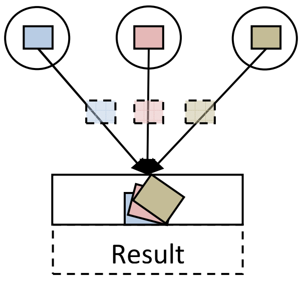
Reduce	The main idea is to do the same as the *Gather* operation. But before storing every single received piece into the reception buffer, a Boolean or arithmetic operation is performed to combine the data. And only the result of this operation is stored in the reception buffer of *root*.	 **Figure 21 MPI Reduce** Erreur ! Source du renvoi introuvable.
AllGather	No *root* process is present in this function. It corresponds to a multi broadcast where every process sends its data to every other process of the group. The final reception buffer of those will be identical.	 **Figure 22 MPI Allgather** Erreur ! Source du renvoi introuvable.
All-to-All	This is an N-N function and thus doesn't contain a *root* process. Process *i* sends the *kth* block of its send buffer to process *k*, which stores it in the *ith* block of its reception buffer.	 **Figure 23 MPI All-to-All** Erreur ! Source du renvoi introuvable.
AllReduce	N-N function with no *root* process. Multi broadcast with following reduction operation.	 **Figure 24 MPI Allreduce** Erreur ! Source du renvoi introuvable.

Tableau 7 - 1 Explanation and visualization of MPI collective communication functions

1.4.Collective communication in POP-C++{common}

In the current version of POP-C++ it is possible to couple MPI code in a POP-C++ program by using a special template class. The disadvantage in this approach is that we lose the object oriented paradigm and introduce the message passing paradigm instead. **Erreur ! Source du renvoi introuvable.** explains how collective communication should be implemented to keep the OO paradigm. Circles represent parallel objects and rectangles represent a data element.

Method	Description	Visualization
Broadcast	The program invokes a method on the group by passing data in parameters. This data is communicated to all members of the group and is identical for every one of them.	 Figure 25 POP-C++ Broadcast
Scatter	The program invokes a method on the group by passing lists of data elements in parameters. Each parameter is a list of data elements. The *ith* element of this list is sent to the *ith* object in the group, thus every one receives potentially different data.	 Figure 26 POP-C++ Scatter
Gather	For the invoked method on the group every member returns one data element. The final result on the caller’s side is a list of data elements.	 Figure 27 POP-C++ Gather
Reduce	The main idea is to do the same as the *Gather* operation. But instead of storing every data element separately, only the result of an arithmetic or logic operation of these elements is stored.	 Figure 28 POP-C++ Reduce

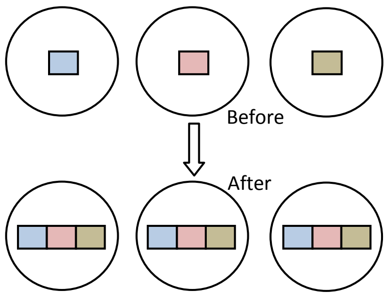
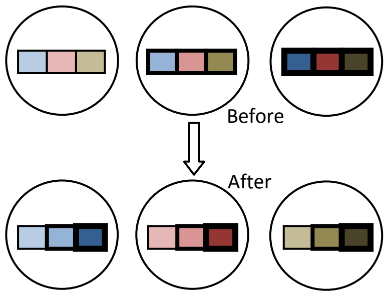
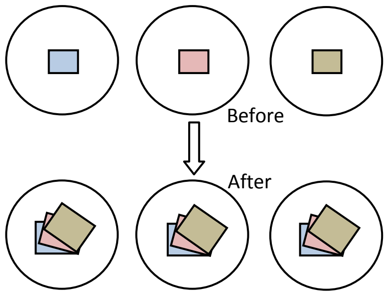
AllGather	This operation corresponds to a multi broadcast where every object of the group calls a method on every other member *i* by passing the *ith* data element of its list as a parameter. After the operation, all objects contain the same list of data elements (same order). The data element passed by object *i* is at the *ith* position.	 Figure 29 POP-C++ Allgather
All-to-All	Object *i* invokes a method on every object *k* by passing the *kth* data element of its list as a parameter. Object *k* stores the received data element at the *ith* position of its list.	 Figure 30 POP-C++ All-to-all
AllReduce	This operation corresponds to a multi broadcast with following reduction operation.	 Figure 31 POP-C++ Allreduce

Tableau 8 - Explanation and visualization of POP-C++ collective communication functions

2. Analysis

This analysis explains the main idea to develop a collective communications library. The functionalities and their ideal comportment will be described in the following chapters. This library has to be developed without modifying the POP-C++ parser. The goal is also to create a syntax easily understandable for a programmer who is used to C++ and POP-C++.

The next chapter will explain a possibility to implement this library and how to use it.

2.1.Independent parser library

With this method, groups of objects which are supposed to use collective methods are made. The class which represents these groups has to give the following possibilities:

Create a group	To create a group, you need to know the type of object the group will contain, and a estimate of the number of objects you will have
Add members to the group	To add a member, you need a reference to the object, an existing group which can contains the same type of objects.
Remove members from the group	To remove a member, you need to know his position in the group.
Get the ID of a member	To get the ID of a member, you need the reference to this member
Call a broadcast on the group	To call a broadcast on the group, you need to a valid function for broadcasting and implement the method *doAction* in the object class.
Call a reduce on the group	To call a reduce on the group, you need to a valid function for reducing, a valid parameter to store the return value, a valid parameter to determine the type of reduction you want and implement the method *doAction* in the object class.
Call a gather on the group	To call a gather on the group, you need to pass in parameter a valid function for gathering, a valid parameter to store the results and implement the method *doAction* in the object class.
Call a scatter on the group	To call a scatter on the group, you need to pass in parameter a valid function for scattering, and implement the method *doAction* in the object class.
Merge groups	To merge groups, you need the reference to two different groups. After a merge, all the objects of both groups will be included in the first group.

Here is a pseudo code for the method doAction:

```
return type doAction(String methodString){
   get methodName from methodString
   get the parameters from methodString
   switch methodName{
      case method1:
         return this.method1(parameters);
         break;
      case method2:
         return this.method2(parameters);
         break;
      case method3:
         return this.method3(parameters);
         break;
      default:
         return null;
  }
}
```

The return type of *doAction* will be the return type of the method passed as a String. *doAction* has to be asynchronous when it has void as return type, and synchronous else.

Having a common method in every object which will be used with our collective library permit us to stay generic. Without that, we would have to write a specific library for each kind of object we have.

That's a way to avoid the problem of the non-existence of reflection in C++. We have to do it "artificially", in the object class. A next step will be finding a way to use the reflection in C++, and then generating this method in the library. The use of reflection will be discussed in the chapter: conception (see chapter 3).

Constraints:

- Each group contains only one kind of objects, which is specified when calling the constructor.
- The object has to implement a *doAction* method, to simulate reflection (in a first time).
- Before the destruction of an object, the programmer has to remove it from every group it belongs to.
- You cannot create group of primitive types.
- Broadcasted and scattered methods cannot have a return type.
- Reduced and gathered methods must have a return type.
- The group is not a parallel object, so you can't give a reference of a group to a remote object. This is due to the use of templates in POPGroup.

2.1.1. Use Case Diagram

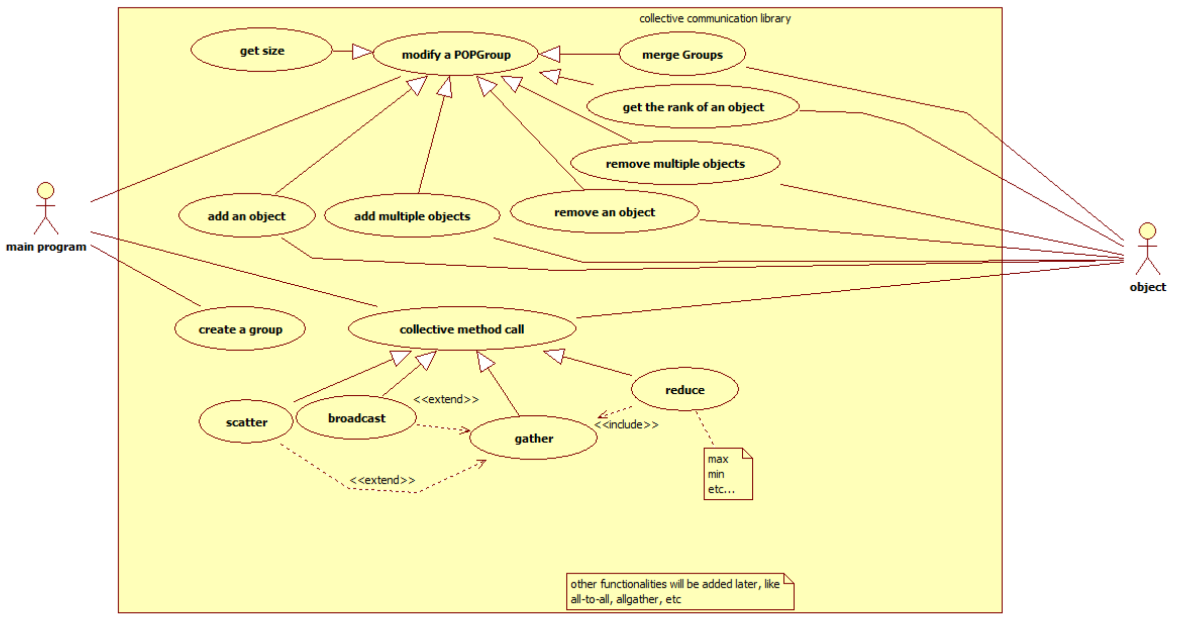


Figure 32 - use case

There are two main parts in this use case.

The first one is all the action possibilities on a group itself, like: add an object to the group, remove an object from the group get the size of the group and get the rank of a member of the group. These actions don't have any effect on the objects; some of them just need a reference to them.

The second part is the actions which interact with objects. This is the collective method calls. When doing a broadcast, a gather or a reduce, a method is called on every object which is in the specified group. The interaction with the objects is the call to the method which is a parameter passed to the collective method.

The link between *collective method* and the user *object* represents in fact the link which is between each collective method (like broadcast, gather …) and the object.

The group doesn't create the objects itself. The programmer has to first create the objects in his main application, and then add the reference of these objects to the group.

2.1.2. Sequence diagrams

In the sequence diagrams, the POPGroup is always on the local machine, the same as the main program, and the T objects (Objects of type T, the type used to initialize the POPGroup) are remote objects. The POPGroup doesn't contain T objects, only a reference to their interface.

2.1.2.1. Create a group

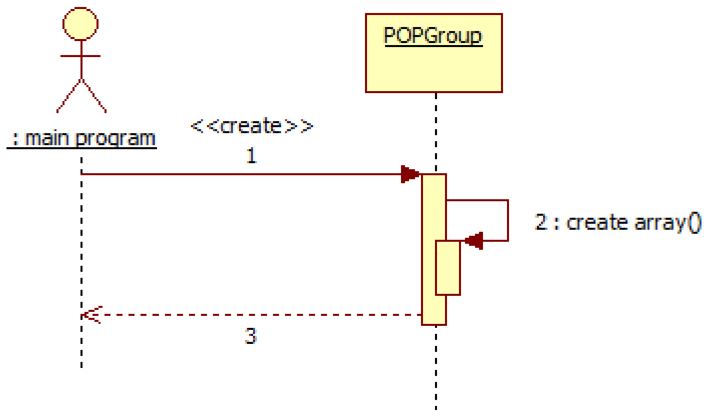


Figure 33 - create a group sequence diagram

At the creation of a group, the size is set to 0 and an array of the type of objects is initialized. The group is then ready to store objects (of the specified type), and execute collective operations on it. The array will have a predefined size, and will grow when necessary.

2.1.2.2. Modify a group

The use case “get size” and “add Members “ have been integrated to this one. It was too simple and little to make a sequence diagram just for them.

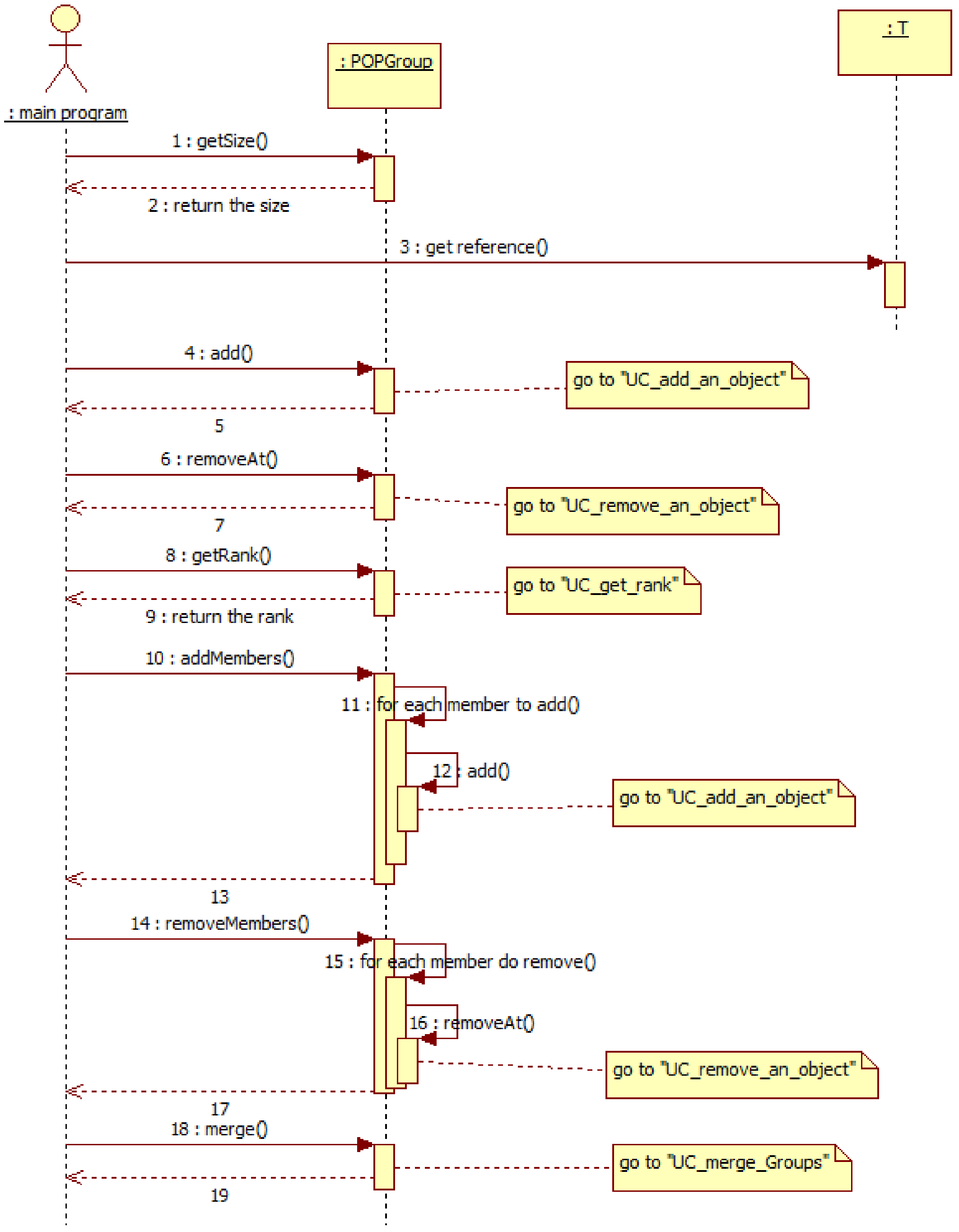


Figure 34 - modify a group sequence diagram

This diagram shows the available methods for handling the group. If a new method to modify the group is needed, it will be shown on this schema. The size returned is the highest rank +1 in the group. For example, if you have 4 objects, the highest rank is 3, so the size returned is 4. A part of addMembers is also on this diagram. This method takes as parameter an array of objects. Every Object of the array is used as parameter for the method addMember. This is done in the library, to avoid this job for the programmer.

2.1.2.3. Add an object

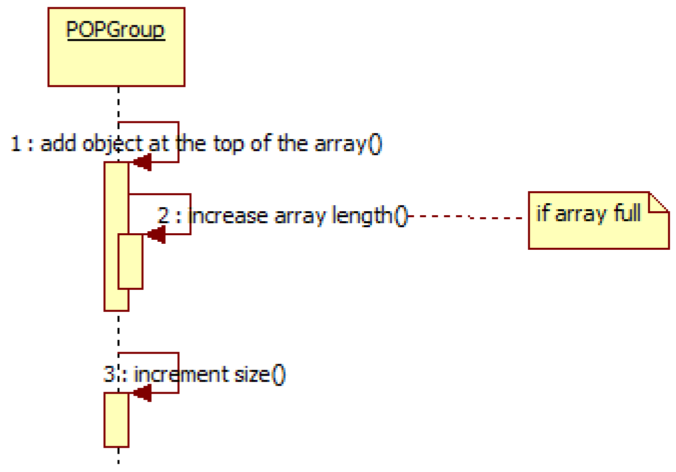

POPGroup
1 : add object at the top of the array()
2 : increase array length()
if array full
3 : increment size()

Figure 35 - add an object sequence diagram

When adding an object to the group, you should do the following verifications: is the object from the good type? One possibility of testing that is using a template table in our library. At the creation of the Group, the array will be initialized with the type of the Object given as parameter, and then it won't allow other type objects to be stored in it.

2.1.2.4. Remove an object

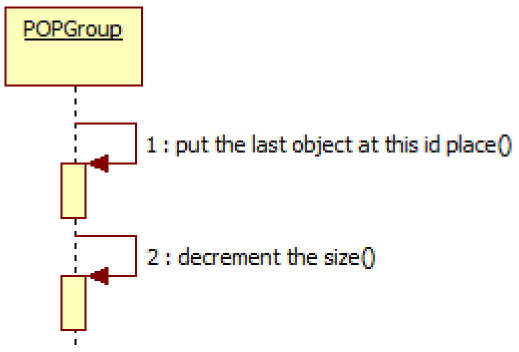


Figure 36 - remove an object sequence diagram

Here is an example of on algorithm which could be used. In this example, we want to delete the second object of the array:

Step 0, actual state			**Step 1, place last object instead**			**Step 2, decrement size**		
0	Obj 1		0	Obj 1		0	Obj 1	
1	Obj 2	Size = 4	1	**Obj 4**	Size = 4	1	Obj 4	Size = **3**
2	Obj 3		2	Obj 3		2	Obj 3	
3	Obj 4		3	Obj 4		3	Obj 4	

Tableau 9 - decrementation algorithm

The object 2 is not really deleted; we only delete the reference to it in the group.

Another solution is to use a vector to handle the members. This will be implemented, because a vector is available in the standard library of C++.

2.1.2.5. Get the rank of an object

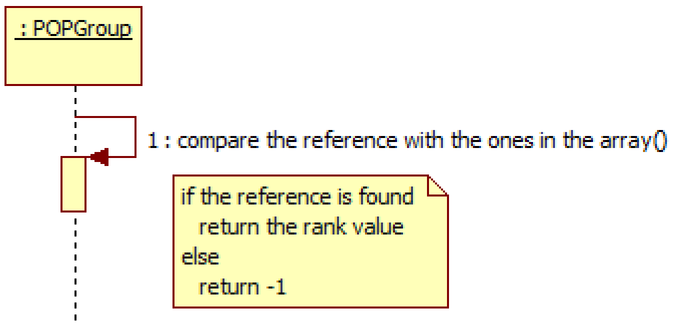

: POPGroup
1 : compare the reference with the ones in the array()
if the reference is found
return the rank value
else
return -1

Figure 37 - get rank sequence diagram

For the comparison in this diagram, the reference to the object (the address contained in the pointer) will be used. This method is mainly useful when you want to delete an object. Due to the deletion algorithm, objects don't keep automatically the same rank during the program run, you have to get the actual rank of an object before trying to delete it.

2.1.2.6. Merge Groups

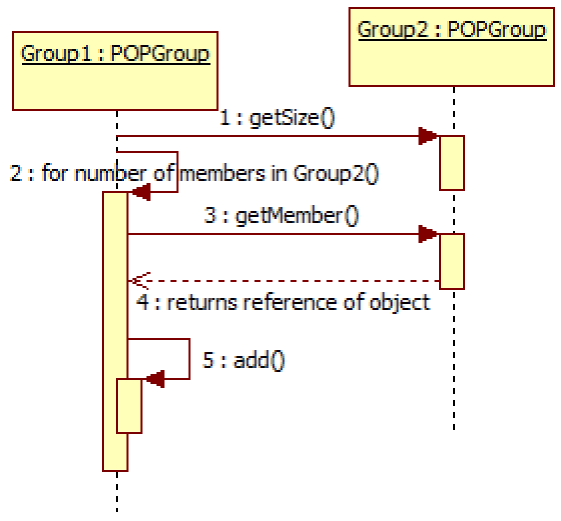


Figure 38 - Merge Groups sequence diagram

Every object included in the second group is added into the first one, using the *getMember()* method. After a merge, the group 2 is not different. This could be useful for creating subgroups to simulate multicast for example. The objects are not copied; their references are copied from a group to the other one.

2.1.2.7. getMember

This method returns a reference to the member at the rank given in parameter.

2.1.2.8. empty & isEmpty

The empty method permits to empty a group from all his object references. The method isEmpty returns true if the size of the group is zero.

2.1.2.9. Collective method call

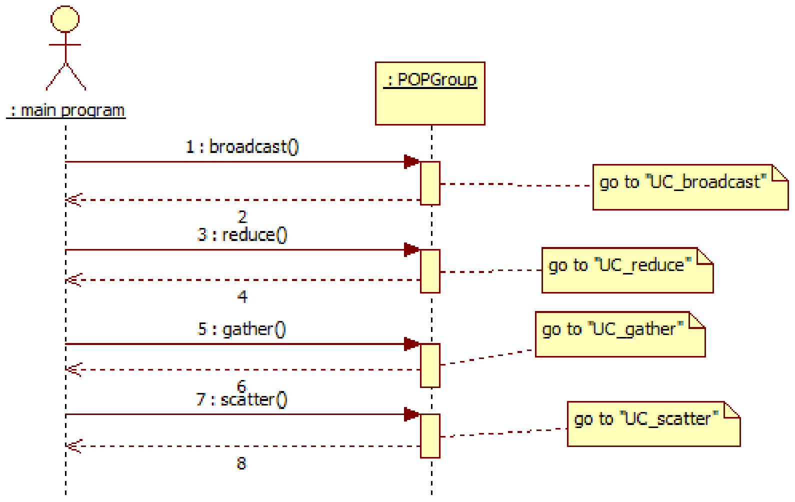


Figure 39 - collective method call sequence diagram

This diagram shows every actually available collective method on the group. Every time a collective method is called, it has a String containing the method to call on the object, and his parameters. For example, if we want to call *add(4)* on each of our Integer objects :

```
Integer o1();
Integer o2(“localhost”);
POPGroup<Integer> myGroup(3);
myGroup.addMember(o1);
myGroup.addMember(o2);
myGroup.broadcast(“add”,4);
```

For a method with a return value, we have to call for example a broadcast followed by a reduce, and give the memory address of the return:

```
int return;
String parameters = {“get”};
myGroup.broadcastreduce(“get”, &return, POPGROUP_REDUCE_MAX);
```

In this case, the reduce call will make a *get* on every object, and put into the variable *return* the value of the highest object.

The reduce and gather cases don’t correspond to real methods calls. The real methods you can call for a reduce are broadcastReduce and scatterReduce. For a gather operation, it’s broadcastGather and scatterGather. Everytime you want to have a value back, you have to send a method call before, using either a broadcast or a scatter. The next sequence diagrams just indicate the send or receive part of the action to focus on the interesting comportment.

2.1.2.10. Broadcast

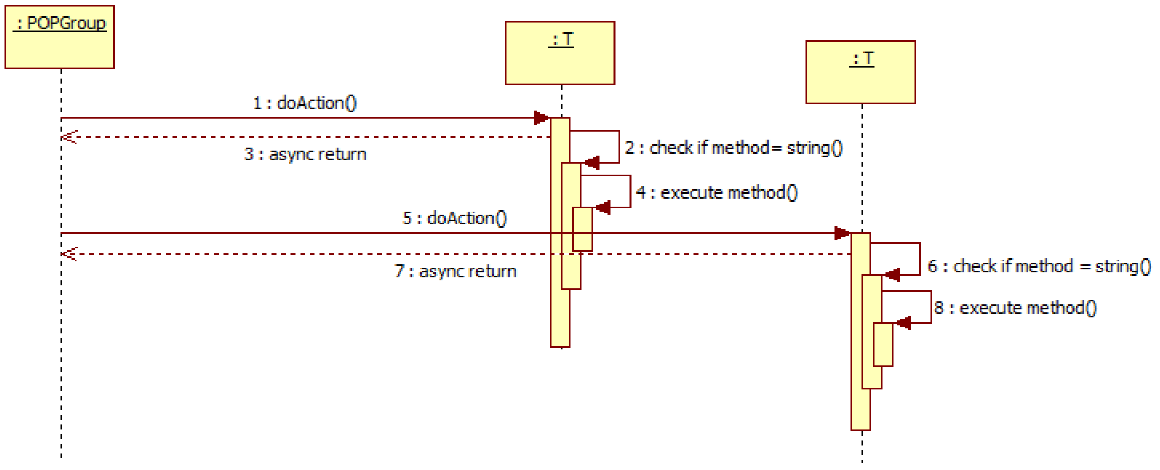


Figure 40 - broadcast sequence diagram

The method which is broadcasted may have an asynchronous call to be more effective. With a synchronous call, the broadcast has to wait until the first object has finished his method to call the second one.

2.1.2.11. Gather

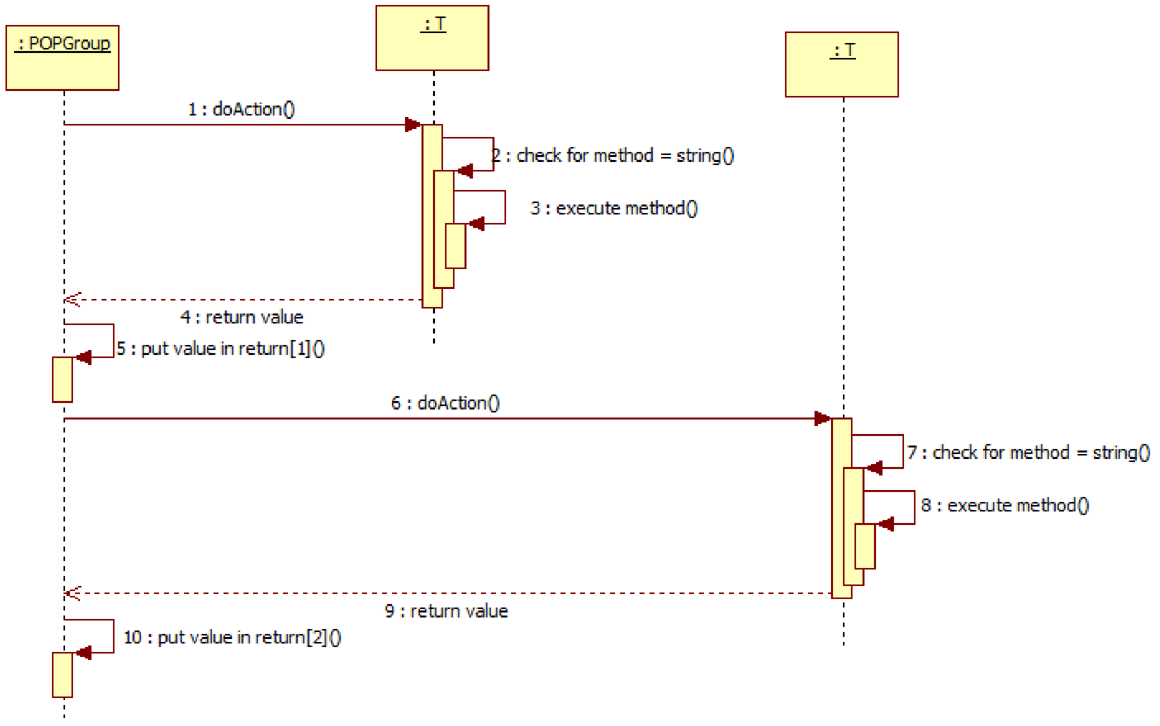

: POPGroup
: T
: T
1 : doAction()
2 : check for method = string()
3 : execute method()
4 : return value
5 : put value in return[1]()
6 : doAction()
7 : check for method = string()
8 : execute method()
9 : return value
10 : put value in return[2]()

Figure 41 - gather diagram sequence

The gather call will function in a synchronous way. You have to wait until each object has finished the method call until the next one can start.

To get the result of a precise object, after the gather is terminated, you can get the rank of the object you want to know the result. This rank will indicate the position of the result of this object in the returned array. Before a gather, you always have a broadcast or scatter, to determine the way the parameters are sent to the objects.

2.1.2.12. Reduce

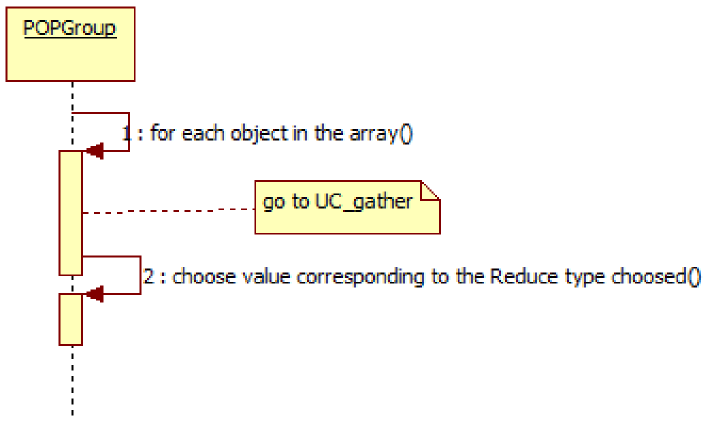

POPGroup
1 : for each object in the array()
go to UC_gather
2 : choose value corresponding to the Reduce type choosed()

Figure 42 - reduce diagram sequence

The difference between the reduce and the gather methods is at the end of the method. Gather just returns all the collected values, and reduce has to choose one amongst all of them. To make this choice, the programmer will have to give a parameter indicating the value selection method. The following selection possibilities will be available:

- POPGROUP_REDUCE_MAX
- POPGROUP_REDUCE_MIN
- POPGROUP_REDUCE_OR

Adding other possibilities will be very simple, you have to add the constant value to the library, and then codes the algorithm to choose the return value of reduce. Before a reduce, you always have a broadcast or scatter, to determine the way the parameters are sent to the objects.

2.1.2.13. Scatter

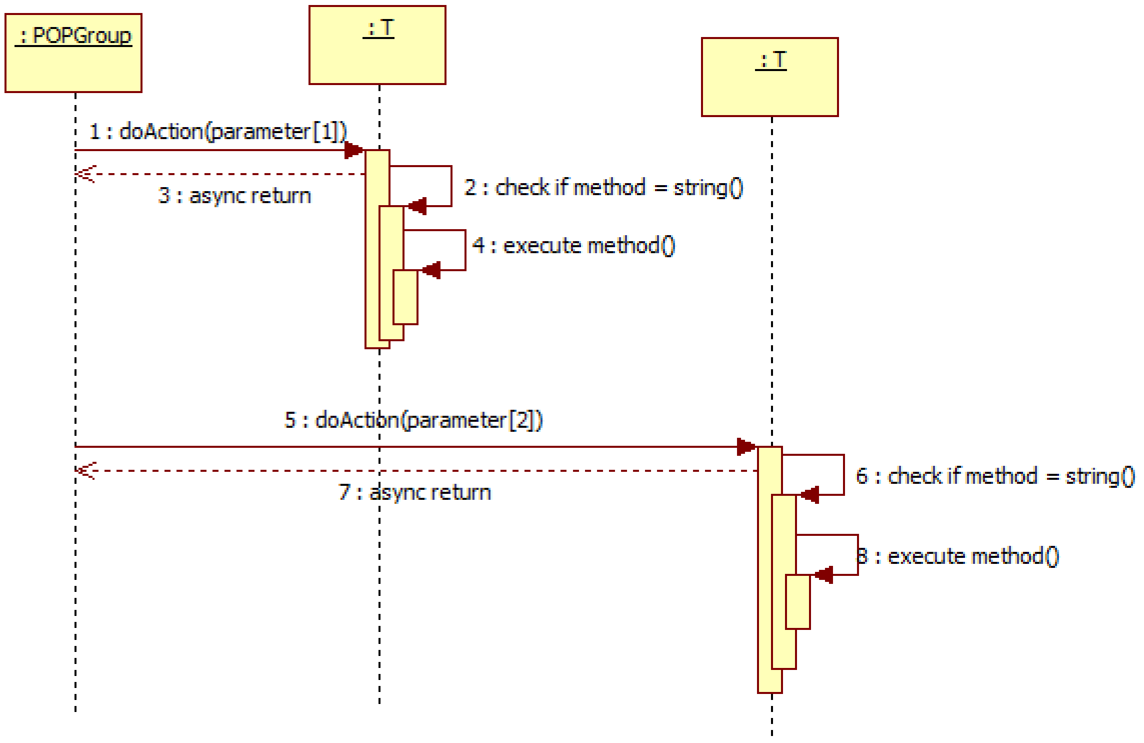

: POPGroup
: T
: T
1 : doAction(parameter[1])
2 : check if method = string()
3 : async return
4 : execute method()
5 : doAction(parameter[2])
6 : check if method = string()
7 : async return
8 : execute method()

Figure 43 - scatter sequence diagram

On this diagram, the scatter operation is explained. The scatter method takes two parameters: the string representing the method and an array which has the different parameters to distribute. For each object in the member's array, doAction() will be called, with the arguments corresponding to his rank in the parameters array.

2.1.3. Class diagram

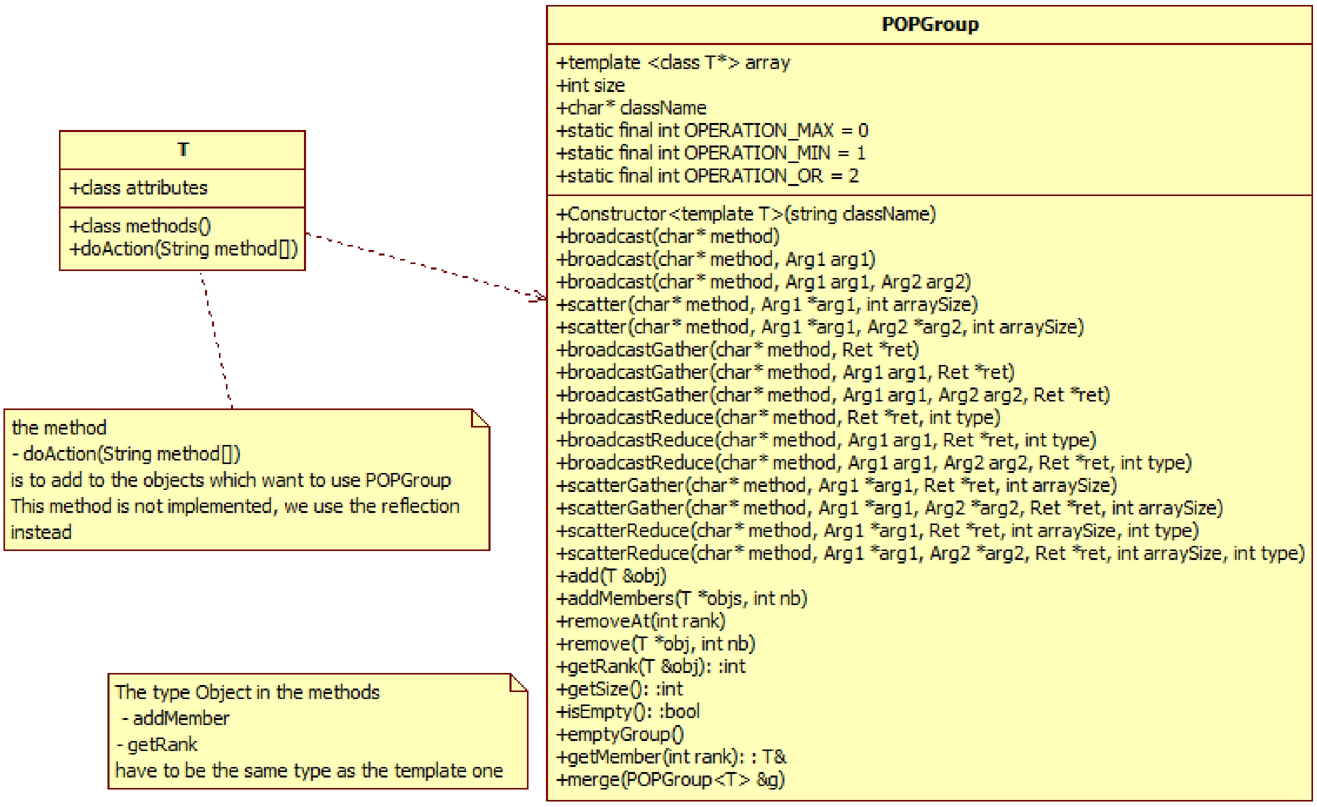

T
+class attributes
+class methods()
+doAction(String method[])
the method
- doAction(String method[])
is to add to the objects which want to use POPGroup
This method is not implemented, we use the reflection instead
The type Object in the methods
- addMember
- getRank
have to be the same type as the template one
POPGroup
+template <class T*> array
+int size
+char* className
+static final int OPERATION_MAX = 0
+static final int OPERATION_MIN = 1
+static final int OPERATION_OR = 2
+Constructor<template T>(string className)
+broadcast(char* method)
+broadcast(char* method, Arg1 arg1)
+broadcast(char* method, Arg1 arg1, Arg2 arg2)
+scatter(char* method, Arg1 *arg1, int arraySize)
+scatter(char* method, Arg1 *arg1, Arg2 *arg2, int arraySize)
+broadcastGather(char* method, Ret *ret)
+broadcastGather(char* method, Arg1 arg1, Ret *ret)
+broadcastGather(char* method, Arg1 arg1, Arg2 arg2, Ret *ret)
+broadcastReduce(char* method, Ret *ret, int type)
+broadcastReduce(char* method, Arg1 arg1, Ret *ret, int type)
+broadcastReduce(char* method, Arg1 arg1, Arg2 arg2, Ret *ret, int type)
+scatterGather(char* method, Arg1 *arg1, Ret *ret, int arraySize)
+scatterGather(char* method, Arg1 *arg1, Arg2 *arg2, Ret *ret, int arraySize)
+scatterReduce(char* method, Arg1 *arg1, Ret *ret, int arraySize, int type)
+scatterReduce(char* method, Arg1 *arg1, Arg2 *arg2, Ret *ret, int arraySize, int type)
+add(T &obj)
+addMembers(T *objs, int nb)
+removeAt(int rank)
+remove(T *obj, int nb)
+getRank(T &obj): :int
+getSize(): :int
+isEmpty(): :bool
+emptyGroup()
+getMember(int rank): : T&
+merge(POPGroup<T> &g)

Figure 44 - POPGroup class diagram

The class POPGroup is the library for collective methods on POP-C++ objects. Until now, the method doAction had to be written in the object file, to simulate reflection. Finding a solution to generate this method dynamically in the library would make it totally generic, for each kind of object.

In the next Chapter, Conception, I will talk about the different problems we can have implementing this analysis, the way to use templates, and an explanation about how to use reflection in C++.

3. Conception

This chapter contains the conception of the collective methods library which doesn't need a change of the parser. I will explain more precisely the possibilities to implement it, considering the type of the arguments in the methods and I will admit that reflection is possible in C++.

3.1. Version without modification of the parser

The following paragraphs explain the methods used to implement the library *POPGroup*. The goal was to make a library as independent as possible to the kind of object which will be stored in it. For each proposition, I will give the advantages and disadvantages.

3.1.1. Type of the stored parallel objects

There are actually two ways to store the type of the objects which have to be stored in the POPGroups. The first one is the utilization of templates, and the second one the utilization of a class from which every parclass has to inherit to use collective communications.

Templates are used to determine the type of the remote objects which will be stocked in the POPGroup. To store them, the library doesn't need to know their methods, variables or structures. A Vector of the type of the object will be initialized. For example, this is what happens when you want to create a POPGroup containing Integer paroc objects:

```
POPGroup<Integer> myGroup(“Integer”);
```

This possibility, coupled with reflection, permits the programmer to call collective methods and pass the name of the object method which has to be handled in parameter. Nothing is added to the object, all the work is done in the library. The programmer has to know which object method can be called with which collective method. To see the rules for calling methods, please refer to the following comparison table on the next page. This table is an abstract for programmers who want to use the POPGroup library. They will have to check on this table the possible calls they can make based on a method signature. The following rules are to know:

- Methods with no return value are able to be broadcasted or scattered.
- Methods with return value are able to be reduced or gathered.
- Methods without parameter are automatically broadcasted.
- Methods with parameters can be broadcasted or scattered.

Method in *.ph* file	Possible collective method calls

`void myMethod()`	`/* broadcast : call the method on every member */`
	`void broadcast(“myMethod“)`

`void myMethod(int n, int m, …)`	`/* broadcast : call the method with same parameters (n,m,…) on every member */`
	`void broadcast(“myMethod“, int n, int m, …)`
	
	`/* scatter : call the method with different parameters (n[i],m[i],…) on every member by specifying the size of the array(s) (size) */`
	`void scatter(“myMethod“, int *n, int *m, …, int size=getSize())`

`int myMethod()`	`/* gather : call the method on every member. Gather the return values in an array (res[])*/`
	`void gather(“myMethod“, int res[])`
	
	`/* reduce : call the method on every member. Return result of reduce operation passed as a parameter (op) in the out parameter res*/`
	`void reduce(“myMethod“, int res,char *op)`

`int myMethod(int n, int m, …)`	`/* broadcast and gather : call the method with same parameters (n,m,…) on every member. Gather results in an array (res[]) */`
	`void broadcastgather(“myMethod“, int n, int m,…, int res[])`
	
	`/* scatter and gather: call the method with different parameters (n[i],m[i],…) on every member by specifying the size of the array(s) (size). Gather the return values in an array (res[]) */`
	`void scattergather(“myMethod“, int *n, int *m,…, int res[], int size=getSize())`
	
	`/*` **`broadcast`** `and reduce : call the method with same parameters (n,m,…) on every member. Return result of reduce operation passed as a parameter (op) in the out parameter res*/`
	`int broadcastreduce(“myMethod“,int n, int m,…, int res, char *op)`
	
	`/* scatter and reduce : call the method with different parameters (n[i],m[i],…) on every member by specifying the size of the array(s) (size). Return result of reduce operation passed as a parameter (op) in the out parameter res */`
	`int scatterreduce(“myMethod“, int *n, int *m,…, int nb=getSize(), int res, char *op)`

Tableau 10 - abstract of the translation of standard method to collective method

The template possibility has the following disadvantage: templates are not usable in the actual version of POP-C++. Due to this restriction, we cannot design the POPGroup as a remote object, and so we cannot pass his reference to remote objects. Due to this, it is impossible to make N-to-N communications, or create trees on the network to gain methods execution time.

Another possibility is creating a class containing the methods which will be called by the library, when the programmer wants to use collective communication. A big disadvantage of this concept is that the programmer has to inherit his objects from this library class, and he has to implement the collective methods. We don't want to give this amount of work to the programmer, so will this solution will not be implemented, it is just mentioned here, and was not chosen for the rest of this conception.

3.1.2. Type of the parameters

The parameter type problem is the following: we cannot write in the library a method for each possibility of parameter type. We have to find a solution to pass the parameter type when calling a collective method on a group.

I found different solutions:

- Use templates for each parameter
    - This method will be more effective than the second one, and was chosen for this conception. We can pass directly the parameter in the function. A disadvantage is that we are limited by the number of arguments we want to pass. For every possible number of arguments, we have to create the corresponding method for broadcast, gather, scatter and reduce. This method will permit us to pass paroc objects, and standard C++ objects if they inherit from the class POPBase.
- Use char* containing the type to make it generic
    - This method will require that we develop syntax to create every parameter possibility, including arrays, structures, objects… The parsing of the char* will take time, and the possibility of passing objects will function only if we have a method to serialize them. Actually, we can serialize the data in the objects, but not the objects and their structure. This solution is just mentioned here, and was not chosen for the rest of this conception.

3.1.3. Reflection in POP-C++

First of all, we needed to find an open source library on Internet to add reflection to C++. I decided to use Reflcpp (see their website [6]). This library was developed during Spring 2007, so it is quite new, and I had the possibility to send Emails to the developers of this library. They helped me to resolve some problems with the actual version available on their web page (see their website [6]). To see the encountered problems, please refer to section 5.1.1.

The reflection allows me to cast a string into a method call without knowing the structure of the object on which I want to call a method. The reflection is needed in the collective communication calls of the library. For every collective communication, a string representing the method is passed as parameter. I have to transform this string into a method call on the object without knowing before compilation time which kind of object it is.

4. Implementation

4.1.Test

To test the library, I created a main which calls every method of the library, and a class which has every possible method: with or without return and/or parameters, parameters as base type of class type. The code source is available in the appendix, chapter 8.10.10. Here is the output when launching the main for the reflection example:

```
barraf@barraf-laptop:~/example_reflection$ ./main.out

start of the Test
Object A constructed
Object B constructed

Call doMethod on Object B
Name of the class : [A]
Get an instance of the methode 'methodA1' of A
Invoke 'methodA1' on A

Call method1 on Object A
Get an instance of the methode 'methodA2' of A
Invoke 'methodA2' on A, (value should be [4])

 !!! invocation !!!Call method2 on Object A with value=[4]
Get an instance of the methode 'methodA3' of A
Invoke 'methodA3' on A
Call method3 on Object A
Value of return method 3 (should be [14]) = [14]
Get an instance of the methode 'methodA4' of A
Invoke 'methodA4' on A, (value should be 4*2=[8])
Call method4 on Object A with value=[4]
Value of return method 4 (should be [8]) = [8]
Get an instance of the methode 'methodA5' of A
Invoke 'methodA5' on A, (value should be 4+8=[12])
Call method5 on Object A with value1=[4] and value2=[8]
Value of return method 5 (should be [12]) = [12]
```

In *doMethod* in the object B, we make a call to each method of the object A using reflection. Every result that we receive is correct.

A second main with parallel object has been written, but not tested, due to the problem of the compilation with POP-C++ (See 5.1.1 for more details). The same tests are made.

4.2.Actual state of development

Actually, the library cannot be compiled with POP-C++ when using parallel objects. When running the library with a classical C++ object, the result shows that every collective and management method works correctly. The result is in the annexes, chapter 8.10.14. We can see that all the functionalities of the library work. A corrected version of the library reflcpp is available on the CD, and a new version will be soon available. The developers of the reflcpp library are actually working on it.

4.2.1. Installation

To use the POPGroup library, you need to install reflcpp. For his, please refer to their website [6] and README.TXT, which is in the zip file of reflcpp. I made some needed changes into their source code, essentially in codegenerator.cpp. The modified version of reflcpp is on the CD of this project.

You can also take the directory */src* under *reflcpp* and paste it into the home directory of the application. Then compile the library with the following command (assuming you are in the src directory):

```
$ g++ -c *.cpp
```

When compiling, pleaes do not forget to link the .o files of the library with the application.

This installation supposes that POP-C++ is already installed on the machine. If not, please refer to the user guide for POP-C++ [5] for the installation of the POP-C++ runtime.

4.2.2. Utilization

To use the library (for our case, in C++ or POP-C++ without parallel objects, due to the actual issues), the following steps are to be made:

- Implement the application which wants to use POPGroup.
- Generate the needed code for reflection with the codegenerator tool (assumed you are in *home/user/relf-0.1-inst/bin*):

```
./cg.sh -t=cpp /home/myApplic/Object.hpp NameOfObject > /home/myApplic/Object_reflection.cpp
./cg.sh -t=hpp /home/myApplic/Object.hpp NameOfObject > /home/myApplic/Object_reflection.hpp
```

- Include the Object_relfection.cpp file into the file POPGroup.hpp.
- Compile the application with the file Object_reflection.cpp and the *reflcpp* library.

```
g++ -c  mainAB.cpp A.cpp B.cpp A_reflection.cpp
g++ -o main.out *.o ./src/*.o
```

- Run the application

```
./main.out
```

When trying to get an instance of the wrong class, or from a class from which the Class_reflection.cpp and Class_reflection.hpp files are not available, you obtain this error at the execution time:

```
barraf@barraf-laptop:~/example_reflection$ ./main.out

start of the Test
Object A constructed
Object B constructed
main.out: Type.cpp:90: static const reflcpp::Type_body*
reflcpp::Type_body::getType(const std::string&): Assertion `it != s_class_name_map-
>end()' failed.
Aborted (core dumped)
```

The error says that a pointer should be different from the end of the class name map. The class name map contains all the classes which have reflection files to allow them to be used with reflection. When the pointer it is at the end of this list, the class which we want to instantiate doesn't exist.

When a method which doesn't exist on the actual object is called, the following error will appear at the execution time:

```
barraf@barraf-laptop:~/example_reflection$ ./main.out
```

```
start of the Test
Object A constructed
Object B constructed

Call doMethod on Object B
Name of the class : [A]
Get an instance of the methode 'wrongMethod' of A
terminate called after throwing an instance of 'reflcpp::ReflectionException'
  what():  wrongMethod does not exist in A (src/ClassType_tmpl.hpp:1140)
Aborted (core dumped)
```

In this case, the error message is very clear. The method we try to call, *wrongMethod,* doesn't exist in the class *A.*

When calling an existing method with wrong arguments, like too many arguments, or a wrong type of argument, you obtain this error at execution time:

```
barraf@barraf-laptop:~/example_reflection$ ./main.out

start of the Test
Object A constructed
Object B constructed

Call doMethod on Object B
Name of the class : [A]
Get an instance of the methode 'methodA1' of A
Invoke 'methodA1' on A
Object A constructed
main.out: src/MemberFunctionDcl.hpp:52: typename Ret_TP::type
reflcpp::MemberFunctionDcl000<Obj_TP, Ret_TP, FuncP, Name>::invoke3(Obj_TP*, int,
reflcpp::Arguments&) const [with Obj_TP = A, Ret_TP =
reflcpp::FundamentalType_tmpl<void>, typename Ret_TP::type (Obj_TP::* FuncP)() =
&A::methodA1, const char* Name = ((const char*)(&
reflcpp::A_strings::m_methodA1))]: Assertion `n == 0' failed.
Aborted (core dumped)
```

The error here is less readable. The assertion which failed explains that the reflection library didn't found any method with the required name and arguments. The lines before explain that the method cannot be satisfied with the given types for the templates.

Here is an example of the commands used with the object A in POPGroup (the code source is available in the appendix).

```
barraf@barraf-laptop:~/POPGroup$cd src/
barraf@barraf-laptop:~/POPGroup/src$g++ -c *.cpp
barraf@barraf-laptop:~/POPGroup/src$ cd ..
barraf@barraf-laptop:~/POPGroup$  parocc  -o  mainAB.out  mainAB.cpp  A_reflection.cpp
POPGroup.cpp A.cpp ./src/*.o
barraf@barraf-laptop:~/POPGroup$ parocrun objmap ./mainAB.out
```

4.1. Limitations

Here are the actual limitations of the POPGROUP library:

- All methods must have different names. Different parameter types are not sufficiant to make a difference between methods with the same name. This limitation comes from the reflection library: A string with only the name of the method is needed to call it, so if we have two methods with the same name but different parameters, the reflection tool will not be able to know which one it should call.

- Methods with more than two parameters cannot be used with the POPGroup library. This comes from an implementation choice. This limitation can be removed, but you have to write all the existing collective methods with 3 parameters in input (actually, methods with one and two parameters are implemented).
- You cannot use methods with class parameters which are the same as the class. See section 5.1.1 for more information about it.
- The library is not usable with parallel objects in POP-C++, due to the compilation problems with the templates: see section 5.1.1 for more information about it.

Several syntax and structure rules are to also to respect. There are due to the implementation of the reflection library:

- All header files have to have the extension: *.hpp.*
- All C++ files have to have the extension: *.cpp* .
- Reflection files are to be called *Object_reflection.cpp* and *Object_reflection.hpp.*
- The directory *src* containing the sources of reflcpp modified has to be in the root directory of the application, where the reflection files are.

5. Conclusion

5.1.Encountered problems

5.1.1. Reflection in C++

We tried to find a library to add reflection in C++. The library *Relfex* [10] was interesting, and we tried to install it. You have to follow these steps to install and use it:

- Download the latest version on their CVS, or download it from the Releases page. The last version on the CVS is impossible to build, due to files missing (configure for example). We also took the last version on the release page.
- To build an installation, you have to enter the following commands

```
tar xvzf reflex.tar.gz
Cd reflex
./configure -prefix=/home/myuser/reflex-inst
make
make install
```

- After that, you have to create a dictionary. This is a C++ file, describing the Class we want to use reflectively. A tool is provided to do this, this is *genreflex*. This tool needs a .h file as parameter, and creates a C++ file which then will be used for creating a dynamic library in the memory. The first problem is that .ph files are not accepted as parameter. We rewrite our ph files into h files to permit the tool to create his C++ file.
- The dictionary created should then be compiled with the application. This part was the one which where we are blocked, because we need to compile our POP-C++ application with parocc, and not with gcc.

Another library was the XcppRefl. To install it, we had to install xml2, and create a shortcut to it in the directory /etc/bin, because the *./configure* try to find xml2.pc, and the current version named it libxml-2.0.pc. You also have to install the package Boost-dev (from the package manager for Ubuntu or at this website: [7]) and include it to compile it with *make*. After that, we had to find how this library functions, because there is no API furnished for it.

We used the following script (see appendix 8.10.1) and C code (see appendix 8.10.2) to make the needed modifications to compile the library. The new codegenerator.cpp file is in the sources of the CD, and the C file is under /sources/collective communication/tools. This file permit us to change the include path in every file of the library, which were primarly wrong.

After several corrections of the code made by the developers of the library, we are now blocked at a different place: the generation of the meta-classes, the classes which contain the information about the class which have to be called through reflection, is not correct. Here is an example of a correct generation of the class A:

```
#ifndef _A_H
#define _A_H

class A {
  public:
   A();
   void methodA1();
   void methodA2(int value);
   int methodA3();
   int methodA4(int value);

};
#endif
```

And his correct generation:

```
//gccxml: generate the xml file  A.xml
#ifndef CPP_REFLECT_A_H
#define CPP_REFLECT_A_H

#include "src/ArrayType.hpp"
#include "src/ClassType_tmpl.hpp"
#include "src/FundamentalType.hpp"
#include "src/PointerType.hpp"

#include "src/MemberFunctionDcl.hpp"
#include "A.hpp"
namespace reflcpp {

template <typename Bottom_TP, typename Der_TP, int N>
class Bases<Bottom_TP, Der_TP, A, N> : public BaseList<Bottom_TP, A > {};

struct A_strings {
       static const char name[];
       static const char m_methodA1[];
       static const char m_methodA2[];
       static const char m_methodA3[];
       static const char m_methodA4[];
};

template <>
class Members <A>
```

```
 : public MemberList<
        A,
        A_strings::name
        ,MemberFunctionDcl000<A, FundamentalType_tmpl<void>, &A::methodA1,
A_strings::m_methodA1>
        ,MemberFunctionDcl001<A,
FundamentalType_tmpl<void>,FundamentalType_tmpl<int>, &A::methodA2,
A_strings::m_methodA2>
        ,MemberFunctionDcl000<A, FundamentalType_tmpl<int>, &A::methodA3,
A_strings::m_methodA3>
        ,MemberFunctionDcl001<A, FundamentalType_tmpl<int>,FundamentalType_tmpl<int>,
&A::methodA4, A_strings::m_methodA4>
        >
{};
}
#endif
```

When we try to generate it, we get the following result:

```
//gccxml: generate the xml file  A.xml
#ifndef CPP_REFLECT_A_H
#define CPP_REFLECT_A_H

// modified by Frederic BARRAS to use local files
#include "src/ArrayType.hpp"
#include "src/ClassType_tmpl.hpp"
#include "src/FundamentalType.hpp"
#include "src/PointerType.hpp"

#include "src/MemberFunctionDcl.hpp"
#include "A.hpp"
namespace reflcpp {

template <typename Bottom_TP, typename Der_TP, int N>
class Bases<Bottom_TP, Der_TP, A, N> : public BaseList<Bottom_TP, A > {};

struct A_strings {
        static const char name[];
        static const char m_methodA1[];
        static const char m_methodA2[];
        static const char m_methodA3[];
        static const char m_methodA4[];
};

template <>
class Members <A>
 : public MemberList<
        A,
        A_strings::name
        >
{};
}
#endif
```

All the declaration of the function members (methods, for example: *methodA1*) of the class are not written in our file generation. This generation of files uses gccxml and codegenerator, a tool developed by the programmers of reflcpp. I tried to use the same gccxml version as them (version 0.6.0), and the actual latest version (0.7.0), but no changes were seen. After I received the code of the tool codegenerator, I found an issue. In the method which generates the code for public methods of a class, it was a test, to check if a method is public or not. The non-public methods are not generated. The test was made in a wrong way; it compared the declaration of the method with an empty string instead of the reserved word “public”. Some other changes were needed in the code generator: when a parameter is an array, the generated code had syntax errors. I had to add spaces between the different templates to make it able to be compiled.

For the compilation with POP-C++ (using parocc), the following errors occurred:

```
barraf@barraf-laptop:~/POPGroup$  parocc  -o  mainReflection.out  mainreflection2.cc
Integer.cpp Integer.ph POPGroup.cpp POPGroup.hpp
Warning: class unique identifier (classuid) for Integer is not specified.
Integer_reflection.hpp:36:    error:    could    not    convert    template    argument
‘&Integer__parocobj::Set’ to ‘void (Integer::*)(int)’
Integer_reflection.hpp:37:    error:    could    not    convert    template    argument
‘&Integer__parocobj::Get’ to ‘int (Integer::*)()’
Integer_reflection.hpp:38:    error:    could    not    convert    template    argument
‘&Integer__parocobj::Add’ to ‘void (Integer::*)(int)’
Integer_reflection.hpp:39:    error:    could    not    convert    template    argument
‘&Integer__parocobj::Add2’ to ‘void (Integer::*)(int, int)’
Integer_reflection.hpp:40:    error:    could    not    convert    template    argument
‘&Integer__parocobj::Wait’ to ‘void (Integer::*)(int)’
Integer_reflection.hpp:41:    error:    could    not    convert    template    argument
‘&Integer__parocobj::Sum’ to ‘int (Integer::*)(int*)’
Integer_reflection.hpp:42: error: template argument 3 is invalid
Integer_reflection.hpp:42: error: template argument 4 is invalid
Integer_reflection.hpp:42: error: template argument 5 is invalid
Integer_reflection.hpp:42: error: template argument 6 is invalid
Integer_reflection.hpp:42: error: template argument 7 is invalid
Integer_reflection.hpp:42: error: template argument 8 is invalid
```

During the preprocessing of the POP-C++ parser, a problem occurs to convert templates of the reflection with templates used in POP-C++. This issue is actually not corrected, due to a lack of time. The template in Integer_reflection is trying to convert the method of Integer_parocobj, and not the method from Integer which inherited from Interface. A clue could be linking the Integer_reflection with one of the temporary files created during a compilation of a POP-C++ program, like for example _paroc3_interface.ph.cc.

Actually, the following problems are not resolved:

- When a method is using its own class in parameters, the execution fails, due to a recursive call problem in the source code of the reflcpp library. (Example *in class A : methodA1(A a);*)
- When two methods have the same name, the application compilation fail, also when their parameters are different.
- Impossible to compile in POP-C++ with parallel objects.

An idea to simplify the implementation of POPGroup is to create non-modifiable groups. When creating the groups, a defined number of objects will be created in the group, and you will have no possibility to change this group. It will be no AddMember and RemoveMember for example. After some research, we saw that this solution doesn't simplify the problem. Our main issue is calling a method on the object, without knowing before the compilation time.

5.2. Possible improvements

Here are several improvements which can be done for the actual state of the library:

- Realize the precompilation script which generates the reflection files from the .ph files.
- Find a way to compile it in POP-C++, by finding how to link the methods in the *POPGroup.cpp*, *Object_reflection.cpp* and *Object_reflection.hpp* to the interface which inherits from the *Object.*
- Improve the complexity of the library. Until now, a broadcast is done with a complexity of N. By finding a solution like create a logical tree between the members of the group to propagate the method call, we can obtain a complexity of Log(N). Each member will have the knowledge about his neighborhood, and will send to his sons in the tree the method call received by his father. Another solution can be the call of synchronous methods in different threads.
- Improve the reflcpp library, by correcting the actual issues and adapt it to be compiled with POP-C++.

5.3.Personal conclusion

The conception of this library is correct; the only problem is the implementation of the reflection in C++. It was an interesting theme, which let a lot of improvements possible. Like find a way to distribute the messages among a structure like a tree between the objects in a POPGroup. Trying to correct the reflcpp library with its developers was an interesting experience, and I saw the importance of describing as clearly as possible the encountered problems.

This version of the POPGroup library will be slower in execution than the one which modifies the parser, due to all the work done by the reflcpp library. It has to check for the right method in its dictionary, then to have an implementation of the class, and finally call the method. It also has more limitations, like the number of arguments available.

The next chapter is the final conclusion of this project. It will talk about the planning and the project in general.

General conclusion

1. Personal

This was an interesting project with a lot of new fields to discover and study. It gave me a good idea how to plan several projects, the time needed to realize different tasks and the best method to keep a report up-to-date. One of the biggest difficulties was to choose when we should finish the first task and get into the second one.

2. Planning

The planning changed several times, due to the fact that at the beginning, we had no clue about collective communications. We realized two tasks, and the last two ones were cancelled, according to the decision of M. Kuonen (see meeting minutes from the 10/26/07).

For the first task, the planning was completely followed. All the tasks were realized on time, and we respected the schedule we had to use the UNM PHOENIX Cluster.

For the second task, I completely followed the planning until the 2nd of November. The task *implement the library* and *using reflection in POP-C++* took me the rest of the time at disposition (See 4.2 *Actual state of development* and 5.1 *encountered problems* in the second task for more information).

3. Thanks{common}

We'd like to thank all the persons who helped us for this project:

- the responsible professors in Fribourg:
    - Pierre Kuonen: *for his support and advices during the entire project and for giving us the possibility to accomplish it in Albuquerque.*
    - François Kilchoer: *for his support and advices during the entire project and for the report corrections.*
    - Jean-François Roche: *for his advices during the entire project and his support in organizational tasks.*
    - Guilherme Peretti Pezzi: *for his support and advices to solve technical issues with POP-C++.*
- The responsible externs :
    - Thuan-Anh Nguyen*: for his help to solve issues with POP-C++ on the cluster and for providing the lexer and grammar source files*
    - Barney Maccabe: *for giving us the possibility to come to Albuquerque and all the organizational tasks. And for giving us a good environment to work on our project.*
    - Rolf Riesen: *for his help to understand MPI and collective communications. For his advices during the second task.*

- the developers of the reflpcc library *for their support when we used their library and for their availability:*
    - Tharaka Devadithya (Indiana University, USA)
    - Kenneth Chiu (SUNY Binghamton, USA)
    - Wei Lu (Indiana University, USA)

The next chapter, after this conclusion, will be the appendixes. You will find every reference I used, the different websites which could be useful, and the code of the applications developed.

Appendix

1. Definitions

Benchmarking

Performance test of a system.

Broker (POP-C++)

Part on the remote machine which receives the messages from the network and translate them into methods

Cluster

Supercomputer which contains several nodes composed from memory and processor.

Combox (POP-C++)

Part which is responsible for the communication between the remote and local part of an object (with sockets).

CVS *Concurrent Versions System*

System which permits to depose and get different versions of files and applications.

Deviation, standard

Mathematical operation which indicates the typical deviation of values compared to their average.

Grid group

Name of the group which works on POP-C++ and Gris at the EIA-FR Fribourg.

HPC *High Performance Computing*

Field of the distributed computer science which focus on applications which needs a lot of resources to for calculus.

InfiniBand

Computer communication bus with low latency.

Interface (POP-C++)

Part on the local machine which is responsible to get the method call of the main program and to send them to the remote part, through the buffer and combox.

JobManager (POP-C++)

Part which is responsible for finding machines which require the minimal resources asked by an object.

Linux

Operating System based on UNIX.

Meta-class

Class which contains information about a class, like its name, its method name, etc

MPI *Message Passing Interface*

Standard for collective communications.

NFS *Network File System*

File system distributed among several computers. Every computer sees the same file system.

parclass (POP-C++)

Parallel class describing an object which can be stored remotely.

Parser

Application which goes through a text and generate an output regarding this text.

POE *Parallel Operating Environment.*

Environment designed for parallel applications, with several computers, clusters.

POP-C++

Parallel language, developed at the EIA-FR Fribourg.

Process

Set of executable instructions.

Reflection

Principle which permit to cast a string into a method call.

Runtime

Time during the execution of a program.

SPMD *Simple Program Multiple Data*

Type of program which has one source code, and several behaviors among the data

Tick

Instruction cycle of a processor.

UML *Unified Modeling language*

Analysis and conception graphical language.

UNM *University of New Mexico*

University of the city of Albuquerque, New Mexico, USA, where this project was realized.

2. References

[1] Translation from http://www.eif.ch/gestionprojets/private/rechercher.jsp?inoid=1620

[2] Summer 2007 semester report « WSL with POP-C++ », available in the library from the EIAFR.**Erreur ! Signet non défini.**

[3] http://www.iis.sinica.edu.tw/~kathy/vcstl/templates.htm

[4] Diploma work report: "Improving POP-C++ for HPC" , Manuel Schrag, November 2007, available on the CD of this project

[5] Manuel reference for POP-C++, user guide, version 1.1, EIAFR Fribourg, Gridgroup http://www.eif.ch/gridgroup/popc/docs/manual.pdf

[6] http://www.extreme.indiana.edu/reflcpp/ : website for the reflcpp library

[7] http://www-eleves-isia.cma.fr/documentation/BoostDoc/boost_1_29_0/more/download.html : website for Boost

[8] http://home.hefr.ch/schram04/diplomawork/: website for this project

[9] http://www.cs.unm.edu/~riesen/lesson_10.pdf : Illustration of MPI functions

[10]http://seal-reflex.web.cern.ch/seal-reflex/index.html :website of the Reflex library

3. Links

http://www.mhpcc.edu/training/workshop/mpi/MAIN.html : an English explanation about MPI concepts and structures.

www.**mpi**-forum.org : official English forum about MPI, contains the official documents and releases.

http://en.wikipedia.org/wiki/Computer_cluster : Explanation of what is a cluster computer.

http://www.eif.ch/gridgroup/popc/docs/manual.pdf : Manual Reference for POP-C++.

http://www-unix.mcs.anl.gov/mpi/ : Explanation of MPI.

http://www.llnl.gov/computing/tutorials/mpi/man/MPI_Init.txt: Explanation of *MPI_Init* function.

http://fresh.t-systems-sfr.com/unix/src/privat2/openmpi-1.2.3.tar.gz:a/openmpi-1.2.3/ompi/mpi/man/man3/MPI_Comm_spawn.3 : Explanation of MPI_Comm_spawn function.

http://en.wikipedia.org/wiki/Reflection_(computer_science): explanation about Reflection concept

http://www.iis.sinica.edu.tw/~kathy/vcstl/templates.htm: a good explanation about templates in C++

http://seal-reflex.web.cern.ch/seal-reflex/index.html : Website of a non-used library for this project.

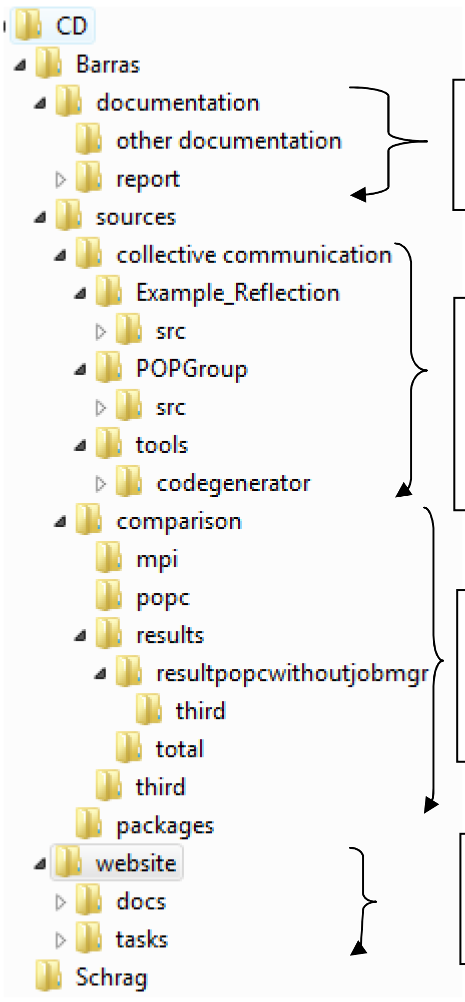
http://www.extreme.indiana.edu/reflcpp/: Website of the library realizing reflection for C++ used for this project.

http://www-eleves-isia.cma.fr/documentation/BoostDoc/boost_1_29_0/more/download.html: Website for the download and installation of the Boost tool.

4. Figures

5. Tables

6. CD content



```
CD
  Barras
    documentation
      other documentation
      report
    sources
      collective communication
        Example_Reflection
          src
        POPGroup
          src
        tools
          codegenerator
      comparison
        mpi
        popc
        results
          resultpopcwithoutjobmgr
            third
          total
        third
      packages
    website
      docs
      tasks
  Schrag
```

The documentation contains the report and other usefuls documents, like the article about reflcpp.

The collective communication sources contain the example of reflection with A and B, and the POPGroup library. Tools contains the code generator modified and some other useful tools.

The comparison sources contains the source codes under mpi and popc. The results are under the directory result (see the .xls files).

The website is under the directory website. Open Index.php to see it.

7. Planning

Please refer to the webpage to see the final planning and the different versions.

8. Sources

8.1.runTests

```
#!/bin/bash
# Runs the test with different values
# remove the old compiled files
rm -f objmap emptymainmpi emptymainpopc *.o mpi/*.out \
popc/*.o popc/objmap popc/*.obj popc/*.out
./compilations
#USAGE : startBenchmarks resultfile NBProcesses NBObjects NbNodes NbChildren
#1 2 4 6 8 10 12 14 16
```

```
for i in 1 2 4 6 8 10 12 14 15
do
prefix=$i.Nodes
suffix=$(date +%b%d%y)
filename=$prefix.$suffix
./startBenchmarks resultnn/$filename $i $i $i $i
done
```

8.2.startBenchmarks

```
#!/bin/bash
# Check parameters
#
if [ $# -ne 5 ]; then
  echo USAGE : startBenchmarks resultfile NBProcesses NBObjects NbNodes NbChildren
  exit 1
fi
echo  ---------------------------------------------------------
echo !!!Please start the SXXparoc deamon and the mpi deamon before launching this
script!!!
echo  ---------------------------------------------------------

-------------------------------MEASURES-----------------
#measure of the runtime executables
echo TEST RESULTS>$1
for i in 1 2 3 4 5 6 7 8 9 10
do
   #MPI
echo  ---------------------------------------------------------->>$1
echo  MPI executable>>$1
echo  ---------------------------------------------------------->>$1
{ time mpirun -n $2 -hostfile hostfile ./emptymainmpi; } 2>> $1
   #POP-C++
echo  ---------------------------------------------------------->>$1
echo  POPC executable>>$1
echo  ---------------------------------------------------------->>$1
{ time parocrun objmap ./emptymainpopc; } 2>> $1
done

#measure of the object/process creation
for j in 1 2 3 4 5 6 7 8 9 10
do
   #MPI Static
echo  ---------------------------------------------------------->>$1
echo  MPI Initialization static>>$1
echo  ---------------------------------------------------------->>$1
   mpirun -n $2 -hostfile hostfile mpi/mainstatic.out>>$1
 #MPI Dynamic
echo  ---------------------------------------------------------->>$1
echo  MPI Initialization dynamic>>$1
echo  ---------------------------------------------------------->>$1
   mpirun -n 1 -hostfile hostfile mpi/maindynamic.out $5 >>$1
   #POP-C++ without jobManager
echo  ---------------------------------------------------------->>$1
echo  POP-C++ Initialization without JobManager>>$1
echo  ---------------------------------------------------------->>$1
   parocrun popc/objmap popc/main.out $3 false>>$1
   sleep 10
#echo canceled, cluster cannot work without jobManager >>$1
   #POP-C++ with jobManager
echo  ---------------------------------------------------------->>$1
echo  POP-C++ initialization with jobManager>>$1
echo  ---------------------------------------------------------->>$1
   parocrun popc/objmap popc/main.out $3 true>>$1
   sleep 10
```

```
done

echo  ---------------------------------------------------------->>$1
echo  Time Measure>>$1
echo  ---------------------------------------------------------->>$1
parocrun popc/objmap popc/cycleTest.out >>$1
```

8.3.myObject.ph

```
#include <string.h>
/*  Header  file  from  the  object  MyObject,  used  in  the  measure  from  POP-C++
initialization */
parclass MyObject
{
public:
       MyObject(int wanted, int minp) @{ power= wanted ?: minp;};
       MyObject(paroc_string machine) @{ od.url(machine);};
       ~MyObject();

};
```

8.4.myObject.cc

```
#include <stdio.h>
#include "myObject.ph"
//#include <unistd.h>
#ifdef _PAROC_
#define printf rprintf
#endif
/*  CC  file  from  the  object  MyObject,  used  in  the  measure  from  POP-C++
initialization
   The prints are here to test if every object is really created at the good place.
They
   have to be put in comment for the real time of initialization test
 */

MyObject::MyObject(int wanted, int minp)
{
//rprintf("Object MyObject on %s\n",(const char *)paroc_system::GetHost());
}

MyObject::MyObject(paroc_string machine)
{
      // rprintf("Object MyObject on %s\n",(const char *)paroc_system::GetHost());
}

MyObject::~MyObject()
{
  //printf("Destroying the object...\n");
}

@pack( MyObject);
```

8.5.Main.cc

```
#include "myObject.ph"
#include <iostream.h>
#include <unistd.h>
//#include <time.h>
#include "cycle.h"
/* main file, used in the measure from POP-C++ initialization
   It takes arguments to know the number of objects to create
   and if you use the Jobmanager or not.
*/
int main(int argc, char **argv)
```

```
{
  if (argc != 3) {
     printf ("Usage: init nbObjects boolwithjobMgr \n");
     return 1;
  }
  int nbObjects = atoi(argv[1]);
  bool withJobMgr= strcmp(argv[2],"true")==0?true:false;
  ticks t0,t1;
  double elapsedTime=-1;
  printf("nbObjects = %d, and job = %d \n",nbObjects,withJobMgr);

char* myNodes[16]={"phx0","phx1","phx2",
              "phx3","phx4","phx5","phx6",
              "phx7","phx8","phx9","phx10","phx11",
              "phx13","phx14","phx_head","phx12"};
  try {
     t0 = getticks();
     if (withJobMgr){
        for (int i=0;i<nbObjects;i++){
           MyObject o1(60,40);
        }
     }
     else{
        for (int j=0;j<nbObjects;j++){
           MyObject o2(myNodes[j]);
        }
     }
     t1 = getticks();
     elapsedTime= elapsed(t1,t0);
     printf("elapsed   time   for   all   objects   creation=   %f   (with   job   :
%d\n",elapsedTime,withJobMgr);
  }
  catch (paroc_exception *e)
    {
     cout<<"Object creation failure"<<"\n";
     return -1;
    }
  return 0;
}
```

8.6.Mainstatic.c

```
/* MPI creation of processes */
#include <stdio.h>
#include <mpi.h>
#include "cycle.h"
//#include <unistd.h>

int main (argc, argv)
     int argc;
     char *argv[];
{
  int myRank,i;
  double localMax, globalMax;
  ticks t0,t1;

  t0 = getticks();
  MPI_Init (&argc, &argv);/* starts MPI */
  t1 = getticks();
 /*Get the time to make the init*/
  localMax= elapsed(t1,t0);
  globalMax= -1;
  MPI_Comm_rank(MPI_COMM_WORLD, &myRank);
 /* Make a reduce to get the highest value between all the processes*/
```

```
  MPI_Reduce(&localMax, &globalMax,1,MPI_DOUBLE,MPI_MAX,0,MPI_COMM_WORLD);
  if (myRank==0){
     printf(  "Highest  value  between  all  the  processes  for  MPI_Init  :%f
\n",globalMax);
  }
  //char hostname[256];
  //gethostname(hostname,256);
  //printf("i'm on host %s \n",hostname);
MPI_Finalize();
  return 0;
}
```

8.7.Maindynamic.c

```
/* MPI creation of processes */
#include <stdio.h>
#include <mpi.h>
#include "cycle.h"

int main (argc, argv)
     int argc;
     char *argv[];
{
if (argc != 2) {
     printf ("Usage: maindynamic Nbchildren \n");
     return 1;
  }
  int myRank,i;
  double localMax, globalMax;
  ticks t0,t1;
  MPI_Init (&argc, &argv);/* starts MPI */
  MPI_Comm_rank(MPI_COMM_WORLD, &myRank);
  /* Testing of the MPI_Comm_spawn method*/
  int nbChild=atoi(argv[1]);
  //printf("NB Children : %d\n",nbChild);
  int errcodes[nbChild];
    MPI_Comm intercomm;
    char* myArgv[]={"-hostfile","hostfile",NULL};
t0 = getticks();
  if   (0!=MPI_Comm_spawn(   "mpi/child.out",myArgv,   nbChild,   MPI_INFO_NULL,   0,
MPI_COMM_WORLD, &intercomm, errcodes )){
      return 1;
}
t1 = getticks();
  if (myRank==0){
  localMax= elapsed(t1,t0);
  printf("childs  called  by  me,  i'm  rank  %d\ntime  to  make  MPI_Comm_spawn  :
%f\n",myRank,localMax);
}
MPI_Finalize();
  return 0;
}
```

8.8.Child.c

```
#include <stdio.h>
#include <mpi.h>
#include "cycle.h"
//#include <unistd.h>
/*Child process which will be created, do nothing else as just call MPI_Init and
return*/
int main (argc, argv)
     int argc;
     char *argv[];
{
  MPI_Init(&argc, &argv);
  //TO Test if children are created
  //int myRank;
  //MPI_Comm_rank(MPI_COMM_WORLD, &myRank);
  //char hostname[256];
  //gethostname(hostname,256);
  //printf("i'm the child number %d on %s \n",myRank,hostname);
  MPI_Finalize();
}
```

8.9.Code source for total initialization

8.9.1. Objectpop.ph

```
#include <string.h>
/*  Header  file  from  the  object  MyObject,  used  in  the  measure  from  POP-C++
initialization */
parclass ObjectPop
{
public:
       ObjectPop(int wanted, int minp) @{ power= wanted ?: minp;};
       ObjectPop(paroc_string machine) @{ od.url(machine);};
       ~ObjectPop();

};
```

8.9.2. Objectpop.cc

```
#include <stdio.h>
#include "objectpop.ph"
#include <stdlib.h>
#ifdef _PAROC_
#define printf rprintf
#endif
/*  CC  file  from  the  object  MyObject,  used  in  the  measure  from  POP-C++
initialization
   The prints are here to test if every object is really created at the good place.
They
   have to be put in comment for the real time of initialization test
 */

ObjectPop::ObjectPop(int wanted, int minp)
{
       //rprintf("Object MyObject on %s\n",(const char *) paroc_system::GetHost());
        //exit(-1);
}

ObjectPop::ObjectPop(paroc_string machine)
{
        //rprintf("Object MyObject on %s\n",(const char *)paroc_system::GetHost());
       //exit(-1);
}

ObjectPop::~ObjectPop()
{
  //printf("Destroying the object...\n");
}

@pack( ObjectPop);
```

8.9.3. Mainpop.cc

```
#include "objectpop.ph"
#include <iostream.h>
#include <unistd.h>
#include <stdlib.h>
/* main file, used in the measure from POP-C++ initialization
   It takes arguments to know the number of objects to create
   and if you use the Jobmanager or not.
*/
int main(int argc, char **argv)
{
  if (argc != 3) {
     printf ("Usage: init nbObjects boolwithjobMgr \n");
     return 1;
  }
  int nbObjects = atoi(argv[1]);
  bool withJobMgr= strcmp(argv[2],"true")==0?true:false;
  //printf("nbObjects = %d, and job = %d \n",nbObjects,withJobMgr);
//  char* myNodes[16]={"localhost","localhost","localhost","localhost",
//                   "localhost","localhost","localhost","localhost",
//                   "localhost","localhost","localhost","localhost",
//                   "localhost","localhost","localhost","localhost"};
char* myNodes[16]={"phx0","phx1","phx2",
              "phx3","phx4","phx5","phx6",
              "phx7","phx8","phx9","phx10","phx11",
              "phx13","phx14","phx_header","phx12"};
  try {
     if (withJobMgr){
        for (int i=0;i<nbObjects;i++){
           ObjectPop o1(60,40);
        }
     }
     else{
        for (int j=0;j<nbObjects;j++){
           ObjectPop o2(myNodes[j]);
        }
     }
   //exit(-1);
  }
  catch (paroc_exception *e)
    {
     cout<<"Object creation failure"<<"\n";
     return -1;
    }
  return 0;
}
```

8.9.4. Mainstatic.c

```
/* MPI creation of processes */
//#include <stdio.h>
#include <mpi.h>
#include <stdlib.h>

int main (argc, argv)
     int argc;
     char *argv[];
{

  MPI_Init (&argc, &argv);/* starts MPI */
  exit(-1);
MPI_Finalize();
  return 0;
}
```

8.9.5. Maindynamic.c

```
/* MPI creation of processes */
//#include <stdio.h>
#include <stdlib.h>
#include <mpi.h>

int main (argc, argv)
     int argc;
     char *argv[];
{
//if (argc != 2) {
//     printf ("Usage: maindynamic Nbchildren \n");
//     return 1;
//  }
  //int myRank,i;
  MPI_Init (&argc, &argv);/* starts MPI */
  //MPI_Comm_rank(MPI_COMM_WORLD, &myRank);
  /* Testing of the MPI_Comm_spawn method*/
  int nbChild=atoi(argv[1]);
  //printf("NB Children : %d\n",nbChild);
char* myArgv[]={"-hostfile","hostfile",NULL};
  int errcodes[nbChild];
    MPI_Comm intercomm;
  if  (0!=MPI_Comm_spawn(  "./childdynamic.out",  myArgv,  nbChild,  MPI_INFO_NULL,  0,
MPI_COMM_WORLD, &intercomm, errcodes )){
       return 1;
}

  //if (myRank==0){
  //localMax= elapsed(t1,t0);
  //printf("childs  called  by  me,  i'm  rank  %d\ntime  to  make  MPI_Comm_spawn  :
%f\n",myRank,localMax);
//}
  exit(-1);
MPI_Finalize();
}
```

8.9.6. Childdynamic.c

```
//#include <stdio.h>
#include <stdlib.h>
#include <mpi.h>

/*Child  process  which  will  be  created,  do  nothing  else  as  just  call  MPI_Init  and
return*/
int main (argc, argv)
     int argc;
     char *argv[];
{
  MPI_Init(&argc, &argv);
  //exit(-1);
  MPI_Finalize();
}
```

8.10. Source code part 2

8.10.1. Copydir

```
#! /bin/bash

./modifyFiles.out
 #copy the new files in the correct directory
cp -ru $1/ ./
```

8.10.2. modifyFiles

```
#include <stdio.h>
#include <string.h>

int main()
{
const int LINE_MAX=1000;
const int MAX_FILENAME=100;

const char* HPP= ".hpp>";
FILE* outFile;
FILE* inFile;
FILE* listFile;

char currentLine[LINE_MAX];
char fileName[MAX_FILENAME];
char outFileName[MAX_FILENAME];
outFileName[0]='s';
outFileName[1]='r';
outFileName[2]='c';

listFile=fopen("testin","r");
if(listFile!=NULL)
{
  while(!feof(listFile))
  {
    fscanf(listFile,"%s\n",fileName);
    printf("converting file :[%s]\n",fileName);

    char* localName= strrchr(fileName,'/');
    printf("localName : [%s]\n",localName);

    outFileName[3]='\0';
    strcat(outFileName,localName);
    printf("outFile name = [%s]\n",outFileName);

    if ((inFile = fopen(fileName,"r"))!=NULL)
    {
      if ((outFile=fopen(outFileName,"w"))!=NULL)
      {
        while(!feof(inFile))
        {
          currentLine[0]='\0';
          if (fgets(currentLine, LINE_MAX, inFile )!=NULL) //EOF encountered
          {
            printf("converting line:[%s]\n",currentLine);
            if (strstr(currentLine, HPP)!=NULL)
            {
              char* p;
             if ((p=strstr(currentLine, "<src/"))!=NULL)
              {
                p[0] = ' '; p[1] = ' '; p[2] = ' '; p[3] = ' ';
                p[4]='"';
                p=strstr(currentLine, ">");
                *p = '"';
              }
              else
              {
                if ((p=strstr(currentLine, "<"))!=NULL)
                {
                  p[0]='"';
                  p=strstr(currentLine, ">");
                  *p = '"';
                }
              }
            } //if
          }
          //printf("%s", currentLine);
          fprintf(outFile,"%s", currentLine);
        } //while
      } //if
      else printf("Unable to open outfile %s\n",outFileName);
    } //if
    else printf("Unable to open infile %s\n",fileName);
    printf("closing files IN and OUT\n");
    fclose(inFile);
    fclose(outFile);
  }//while
  printf("closing files ListFile\n");
  fclose(listFile);
} //if
  else printf("cannot open :%s or testout\n",listFile);

  //Copy of the needed files
char* directory= strchr(fileName,'/');

} //main
```

8.10.3. A.cpp

```
#include "A.hpp"
#include <stdio.h>

A::A(){
 printf("Object A constructed\n");
}

void A::methodA1(){
 printf("\nCall method1 on Object A \n");
}

void A::methodA2(int value){
 printf("Call method2 on Object A with value=[%d] \n",value);
}

int A::methodA3(){
 printf("Call method3 on Object A \n");
 return 14;
}

int A::methodA4(int value){
 printf("Call method4 on Object A with value=[%d] \n",value);
 return value*2;
}
int A::methodA5(int value1, int value2){
printf("Call method5 on Object A with value1=[%d] and
value2=[%d]\n",value1,value2);
return value1+value2;
}

void A::methodA6(int value1, int value2){
printf("Call method6 on Object A with value1=[%d] and
value2=[%d]\n",value1,value2);
}

/*
void A::methodA7(A a){
printf("Call methodA7 on A with non primitive parameter\n");
}
*/
```

8.10.4. A.hpp

```
#ifndef _A_H
#define _A_H

class A {
  public:
   A();
   void methodA1();
   void methodA2(int value);
   int methodA3();
   int methodA4(int value);
   int methodA5(int value1, int value2);
   void methodA6(int value1, int value2);
  //void methodA7(A a);

};

#endif
```

8.10.5. B.cpp

```
#include <stdio.h>
#include <string.h>
#include "B.hpp"

#include "src/ClassType.hpp"
//Includes made by the main.cpp in examples.
//#include "src/ClassType_tmpl.hpp"
#include "src/BoundClassType_tmpl.hpp"
#include "src/Exceptions.hpp"
#include "A_reflection.hpp"

using namespace reflcpp;

B::B(){
 printf("Object B constructed\n");
}

void B::doMethod(A &object){

 printf("\nCall doMethod on Object B \n");
 ClassType ct = ClassType::getClass("A");
 std::string className = ct.name();
 printf("Name of the class : [%s]\n",className.c_str());

 //first method call, without return or parameter
 printf("Get an instance of the methode 'methodA1' of A\n");
 MemberFunction mf =ct.getMemberFunction("methodA1");
 printf("Invoke 'methodA1' on A\n");
 mf.invoke<void>(&object);

 //second method call, with parameter and no returns
 printf("Get an instance of the methode 'methodA2' of A\n");
 MemberFunction mf2 =ct.getMemberFunction("methodA2");
 printf("Invoke 'methodA2' on A, (value should be [4])\n");
 int myValue[1];
 myValue[0]=4;
 //Arguments params; //FundamentalType
  //params.addValueArgument(*myValue);
 printf("\n !!! invocation !!!");
 mf2.invoke<void>(&object,*myValue);

 //third method call, with return and no parameter
 printf("Get an instance of the methode 'methodA3' of A\n");
 MemberFunction mf3 =ct.getMemberFunction("methodA3");
 printf("Invoke 'methodA3' on A\n");
 int resultMethod3;
 resultMethod3=mf3.invoke<int>(&object);
 printf("Value of return method 3 (should be [14]) = [%d]\n",resultMethod3);

 //fourth method call, with return and parameter
 printf("Get an instance of the methode 'methodA4' of A\n");
 MemberFunction mf4 =ct.getMemberFunction("methodA4");
 printf("Invoke 'methodA4' on A, (value should be 4*2=[8])\n");
 int myValue4[1];
 int resultMethod4;
 myValue4[0]=4;
 resultMethod4=mf4.invoke<int>(&object,*myValue4);
 printf("Value of return method 4 (should be [8]) = [%d]\n",resultMethod4);

 //fifth method call, with return and  2 parameters
 printf("Get an instance of the methode 'methodA5' of A\n");
 MemberFunction mf5 =ct.getMemberFunction("methodA5");
 printf("Invoke 'methodA5' on A, (value should be 4+8=[12])\n");
 int myValue5[1];
 int resultMethod5;
 myValue5[0]=4;
 int myValue5bis[1];
 myValue5bis[0]=8;
 resultMethod5=mf5.invoke<int>(&object,*myValue5,*myValue5bis);
 printf("Value of return method 5 (should be [12]) = [%d]\n",resultMethod5);

/*


 printf("Get an instance of the methode 'methodA7' of A\n");
 MemberFunction mf7 = ct.getMemberFunction("methodA7");
 printf("Invoke 'methodA7' on A\n");
 A paramA;
 mf7.invoke<void>(&object,paramA);
*/
}
```

8.10.6. B.hpp

```
#ifndef _B_H
#define _B_H
#include "A.hpp"

class B {
  public:
    B();
    void doMethod(A &object);

};

#endif
```

8.10.7. A_reflection.cpp

```
//gccxml: generate the xml file  A.xml
#include "A_reflection.hpp"

namespace reflcpp {

namespace {
    ClassType_tmpl<A> inserter;
}

const char A_strings::name[] = "A";
const char A_strings::m_methodA1[] = "methodA1";
const char A_strings::m_methodA2[] = "methodA2";
const char A_strings::m_methodA3[] = "methodA3";
const char A_strings::m_methodA4[] = "methodA4";
const char A_strings::m_methodA5[] = "methodA5";
const char A_strings::m_methodA6[] = "methodA6";
} // namespace reflcpp
```

8.10.8. A_reflection.hpp

```
//gccxml: generate the xml file  A.xml
#ifndef CPP_REFLECT_A_H
#define CPP_REFLECT_A_H

#include "src/ArrayType.hpp"
#include "src/ClassType_tmpl.hpp"
#include "src/FundamentalType.hpp"
#include "src/PointerType.hpp"
#include "src/MemberFunctionDcl.hpp"
#include "A.hpp"
namespace reflcpp {

template <typename Bottom_TP, typename Der_TP, int N>
class Bases<Bottom_TP, Der_TP, A, N> : public BaseList<Bottom_TP, A > {};


struct A_strings {
       static const char name[];
       static const char m_methodA1[];
       static const char m_methodA2[];
       static const char m_methodA3[];
       static const char m_methodA4[];
       static const char m_methodA5[];
       static const char m_methodA6[];
};

template <>
class Members <A>
 : public MemberList<
       A,
       A_strings::name
       ,MemberFunctionDcl000<A, FundamentalType_tmpl<void >, &A::methodA1,
A_strings::m_methodA1 >
       ,MemberFunctionDcl001<A, FundamentalType_tmpl<void >,FundamentalType_tmpl<int
>, &A::methodA2, A_strings::m_methodA2 >
       ,MemberFunctionDcl000<A, FundamentalType_tmpl<int >, &A::methodA3,
A_strings::m_methodA3 >
       ,MemberFunctionDcl001<A, FundamentalType_tmpl<int >,FundamentalType_tmpl<int
>, &A::methodA4, A_strings::m_methodA4 >
       ,MemberFunctionDcl002<A, FundamentalType_tmpl<int >,FundamentalType_tmpl<int
>,FundamentalType_tmpl<int >, &A::methodA5, A_strings::m_methodA5 >
       ,MemberFunctionDcl002<A, FundamentalType_tmpl<void >,FundamentalType_tmpl<int
>,FundamentalType_tmpl<int >, &A::methodA6, A_strings::m_methodA6 >
       >
{};

}

#endif
```

8.10.9. mainAB.cpp

```
#include "B.hpp"
#include "A.hpp"
#include "POPGroup.cpp"
```

```
int main(){
   printf("\nstart of the Test\n");
   A a;
   B b;
   b.doMethod(a);
}
```

8.10.10. main.cpp

```
//#include "B.hpp"
#include "A.hpp"
#include "POPGroup.cpp"
//#include "Integer.ph"
#include <iostream>

int main(int arc, char* argv[]){
   try{

   printf("\n\n !!! Test with POPGroup !!! \n\n");
   printf("-----------Creation, add and remove---------\n");
   A a;
   A a2;
   A a3;
   A a4;
   POPGroup<A> myGroup("A");

   myGroup.add(a);
   myGroup.add(a2);
   myGroup.add(a3);
   myGroup.add(a4);
   printf("Size of the group (should be [4]) : [%d]\n",myGroup.getSize());

   myGroup.removeAt(3);
   printf("Size of the group (should be [3]) : [%d]\n",myGroup.getSize());
   A atab[4];
   myGroup.add(atab,4);
   printf("Size of the group (should be [7]) : [%d]\n",myGroup.getSize());
   myGroup.remove(atab,4);
   printf("Size of the group (should be [3]) : [%d]\n",myGroup.getSize());

   POPGroup<A> myGroup2("A");
   myGroup2.add(a);
   myGroup.merge(myGroup2);
   printf("Size of the group (should be [4]) : [%d]\n",myGroup.getSize());
//Group merging

 /* broadcast*/
   printf("----------BROADCASTS----------\n");
   myGroup.broadcast("methodA1");
   myGroup.broadcast("methodA2",12);
   myGroup.broadcast("methodA6",22,33);
/*scatter*/
printf("----------SCATTERS----------\n");
   int scattarg1[myGroup.getSize()];
   int scattarg2[myGroup.getSize()];
   for (int i=0;i<myGroup.getSize();i=i+1){
      scattarg1[i]=i;
      scattarg2[i]=10+i;
   }
   myGroup.scatter("methodA2",scattarg1,myGroup.getSize());
   myGroup.scatter("methodA6",scattarg1,scattarg2,myGroup.getSize());
/*broadcastGather*/
   printf("----------BROADCASTGATHERS----------\n");
   int gatherResult[myGroup.getSize()];
   myGroup.broadcastGather("methodA3",gatherResult);
   for (int i=0;i<myGroup.getSize();i=i+1)
```

```
      printf("result from broadcastgather(0 param) (should be [%d]) for object [%d]
: [%d]\n",14,i,gatherResult[i]);
   myGroup.broadcastGather("methodA4",14,gatherResult);
   for (int i=0;i<myGroup.getSize();i=i+1)
      printf("result from broadcastgather(1 param) (should be [%d]) for object [%d]
: [%d]\n",14*2,i,gatherResult[i]);
   myGroup.broadcastGather("methodA5",23,43,gatherResult);
   for (int i=0;i<myGroup.getSize();i=i+1)
      printf("result from broadcastgather(2 params) (should be [%d]) for object
[%d] : [%d]\n",23+43,i,gatherResult[i]);
/*broadcastReduce*/
printf("----------BROADCASTREDUCES----------\n");
   int reduceResult[1];
   myGroup.broadcastReduce("methodA3",reduceResult,0);//POPGROUP_OPERATION_MAX
   printf("result from broadcastreduce(0 param) (shoudld be [%d]) for Group :
[%d]\n",14,reduceResult[0]);
   myGroup.broadcastReduce("methodA4",14,reduceResult,0);
   printf("result from broadcastreduce(1 param) (shoudld be [%d]) for Group :
[%d]\n",14*2,reduceResult[0]);
   myGroup.broadcastReduce("methodA5",23,43,reduceResult,0);
   printf("result from broadcastreduce(2 param) (shoudld be [%d]) for Group :
[%d]\n",23+43,reduceResult[0]);

/*scattergather*/
printf("----------SCATTERGATHERS----------\n");

myGroup.scatterGather("methodA4",scattarg1, gatherResult, myGroup.getSize());
for (int i=0;i<myGroup.getSize();i=i+1)
      printf("result from scattergather(1 param) (should be [%d]) for object [%d] :
[%d]\n",scattarg1[i]*2,i,gatherResult[i]);
myGroup.scatterGather("methodA5",scattarg1,scattarg2,
gatherResult,myGroup.getSize());
for (int i=0;i<myGroup.getSize();i=i+1)
      printf("result from scattergather(2 params) (should be [%d]) for object [%d]
: [%d]\n",scattarg1[i]+scattarg2[i],i,gatherResult[i]);
/*scatterreduce*/
printf("----------SCATTERREDUCES----------\n");
int scatterReduceResult[1];
myGroup.scatterReduce("methodA4",scattarg1,scatterReduceResult,myGroup.getSize(),0)
;
printf("result from scatterreduce(1 param) (shoudld be [%d]) for Group :
[%d]\n",2*2,scatterReduceResult[0]);
myGroup.scatterReduce("methodA5",scattarg1,scattarg2,
scatterReduceResult,myGroup.getSize(),0);
 printf("result from scatterreduce(2 param) (shoudld be [%d]) for Group :
[%d]\n",13+3,scatterReduceResult[0]);


//printf("----------BROADCAST with objects as parameters----------\n");
//A argA;
//myGroup.broadcast("methodA7",argA);
printf("----------BROADCASTGATHER with objects as return----------\n");
A aReturn[myGroup.getSize()];
myGroup.broadcastGather("methodA8",aReturn);
//myGroup.broadcastGather("methodA9",argA,aReturn);

printf("------ emptying group---------\n");
 myGroup.emptyGroup();
 printf("is Group now empty ? [%s]\nsize of the group (should be
[0])=%d\n",myGroup.isEmpty()==0?"false":"true",myGroup.getSize());

    } catch (RankException error) {
       printf("%s\n",error.getMessage());
    }
 return 0;
```

```
}

//Add objects in parameter
//Add array of objects in parameter
//Add array of primitive types in parameter

//empty group
/*
int testWithInteger()
{
  try {
    Integer o1(60,40);
    Integer o2(paroc_system::GetHost());
    Integer o3(100,20);

   // Create an empty group
    POPGroup<Integer> myGroup("Integer");

   // Adding 3 Integer objects to the group
    myGroup.add(o1);
    myGroup.add(o2);
    myGroup.add(o3);

   // Broadcast
    char* method="Set";
    myGroup.broadcast(method,3);

    int res[myGroup.getSize()];
   // Gather
   method="Get";
    myGroup.broadcastGather(method,res);


    int val[myGroup.getSize()];
    for(int i=0;i<myGroup.getSize();i++){
      val[i]=i;
    }
   // Scatter
    method="Set";
    myGroup.scatter(method,val);

   // Different Reduce operations
    method="Get";
    int resultReduce[1];
    myGroup.broadcastReduce(method,resultReduce,POPGroup<Integer>::OPERATION_MAX);
    myGroup.broadcastReduce(method,resultReduce,POPGroup<Integer>::OPERATION_MIN);
    myGroup.broadcastReduce(method,resultReduce,POPGroup<Integer>::OPERATION_OR);

   // Preparing array of remote objects
    Integer tmp[4];
    tmp[0].Set(5);
    tmp[1].Set(6);
    tmp[2].Set(7);
    tmp[3].Set(8);

   // Broadcast operation with remote object in parameter
    method="Add";

    //myGroup.broadcast(method,&o1); //TO VERIFY, MAYBE IMPOSSIBLE

   // Scatter operation with array of remote objects in parameter
    method="Add";
    //myGroup.scatter(method,tmp);

   // Remove single members from group
```

```
     myGroup.removeAt(2);
     myGroup.removeAt(1);
     myGroup.removeAt(0);

    // Adding multiple Integer objects to the group
     myGroup.add(tmp, 4);

     /*res[myGroup.getSize()];
     int x[5000];
     int y[5000];
     int *a[2] = {x,y};
     for(int i=0;i<5000;i++) {
       x[i]=i;
       y[i]=5;
     }
    // Broadcast operation with array as parameter
     method="Sum";
     myGroup.broadcast(method,x);

    // Scatter operation with array as parameter (only 2 objects will be invoked)
     myGroup.scatter(method,a,res);*/
/*
    // Removing multiple objects from the group;
     myGroup.remove(tmp,4);

     myGroup.add(o1);
     myGroup.add(o2);
     myGroup.add(o3);

    // Group merging
     POPGroup<Integer> myGroup2;
     myGroup2.add(tmp,4);
     myGroup.merge(myGroup2);

    // Exception handling
     try {
        myGroup.removeAt(8);
     } catch (RankException error) {
        cout<<error.getMessage();
     }
     try {
        myGroup.getMember(-1);
     } catch (RankException error) {
        cout<<error.getMessage();
     }

    // Emptying group;
     if(!myGroup.isEmpty())
       myGroup.emptyGroup();

     o1.Wait(5);
     o2.Wait(5);
     o3.Wait(5);
   }
   catch (paroc_exception *e)
     {
       cout<<"Object creation failure"<<"\n";
       return -1;
     }
   return 0;
}
*/
```

8.10.11. POPGroup.cpp

```
#include "POPGroup.hpp"
```

```
#include "A_reflection.hpp"
#include "Integer_reflection.hpp"
#include <limits.h>


using namespace reflcpp;
using namespace std;
/*************POPGroup()***********************/
template <class T>
POPGroup<T>::POPGroup(){
   size=0;
   char* className="A";
   insideClass=className;
   //printf("\nConstructor with [%s] as Class \n",className);
   ClassType ct = ClassType::getClass(className);
   std::string classNameRet = ct.name();
   //printf("Name       of       the       class       (should       be       [%s]):
[%s]\n",className,classNameRet.c_str());
}

/*************POPGroup(char*)******************/
template <class T>
POPGroup<T>::POPGroup(char* className){
   size=0;
   //printf("\nConstructor with [%s] as Class \n",className);
   ClassType ct = ClassType::getClass(className);
   insideClass=className;
   std::string classNameRet = ct.name();
   //printf("Name       of       the       class       (should       be       [%s]):
[%s]\n",className,classNameRet.c_str());
}

/*************broadcast(char*)*****************/
template <class T>
void POPGroup<T>::broadcast(char* method){
   //printf("broadcastmethod. try to call [%s] on [%s]\n",method,insideClass);
   ClassType ct = ClassType::getClass(insideClass); //BUG, insideClass vaut de la
MERDE-- corrigÃ©
   //printf("Broadcast       without       parameter       from       [%s]       on
[%s]\n",method,(ct.name()).c_str());
   MemberFunction mf =ct.getMemberFunction(method);
   int i;
   for (i=0;i<members.size();i=i+1)
   {
      //printf("Invoke [%s] on Element number [%d] \n",method,i);
      mf.invoke<void>(members[i]);
   }
}

/*************broadcast(char*,Arg1)*************/
template <class T>
template<class Arg1>
void POPGroup<T>::broadcast(char* method,Arg1 arg1){
   //printf("try to call [%s] on [%s] with 1 param\n",method,insideClass);
   ClassType ct = ClassType::getClass(insideClass);
   MemberFunction mf =ct.getMemberFunction(method);
   int i;
   for (i=0;i<members.size();i=i+1)
   {
      //printf("Invoke [%s] on Element number [%d] with 1 param\n",method,i);
      mf.invoke<void>(members[i],arg1);
   }
}

/*************broadcast(char*,Arg1,Arg2)********/
```

```
template <class T>
template<class Arg1,class Arg2>
void POPGroup<T>::broadcast(char* method,Arg1 arg1, Arg2 arg2){
   //printf("try to call [%s] on [%s] with 2 param\n",method,insideClass);
   ClassType ct = ClassType::getClass(insideClass);
   MemberFunction mf =ct.getMemberFunction(method);
   int i;
   for (i=0;i<members.size();i=i+1)
   {
      //printf("Invoke [%s] on Element number [%d] with 2 param\n",method,i);
      mf.invoke<void>(members[i],arg1,arg2);
   }
}

/*************scatter(char*,Arg1)***************/
template <class T>
template<class Arg1>
void POPGroup<T>::scatter(char* method,Arg1* arg1,int arraySize){
   //printf("try to call [%s] on [%s] with 1 param\n",method,insideClass);
   ClassType ct = ClassType::getClass(insideClass);
   MemberFunction mf =ct.getMemberFunction(method);
   int i,j;
   for (i=0;i<arraySize;i=i+1)
   {
      j=i%members.size();
      //printf("Invoke [%s] on Element number [%d] with 1 param\n",method,j);
      mf.invoke<void>(members[j],arg1[i]);
   }
}

/*************scatter(char*,Arg1,Arg2)**********/
template <class T>
template<class Arg1,class Arg2>
void POPGroup<T>::scatter(char* method,Arg1* arg1, Arg2* arg2,int arraySize){
   //printf("try to call [%s] on [%s] with 2 param\n",method,insideClass);
   ClassType ct = ClassType::getClass(insideClass);
   MemberFunction mf =ct.getMemberFunction(method);
   int i,j;
   for (i=0;i<arraySize;i=i+1)
   {
      j=i%members.size();
     // printf("Invoke [%s] on Element number [%d] with 2 param\n",method,j);
      mf.invoke<void>(members[j],arg1[i],arg2[i]);
   }
}

/*************broadcastGather(char*,Ret*)*******/
template<class T>
template<class Ret>
void POPGroup<T>::broadcastGather(char* method,Ret *ret){
   ClassType ct = ClassType::getClass(insideClass);
   MemberFunction mf =ct.getMemberFunction(method);
   int i;
   for (i=0;i<members.size();i=i+1)
   {
      //printf("Invoke [%s] on Element number [%d] \n",method,i);
      ret[i]=mf.invoke<Ret>(members[i]);
   }
}
/*************broadcastGather(char*,Arg1,Ret*)********/
template<class T>
template<class Ret,class Arg1>
void POPGroup<T>::broadcastGather(char* method, Arg1 arg1, Ret* ret){
   ClassType ct = ClassType::getClass(insideClass);
   MemberFunction mf =ct.getMemberFunction(method);
   int i;
   for (i=0;i<members.size();i=i+1)
   {
      //printf("Invoke [%s] on Element number [%d] \n",method,i);
      ret[i]=mf.invoke<Ret>(members[i],arg1);
   }
}
/*************broadcastGather(char*,Arg1,Arg2,Ret*)********/
template<class T>
template<class Ret,class Arg1,class Arg2>
void POPGroup<T>::broadcastGather(char* method, Arg1 arg1, Arg2 arg2, Ret* ret){
   ClassType ct = ClassType::getClass(insideClass);
   MemberFunction mf =ct.getMemberFunction(method);
   int i;
   for (i=0;i<members.size();i=i+1)
   {
      //printf("Invoke [%s] on Element number [%d] \n",method,i);
      ret[i]=mf.invoke<Ret>(members[i],arg1,arg2);
   }
}
/*************broadcastReduce(char*,Ret*,int)********/
template<class T>
template<class Ret>
void POPGroup<T>::broadcastReduce(char* method,Ret *retf,int type){
   ClassType ct = ClassType::getClass(insideClass);
   MemberFunction mf =ct.getMemberFunction(method);
   Ret ret[members.size()];
   int i;
   for (i=0;i<members.size();i=i+1)
   {
      //printf("Invoke [%s] on Element number [%d] \n",method,i);
      ret[i]=mf.invoke<Ret>(members[i]);
   }
   switch(type) {
         case 0:
            retf[0]= getMax(ret, size);
            break;
         case 1:
            retf[0]= getMin(ret, size);
            break;
         case 2:
            retf[0]= getOR(ret, size);
            break;
      }
}

/*************broadcastReduce(char*,Arg1,Ret*,int)********/
template<class T>
template<class Ret,class Arg1>
void POPGroup<T>::broadcastReduce(char* method, Arg1 arg1, Ret* retf, int type){
 ClassType ct = ClassType::getClass(insideClass);
   MemberFunction mf =ct.getMemberFunction(method);
   Ret ret[members.size()];
   int i;
   for (i=0;i<members.size();i=i+1)
   {
      //printf("Invoke [%s] on Element number [%d] \n",method,i);
      ret[i]=mf.invoke<Ret>(members[i],arg1);
   }
   switch(type) {
         case 0:
            retf[0]= getMax(ret, size);
            break;
         case 1:
            retf[0]= getMin(ret, size);
            break;
         case 2:
            retf[0]= getOR(ret, size);
            break;
      }
}
/*************broadcastReduce(char*,Arg1,Arg2,Ret*,int)********/
template<class T>
template<class Ret,class Arg1,class Arg2>
void POPGroup<T>::broadcastReduce(char* method, Arg1 arg1, Arg2 arg2, Ret* retf,
int type){
 ClassType ct = ClassType::getClass(insideClass);
   MemberFunction mf =ct.getMemberFunction(method);
   Ret ret[members.size()];
   int i;
   for (i=0;i<members.size();i=i+1)
   {
      //printf("Invoke [%s] on Element number [%d] \n",method,i);
      ret[i]=mf.invoke<Ret>(members[i],arg1,arg2);
   }
   switch(type) {
         case 0:
            retf[0]= getMax(ret, size);
            break;
         case 1:
            retf[0]= getMin(ret, size);
            break;
         case 2:
            retf[0]= getOR(ret, size);
            break;
      }
}
/*************scatterGather(char*,Arg1*,Ret*,int)********/
template<class T>
template<class Ret,class Arg1>
void POPGroup<T>::scatterGather(char* method, Arg1* arg1, Ret* ret, int arraySize){
//printf("try to call [%s] on [%s] with 1 param\n",method,insideClass);
   ClassType ct = ClassType::getClass(insideClass);
   MemberFunction mf =ct.getMemberFunction(method);
   int i,j;
   for (i=0;i<arraySize;i=i+1)
   {
      j=i%members.size();
  //    printf("Invoke [%s] on Element number [%d] with 1 param\n",method,j);
      ret[i]=mf.invoke<Ret>(members[j],arg1[i]);
   }
}
/*************scatterGather(char*,Arg1*,Arg2*,Ret*,int)********/
template<class T>
template<class Ret,class Arg1,class Arg2>
void POPGroup<T>::scatterGather(char* method, Arg1* arg1, Arg2* arg2, Ret* ret, int
arraySize){
   ClassType ct = ClassType::getClass(insideClass);
   MemberFunction mf =ct.getMemberFunction(method);
   int i,j;
   for (i=0;i<arraySize;i=i+1)
   {
      j=i%members.size();
    //  printf("Invoke [%s] on Element number [%d] with 1 param\n",method,j);
      ret[i]=mf.invoke<Ret>(members[j],arg1[i],arg2[i]);
   }
}
/*************scatterReduce(char*,Arg1*,Ret*,int,int)********/
template<class T>
template<class Ret,class Arg1>
```

```
void    POPGroup<T>::scatterReduce(char*    method,    Arg1*    arg1,Ret*    retf,int
arraySize,int type){
ClassType ct = ClassType::getClass(insideClass);
   MemberFunction mf =ct.getMemberFunction(method);
   Ret ret[members.size()];
   int i,j;
   for (i=0;i<arraySize;i=i+1)
   {
      j=i%members.size();
      //printf("Invoke [%s] on Element number [%d] with 1 param\n",method,j);
      ret[i]=mf.invoke<Ret>(members[j],arg1[i]);
   }
   switch(type) {
         case 0:
            retf[0]= getMax(ret, arraySize);
            break;
         case 1:
            retf[0]= getMin(ret, arraySize);
            break;
         case 2:
            retf[0]= getOR(ret, arraySize);
            break;
      }
}
/*************scatterReduce(char*,Arg1*,Arg2*,Ret*,int,int)********/
template<class T>
template<class Ret,class Arg1,class Arg2>
void POPGroup<T>::scatterReduce(char* method, Arg1* arg1, Arg2* arg2, Ret* retf,
int arraySize, int type){
ClassType ct = ClassType::getClass(insideClass);
   MemberFunction mf =ct.getMemberFunction(method);
   Ret ret[members.size()];
   int i,j;
   for (i=0;i<arraySize;i=i+1)
   {
      j=i%members.size();
      //printf("Invoke [%s] on Element number [%d] with 1 param\n",method,j);
      ret[i]=mf.invoke<Ret>(members[j],arg1[i],arg2[i]);
   }
   switch(type) {
         case 0:
            retf[0]= getMax(ret, arraySize);
            break;
         case 1:
            retf[0]= getMin(ret, arraySize);
            break;
         case 2:
            retf[0]= getOR(ret, arraySize);
            break;
      }
}


/*************add(T&)**************************/
template <class T>
void POPGroup<T>::add(T &o) {
   members.push_back(&o); //retirÃ© le & devant o-- BARRAS
   size++;
}

/*************add(T*,int)**********************/
template <class T>
void POPGroup<T>::add(T *o, int nb) {
   for(int i=0;i<nb;i++)
   {
      members.push_back(&(o[i]));//retirÃ© le & devant o-- BARRAS
      size++;
   }
}

/*************removeAt(int)*********************/
template <class T>
void POPGroup<T>::removeAt(int rank) {
   if(rank<0 || rank>(size-1))
      throw RankException(rank);
   it=members.begin()+rank;
   members.erase(it);
   size--;
}

/*************getRank(T&)**********************/
template <class T>
int POPGroup<T>::getRank(T &o) {
   for(int i=0;i<size;i++)
   {
      if(members.at(i)==&o) //BARRAS added the first &, new : retired both &
      {
         return i;
         break;
      }
   }
   return -1;
}

/*************isEmpty()************************/
template <class T>
bool POPGroup<T>::isEmpty() {
   return size==0;
}

/*************emptyGroup()*********************/
template <class T>
void POPGroup<T>::emptyGroup() {
   members.clear();
   size=0;
}

/*************remove(T*,int)*******************/
template <class T>
void POPGroup<T>::remove(T *o, int nb) {
   int rank;
   for(int i=0;i<nb;i++)
   {
      rank = getRank(o[i]); //ajoutÃ© * devant o
      removeAt(rank);
   }
}

/*************getSize()************************/
template <class T>
int POPGroup<T>::getSize() const {
   return size;
}

/*************getMember(int)*******************/
template <class T>
T& POPGroup<T>::getMember(int rank) {
   if(rank<0 || rank>(size-1))
      throw RankException(rank);
   else
```

```
      return *(members.at(rank)); //retirÃ© le * devant members BARRAS
}

/************merge(POPGroup<T>)***************/
template <class T>
void POPGroup<T>::merge(POPGroup<T> &g) {
   for(int i=0;i<g.getSize();i++)
      add(g.getMember(i));
}

// reduce operations for every needed return type
template <class T>
template <class Arg>
   Arg POPGroup<T>::getMax(Arg res[], int size) {
      Arg result = res[0];
      for(int i=1;i<size;i++)
         result = res[i]>result?res[i]:result;
      return result;
   }
template <class T>
template <class Arg>
   Arg POPGroup<T>::getMin(Arg res[], int size) {
      Arg result = res[0];
      for(int i=1;i<size;i++)
         result = res[i]<result?res[i]:result;
      return result;
   }
template <class T>
template <class Arg>
   Arg POPGroup<T>::getOR(Arg res[], int size) {
      Arg result = res[0];
      for(int i=1;i<size;i++)
         result |= res[i];
      return result;
   }
```

8.10.12. POPGroup.hpp

```
#ifndef _POPGROUP_H
#define _POPGROUP_H
#include <vector>
#include "src/ClassType.hpp"
//Includes made by the main.cpp in examples.
//#include "src/ClassType_tmpl.hpp"
#include "src/BoundClassType_tmpl.hpp"
#include "src/Exceptions.hpp"
#include "RankException.hpp"

using namespace reflcpp;

template <class T>
class POPGroup
{
public:
   POPGroup<T>();
   POPGroup<T>(char* className);

   void broadcast(char* method);
   template<class Arg1>
      void broadcast(char* method,Arg1 arg1);
   template<class Arg1,class Arg2>
      void broadcast(char* method,Arg1 arg1, Arg2 arg2);
  // void reduce(char* method[], int &result, int reduceType);
   template<class Arg1>
      void scatter(char* method, Arg1* arg1,int arraySize);
   template<class Arg1,class Arg2>
```

```
      void scatter(char* method, Arg1* arg1,Arg2*arg2,int arraySize);
   //no argument, an array in return
   template<class Ret>
      void broadcastGather(char* method,Ret *ret);
   //one argument, an array in return
   template<class Ret,class Arg1>
      void broadcastGather(char* method,Arg1 arg1,Ret *ret);
   //tow arguments, an array in return
  template<class Ret,class Arg1,class Arg2>
      void broadcastGather(char* method,Arg1 arg1,Arg2 arg2,Ret *ret);
   //no argument, a value in return
   template<class Ret>
      void broadcastReduce(char* method,Ret *ret, int type);
   //one argument, a value in return
   template<class Ret,class Arg1>
      void broadcastReduce(char* method,Arg1 arg1,Ret *ret,int type) ;
   //two arguments, a value in return
   template<class Ret,class Arg1,class Arg2>
      void broadcastReduce(char* method,Arg1 arg1,Arg2 arg2,Ret *ret, int type);
   //one argument, an array in return
   template<class Ret,class Arg1>
      void scatterGather(char* method,Arg1 *arg1, Ret *ret,int arraySize);
   //two arguments, an array in return
   template<class Ret,class Arg1,class Arg2>
      void  scatterGather(char*  method,Arg1  *arg1,  Arg2  *arg2,  Ret  *ret,int
arraySize);
   //one argument, a value in return
   template<class Ret,class Arg1>
      void  scatterReduce(char*  method,Arg1  *arg1,  Ret  *ret,  int  arraySize,  int
type);
   //two arguments, a value in return
   template<class Ret,class Arg1,class Arg2>
      void  scatterReduce(char*  method,Arg1  *arg1,  Arg2  *arg2,  Ret  *ret,int
arraySize, int type);

   void add(T &obj);
   void add(T *o, int nb);
   void removeAt(int rank);
   void remove(T *o, int nb);
   int getRank(T &o);
   int getSize() const;
   bool isEmpty();
   void emptyGroup();
   T &getMember(int rank);
   void merge(POPGroup<T> &g);

      const static int OPERATION_MAX=0;
   const static int OPERATION_MIN=1;
   const static int OPERATION_OR=2;
private:

  template<class Ret> Ret getMax(Ret res[], int size);
template<class Ret> Ret getMin(Ret res[], int size);
template<class Ret> Ret getOR(Ret res[], int size);
 char* insideClass;
 std::vector<T*> members;
 typename std::vector<T*>::iterator it;
 int size;
};


#endif
```

8.10.13. RankException.hpp

```
class RankException {
   int rank;
   char message[40];

 public:
   RankException(int r) {
      rank=r;
      sprintf(message,"Exception: Rank %d out of range\n",rank);
   }

   char *getMessage() {
      return message;
   }
};
```

8.10.14. Result of a POPGroup containing A objects

```
barraf@barraf-laptop:~/POPGroup$  parocc  -o  mainAB.out  mainAB.cpp  A_reflection.cpp
POPGroup.cpp A.cpp ./src/*.o
barraf@barraf-laptop:~/POPGroup$ parocrun objmap ./mainAB.o
mainAB.o    mainAB.out
barraf@barraf-laptop:~/POPGroup$ parocrun objmap ./mainAB.out


 !!! Test with POPGroup !!!

-----------Creation, add and remove---------
Object A constructed
Object A constructed
Object A constructed
Size of the group (should be [4]) : [4]
Size of the group (should be [3]) : [3]
Object A constructed
Object A constructed
Object A constructed
Object A constructed
Size of the group (should be [7]) : [7]
Size of the group (should be [3]) : [3]
Size of the group (should be [4]) : [4]
----------BROADCASTS----------
Object A constructed

Call method1 on Object A

Call method1 on Object A

Call method1 on Object A

Call method1 on Object A
Call method2 on Object A with value=[12]
Call method2 on Object A with value=[12]
Call method2 on Object A with value=[12]
Call method2 on Object A with value=[12]
Call method6 on Object A with value1=[22] and value2=[33]
Call method6 on Object A with value1=[22] and value2=[33]
Call method6 on Object A with value1=[22] and value2=[33]
----------SCATTERS----------
Call method6 on Object A with value1=[22] and value2=[33]
Call method2 on Object A with value=[0]
Call method2 on Object A with value=[1]
Call method2 on Object A with value=[2]
Call method2 on Object A with value=[3]
Call method6 on Object A with value1=[0] and value2=[10]
Call method6 on Object A with value1=[1] and value2=[11]
Call method6 on Object A with value1=[2] and value2=[12]
```

```
----------BROADCASTGATHERS----------
Call method6 on Object A with value1=[3] and value2=[13]
Call method3 on Object A
Call method3 on Object A
Call method3 on Object A
result from broadcastgather(0 param) (should be [14]) for object [0] : [14]
result from broadcastgather(0 param) (should be [14]) for object [1] : [14]
result from broadcastgather(0 param) (should be [14]) for object [2] : [14]
result from broadcastgather(0 param) (should be [14]) for object [3] : [14]
Call method3 on Object A
Call method4 on Object A with value=[14]
Call method4 on Object A with value=[14]
Call method4 on Object A with value=[14]
result from broadcastgather(1 param) (should be [28]) for object [0] : [28]
result from broadcastgather(1 param) (should be [28]) for object [1] : [28]
result from broadcastgather(1 param) (should be [28]) for object [2] : [28]
result from broadcastgather(1 param) (should be [28]) for object [3] : [28]
Call method4 on Object A with value=[14]
Call method5 on Object A with value1=[23] and value2=[43]
Call method5 on Object A with value1=[23] and value2=[43]
Call method5 on Object A with value1=[23] and value2=[43]
result from broadcastgather(2 params) (should be [66]) for object [0] : [66]
result from broadcastgather(2 params) (should be [66]) for object [1] : [66]
result from broadcastgather(2 params) (should be [66]) for object [2] : [66]
result from broadcastgather(2 params) (should be [66]) for object [3] : [66]
----------BROADCASTREDUCES----------
Call method5 on Object A with value1=[23] and value2=[43]
Call method3 on Object A
Call method3 on Object A
Call method3 on Object A
result from broadcastreduce(0 param) (shoudld be [14]) for Group : [14]
Call method3 on Object A
Call method4 on Object A with value=[14]
Call method4 on Object A with value=[14]
Call method4 on Object A with value=[14]
result from broadcastreduce(1 param) (shoudld be [28]) for Group : [28]
Call method4 on Object A with value=[14]
Call method5 on Object A with value1=[23] and value2=[43]
Call method5 on Object A with value1=[23] and value2=[43]
Call method5 on Object A with value1=[23] and value2=[43]
result from broadcastreduce(2 param) (shoudld be [66]) for Group : [66]
----------SCATTERGATHERS----------
Call method5 on Object A with value1=[23] and value2=[43]
Call method4 on Object A with value=[0]
Call method4 on Object A with value=[1]
Call method4 on Object A with value=[2]
result from scattergather(1 param) (should be [0]) for object [0] : [0]
result from scattergather(1 param) (should be [2]) for object [1] : [2]
result from scattergather(1 param) (should be [4]) for object [2] : [4]
result from scattergather(1 param) (should be [6]) for object [3] : [6]
Call method4 on Object A with value=[3]
Call method5 on Object A with value1=[0] and value2=[10]
Call method5 on Object A with value1=[1] and value2=[11]
Call method5 on Object A with value1=[2] and value2=[12]
result from scattergather(2 params) (should be [10]) for object [0] : [10]
result from scattergather(2 params) (should be [12]) for object [1] : [12]
result from scattergather(2 params) (should be [14]) for object [2] : [14]
result from scattergather(2 params) (should be [16]) for object [3] : [16]
----------SCATTERREDUCES----------
Call method5 on Object A with value1=[3] and value2=[13]
Call method4 on Object A with value=[0]
Call method4 on Object A with value=[1]
Call method4 on Object A with value=[2]
result from scatterreduce(1 param) (shoudld be [4]) for Group : [6]
Call method4 on Object A with value=[3]
Call method5 on Object A with value1=[0] and value2=[10]
Call method5 on Object A with value1=[1] and value2=[11]
Call method5 on Object A with value1=[2] and value2=[12]
result from scatterreduce(2 param) (shoudld be [16]) for Group : [16]
----------BROADCASTGATHER with objects as return----------
Call method5 on Object A with value1=[3] and value2=[13]
Object A constructed
Object A constructed
Object A constructed
Object A constructed
Object A constructed
Call method8 on Object A Object A constructed
Object A constructed
Call method8 on Object A Object A constructed
Object A constructed
Call method8 on Object A Object A constructed
Object A constructed
Call method8 on Object A ------ emptying group---------
is Group now empty ? [true]
size of the group (should be [0])=0
Object A constructed
[objectmonitor.cc:69]Check parallel objects....0 object alive
barraf@barraf-laptop:~/POPGroup$ [codemgr.cc:13]Now destroy CodeMgr
```