Parallel Object Programming Java
User and Installation Manual

Software version : 1.0 RC
Manual version : 1.0 RC

Author:
Valentin Clément
Beat Wolf
Dr. Pierre Kuonen

Contents

1 Remarks

This document is part of the bachelor thesis project named "POP-Java" written by Valentin Clément in 2010.

2 Introduction and background

2.1 Introduction

Programming large heterogeneous distributed environments such as GRID or P2P infrastructures is a challenging task. This statement remains true even if we consider researches that have focused on enabling these types of infrastructures for scientific computing such as resource management and discovery [7, 9, 5], service architecture [8], security [17] and data management [4, 15]. Efforts to port traditional programming tools such as MPI [6, 14, 10] or BSP [16, 18], also had some success. These tools allow programmers to run their existing parallel applications on large heterogeneous distributed environments. However, efficient exploitation of performance regarding the heterogeneity still needs to be manually controlled and tuned by programmers.

POP-C++ and POP-Java are implementations of the POP (**P**arallel **O**bject **P**rograming) model first introduced by Dr. Tuan Anh Nguyen in his PhD thesis [12]. POP-C++ is an extension of the C++ programming language[11] and POP-Java is an extension of the Java programming language[1]. The POP model is based on the very simple idea that objects are suitable structures to distribute data and executable codes over heterogeneous distributed hardware and to make them interact between each other.
Inspired by CORBA [13] and C++, the POP-C++ programming language extends C++ by adding a new type of **parallel object**, allowing to run C++ objects in distributed environments. With POP-C++, programming efficients distributed applications is as simple as writing a C++ programs. The POP-Java programming language extends Java and implements the same mechanisms as POP-C++.

2.2 The POP model

The POP model extends the traditional object oriented programming model by adding the minimum necessary functionality to allow for an easy development of coarse grain distributed high performance applications. When the object oriented paradigm has unified the concept of module and type to create the new concept of **class**, the POP model unifies the concept of class with the concept of **task** (or **process**). This is realized by adding to traditional sequential classes a new type of class: **the parallel class**. By instantiating parallel classes we are able to create a new category of objects we will call **parallel objects** in the rest of this document.

Parallel objects are objects that can be remotely executed. They coexist and cooperate with traditional sequential objects during the application execution. Parallel objects keep advantages of object-orientation such as data encapsulation, inheritance and polymorphism and adds new properties to objects such as:

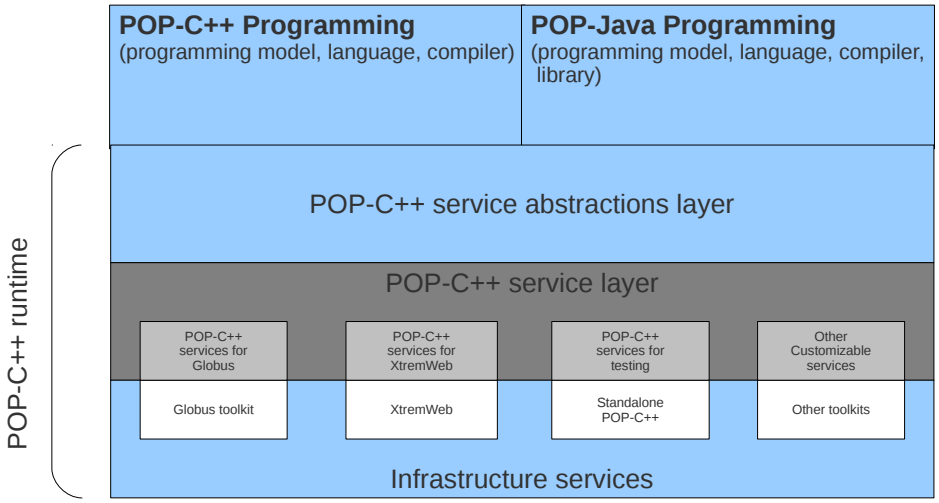
- Distributed shareable objects
- Dynamic and transparent object allocation
- Various method invocation semantics

2.3 System overview

Although the POP-C++ programming system focuses on an object-oriented programming model, it also includes a runtime system which provides the necessary services to run POP-C++ and POP-Java applications over distributed environments.

An overview of the POP system (Both POP-C++ and POP-Java) architecture is illustrated in figure[1]. In POP-Java, only the programming system is implemented and the runtime system is the same as the one used in POP-C++.

Figure 1: POP system architecture



The POP-C++ runtime system consists of three layers: the service layer, the POP-C++ service abstractions layer, and the programming layer. The service layer is built to interface with lower level toolkits (e.g. Globus) and the operating system. The essential service abstraction layer provides an abstract interface for the programming layer. On top of the architecture is the programming layer, which provides necessary support for developing distributed object-oriented applications. More details of the POP-C++ runtime layers are given in a separate document [12].

2.4 Structure of this manual

This manual has 8 chapters, including this introduction. The next chapter (chapter 3) explains the POP model. The chapter 4 describes the POP-Java application development process. The chapter 5 explains the compilation and the launch process of a POP-Java application. The chapter 6 aims to describe and explain how to use of POP-C++ and POP-Java together in a same application. The chapter 7 describes the POP-Java plugin system. The chapter 8 guides the user trough the installation process. Finally, the chapter 10 gives some hints to solve the main problems that can occurs with a POP-Java application.

3 Parallel Object Model

3.1 Introduction

Object-oriented programming provides high level abstractions for software engineering. In addition, the nature of objects makes them ideal structures to distribute data and executable codes over heterogeneous distributed hardware and to make them interact between each other. Nevertheless, two questions remain:

- Question 1: which objects should run remotely?
- Question 2: where does each remote object live?

The answers, of course, depend on what these objects do and how they interact with each other and with the outside world. In other words, we need to know the communication and the computation requirements of objects. The parallel object model presented in this chapter provides an object-oriented approach for requirement-driven high performance applications in a distributed heterogeneous environment.

3.2 Parallel Object Model

POP stands for *Parallel Object Programming*, and POP parallel objects are generalizations of traditional sequential objects. POP-Java is an extension of Java that implements the POP model. POP-Java instantiates parallel objects transparently and dynamically, assigning suitable resources to objects. POP-Java also offers various mechanisms to specify different ways to do method invocations. Parallel objects have all the properties of traditional objects plus the following ones:

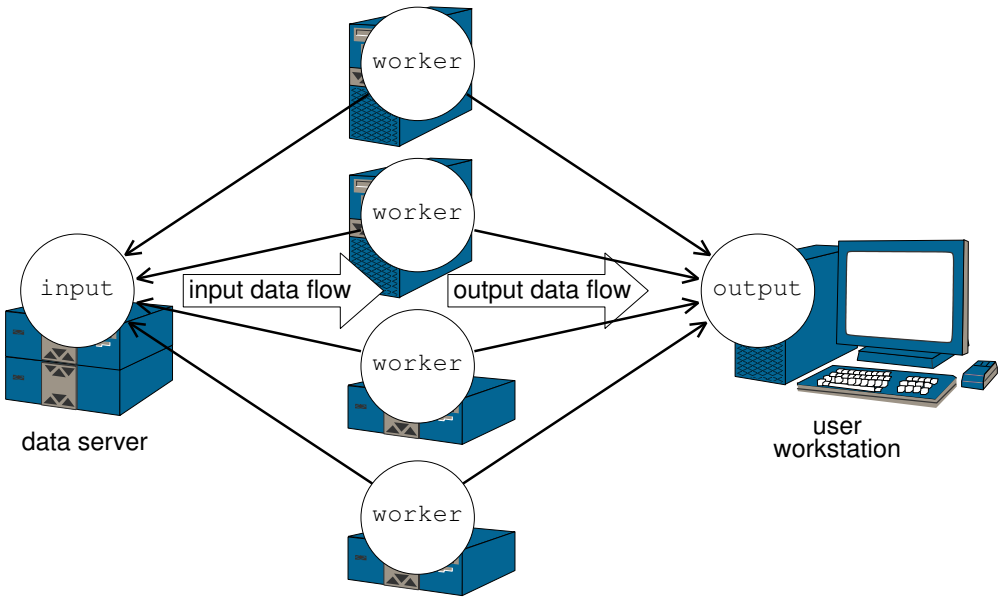
- Parallel objects are shareable. References to parallel objects can be passed to any other parallel object. This property is described in section 3.3.
- Syntactically, invocations on parallel objects are identical to invocations on traditional sequential objects. However, parallel objects support various method invocation semantics: synchronous or asynchronous, and sequential, mutex or concurrent. These semantics are explained in section 3.4.
- Parallel objects can be located on remote resources in separate address spaces. Parallel objects allocations are transparent to the programmer. The object allocation is presented in section 3.5.
- Each parallel object has the ability to dynamically describe its resource requirement during its lifetime. This feature is discussed in detail in section 3.6

As for traditional objects, parallel objects are active only when they execute a method (non active object semantic). Therefore, communication between parallel objects are realized thank to remote methods invocation.

3.3 Shareable Parallel Objects

Parallel objects are shareable. This means that the reference of a parallel object can be shared by several other parallel objects. Sharing references of parallel objects are useful in many cases. For example, figure[2] illustrates a scenario of using shared parallel objects: `input` and `output` parallel objects are shareable among `worker` objects. A `worker` gets work units from `input` which is located on the data server, performs the computation and stores the results in the `output` located at the user workstation. The results from different `worker` objects can be automatically synthesized and visualized inside `output`.

Figure 2: A scenario using shared parallel objects



To share the reference of a parallel object, POP-Java allows parallel objects to be arbitrarily passed from one place to another as arguments of method invocations.

3.4 Invocations semantics

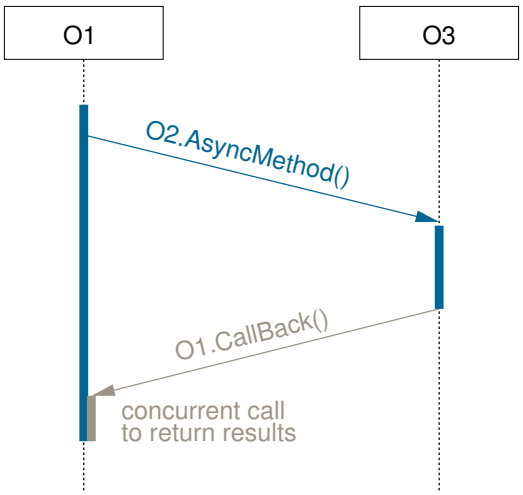
Syntactically, method invocations on parallel objects are identical to those on traditional sequential objects. However, to each method of a parallel object, one can associate different invocation semantics. Invocation semantics are specified by programmers when declaring methods of parallel objects. These semantics define different behaviors for the execution of the method as described below (example of syntax in chapter 4)

- **Interface semantics**, the semantics that affect the caller of the method:

  Synchronous invocation: the caller waits until the execution of the called method on the remote object is terminated. This corresponds to the traditional method invocation.

  Asynchronous invocation: the invocation returns immediately after sending the request to the remote object. Asynchronous invocation is important to exploit the parallelism. However, as the caller does not wait the end of the execution of the called method, no computing result is available. This excludes asynchronous invocations from producing results. Results can be actively returned to the caller object using a callback to the caller. To do so the called object must have a reference to the caller object. This reference can be passed as an argument to the called method (see figure[3]).

Figure 3: Callback method returning values from an asynchronous call



- **Object-side semantics**, the semantics that affect the order of the execution of methods in the called parallel object:

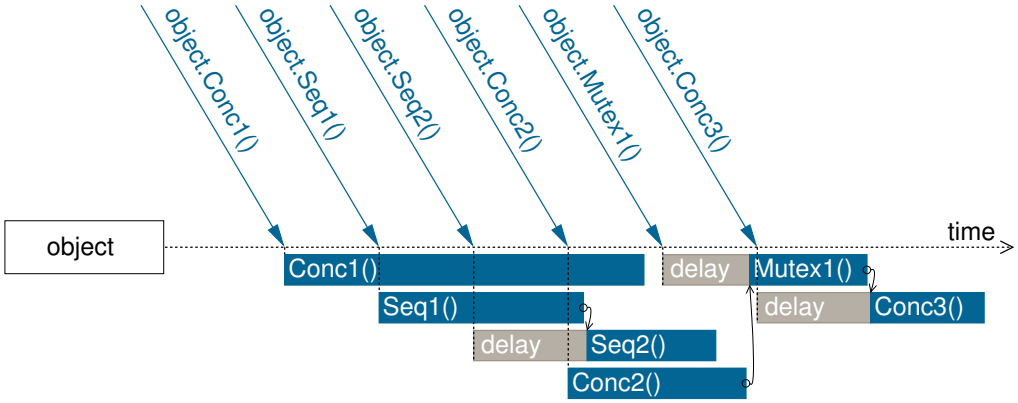
  A mutex call is executed after completion of all calls previously arrived.

  A sequential call is executed after completion of all sequential and mutex calls previously arrived.

  A concurrent call can be executed concurrently (time sharing) with other concurrent or sequential calls, except if mutex calls are pending or executing. In the later case he is executed after completion of all mutex calls previously arrived.

In a nutshell, different object-side invocation semantics can be expressed in terms of atomicity and execution order. The mutex invocation semantics guarantees the global order and the atomicity of all method calls. The sequential invocation semantics guarantees only the execution order of sequential methods. Concurrent invocation semantics guarantees neither the order nor the atomicity.
Figure[4] illustrates different method invocation semantics. Sequential invocation `Seq1()` is served immediately, running concurrently with `Conc1()`. Although the sequential invocation `Seq2()` arrives before the concurrent invocation `Conc2()`, it is delayed due to the current execution of `Seq1()` (no order between

Figure 4: Example of different invocation requests



concurrent and sequential invocations). When the mutex invocation `Mutex1()` arrives, it has to wait for other running methods to finish. During this waiting, it also blocks other invocation requests arriving afterward (`Conc3()`) until the mutex invocation request completes its execution (atomicity and barrier).

3.5 Parallel Object Allocation

The first step to allocate a new object is the selection of an adequate placeholder. The second step is the object creation itself. Similarly, when an object is no longer in use, it must be destroyed in order to release the resources it is occupying in its placeholder. The POP-C++ runtime system provides automatic placeholder selection, object allocation, and object destruction. This automatic features result in a dynamic usage of computational resources and gives to the applications the ability to adapt to the changes in both the environment and the user behavior.

The creation of POP-Java parallel objects is driven by high-level requirements on the resources where the object should lie (see section 3.6). If the programmer specifies these requirements they are taken into consideration by the runtime system for the transparent object allocation. The allocation process consists of three phases: first, the system finds a suitable resource, where the object will lie; then the object code is transmitted and executed on that resource; and finally, the corresponding interface is created and connected to the object.

3.6 Requirement-driven parallel objects

Parallel processing is increasingly being done using distributed systems, with a strong tendency towards web and global computing. Efficiently extract high performance from highly heterogeneous and dynamic distributed environments is a challenge today. POP-C++ and POP-Java were conceived under the belief that for such environments, high performance can only be obtained if the two following conditions are satisfied:

- The application should be able to adapt to the environment;
- The programming environment should somehow enables objects to describe their resource requirements.

The application adaptation to the environment can be fulfilled by multilevel parallelism, dynamic utilization of resources or adaptive task size partitioning. One solution is to dynamically create parallel objects on demand.

Resource requirements can be expressed by the quality of service that objects require from the environment. Most of the systems offering quality of service focus on low-level aspects, such as network bandwidth reservation or real-time scheduling. Both POP-C++ and POP-Java integrate the programmer requirements into parallel objects in the form of high-level resource descriptions. Each parallel object is associated with an object description that depicts the characteristics of the resources needed to execute the object. The resource requirements in object descriptions are expressed in terms of:

- Resource (host) name (low level description, mainly used to develop system services).
- The maximum computing power that the object needs (expressed in MFlops).
- The maximum amount of memory that the parallel object consumes.
- The expected communication bandwidth and latency.
- The preferred communication protocol.
- The preferred encoding protocol.

An object description can contain several items. Each item corresponds to a type of characteristics of the desired resource. The item is classified into two types: strict item and non-strict item. A strict item means that the designated requirement must be fully satisfied. If no satisfying resource is available, the allocation of parallel object fails. Non-strict items, on the other hand, give the system more freedom in selecting a resource. Resource that partially match the requirements are acceptable although a full qualification resource is preferable. For example, a certain object has a preferred performance 150MFlops although 100MFlops is acceptable (non-strict item), but it need memory storage of at least 128MB (strict item).

The construction of object descriptions occurs during the parallel object creation. The programmer can provide an object description to each object constructor. The object descriptions can be parametrized by the arguments of the constructor. Object descriptions are used by the runtime system to select an appropriate resource for the object. Some example of the syntax of object descriptions can be found in the section 4.1.4.

It can occur that, due to some changes on the object data or some increase of the computation demand, an object description needs to be re-adjusted during the life time of the parallel object. If the new requirement exceeds some threshold, the adjustment could cause the object migration. The current implementations of POP-C++ and POP-Java do not support object migration yet.

4 Developing POP-Java application

The POP model (chapter 3) is a suitable programming model for large heterogeneous distributed environments but it should also remain as close as possible to traditional object oriented programming. Parallel objects of the POP model generalize sequential objects, keep the properties of object oriented programming (data encapsulation, inheritance and polymorphism) and add new properties.

The POP-Java language is an extension of Java that implements the POP model. Its syntax remains as close as possible to standard Java so that Java programmer can easily learn it and existing Java libraries can be parallelized without much effort. Changing a sequential Java application into a distributed parallel application is rather straightforward. POP-Java is also very close to POP-C++ so that POP programmer can use both system easily.

Parallel objects are created using parallel classes. Any object that instantiates a parallel class is a parallel object and can be executed remotely. To help the POP-C++ runtime to choose a remote machine to execute the remote object, programmers can add object description information to each constructor of the parallel object. In order to create parallel execution, POP-Java offers new semantics for method invocations. These new semantics are indicated thanks to new POP-Java keywords. This chapter describes the syntax of the POP-Java programming language and presents the main tools available in the POP-Java system.

4.1 Parallel objects

POP-Java parallel objects are a generalization of sequential objects. Unless the term sequential object is explicitly specified, a parallel object is simply referred as an object in the rest of this chapter.

4.1.1 Create a parallel class

Developing POP-Java application mainly consists of designing an implementing parallel classes. The declaration of a parallel class is the same as a standard Java class declaration, but it has to be annotated with the annotation **@POPClass**. The parallel class can extend another parallel class but not a sequential class.

Simple parallel class declaration

```
@POPClass
public class MyParallelClass {
        //Implementation
}
```

Parallel class declaration with an inheritance

```
@POPClass
public class MyParallelClass extends AnotherParallelClass {
        //Implementation
}
```

As Java allows only the single inheritance, a parallel class can only inherit from **one** other parallel class. The Java language also impose that the file including the parallel class has the same name than the parallel class.

POP-Java imposes also one restrictions. The parallel class must be declared and implemented in a file with **.pjava** extension. For example, the parallel class *MyParallelClass* declared before, must be in a file **MyParallelClass.pjava**.

Parallel classes are very similar to standard Java classes. As POP-Java has some different behavior, some restrictions applied to the parallel classes :

- All attributes in a parallel class must be protected or private
- The objects do not access any global variable
- A parallel class does not contain any static attributes or static methods

4.1.2 Creation and destruction

The object creation process consists of several steps: locating a resource satisfying the object description (resource discovery), transmitting and executing the object code, establishing the communication, transmitting the constructor arguments and finally invoking the corresponding object constructor. Failures on the object creation will raise an exception to the caller. Section 4.3 will describe the POP-Java exception mechanism.

As a parallel object can be accessible concurrently from multiple distributed locations (shared object), destroying a parallel object should be carried out only if there is no other reference to the object. POP-Java manages parallel objects' life time by an internal reference counter. A counter value of 0 will cause the object to be physically destroyed.

Syntactically, the creation of a parallel object is identical to the one in Java. A parallel object can be created by using the standard new operator of Java.

4.1.3 Parallel class methods

Like sequential classes, parallel classes contain methods and attributes. Method can be public or private but attribute must be either protected or private. For each public method, the programmer must define the invocation semantics. These semantics, described in section 3.4, are specified by an annotations.

- **Interface side** : These semantics affect the caller side.

  sync : Synchronous invocation.

  async : Asynchronous invocation.

- **Object side** : These semantics affect the order of the class inside the object itself.

  seq : Sequential invocation

  conc : Concurrent invocation

  mutex : Mutual exclusive invocation

The combination of the interface and object-side semantics defines the overall semantics of a method. There are 6 possible combinations of the interface and object-side semantics, resulting in 6 annotions:

```
@POPSyncConc
@POPSyncSeq
@POPSyncMutex
@POPAsyncConc
@POPAsyncSeq
@POPAsyncMutex
```

For example, a synchronous concurrent method returning an int value must be declared as follow :

```
@POPSyncConc
public int myMethod(){
        return myIntValue;
}
```

A method declared as asynchronous must have its return type set to void. Otherwise, the compiler will raise an error.

4.1.4 Object description

Object descriptions are used to describe the resource requirements for the execution of an object. Object descriptions are declared along with parallel object constructor declarations. The object description can be declared in a static way as an annotation of the constructor, or in a dynamic way as an annotation on the parameters of the constructor. First an example of a static annotation:

```
@POPObjectDescription(url="localhost")
public MyObject(){
}
```

and now a dynamic example:

```
public MyObject(@POPConfig(Type.URL) String host){
}
```

Currently only the url annotation is implemented, allowing to specify the url/IP of the machine on which the POP-Object is executed. If the annotation is not set, POP-Java will use the POP-C++ jobmanager to find a suitable machine.

4.1.5 Data marshaling and IPOPBase

When calling a remote methods, the arguments must be transferred to the object being called (the same happens for the return value and the exception). In order to operate with different memory spaces and different architecture, data is marshaled into a standard format prior to be send to remote objects. All data passed is serialized (marshaled) at the caller side an deserialized (unmarshaled) at the called side.

With POP-Java all primitive types, primitive types arrays and parallel classes can be passed without any trouble to another parallel object. This mechanism is transparent for the programmer.

If the programmer want to pass a special object to or between parallel classes, this object must implement the IPOPBase interface from the POP-Java library. This library is located in the installation directory (*POPJAVA_LOCATION/JarFile/popjava.jar*). By implementing this interface, the programmer will have to override the two following methods :

```
@Override
public boolean deserialize(Buffer buffer) {
  return true;
}

@Override
public boolean serialize(Buffer buffer) {
  return true;
```

```
}
```

These methods will be called by the POP-Java system when an argument of this type need to be serialized or deserialized. As the object will be reconstruct on the other side and after the values will be set to it by the deserialize method, any class implementing the IPOPBase interface must have a default constructor.

The code below shows a full example of a class implementing the IPOPBase interface.

```
import popjava.buffer.Buffer;
import popjava.dataswaper.IPOPBase;

public class MyComplexType implements IPOPBase {
        private int theInt;
        private double theDouble;
        private int[] someInt;

        public MyComplexType(){}

        public MyComplexType(int i, double d, int[] ia){
                theInt = i;
                theDouble = d;
                someInt = ia;
        }

        @Override
        public boolean deserialize(Buffer buffer) {
                theInt = buffer.getInt();
                theDouble = buffer.getDouble();
                int size = buffer.getInt();
                someInt = buffer.getIntArray(size);
                return true;
        }

        @Override
        public boolean serialize(Buffer buffer) {
                buffer.putInt(is);
                buffer.putDouble(ds);
                buffer.putIntArray(ias);
                return true;
        }
}
```

4.2 POP-Java behavior

This section aims to explain to difference between the standard Java behavior and the POP-Java behavior.

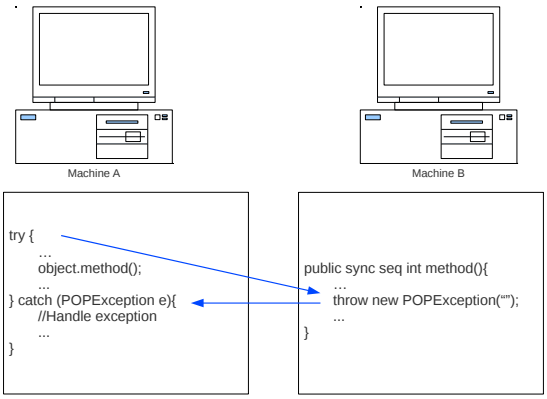
As in standard Java, the primitive types will not be affected by any manipulations inside a methods. The objects will be affected only if the method semantic is "Synchronous". In fact, POP-Java serialize the method arguments to pass them on the object-side. Once the method work is done, the arguments are serialize once again to be send back to the interface-side. If the method semantic is "Synchronous", the interface-side will deserialize the arguments and replace the local ones by the deserialized arguments. If the method semantic is "Asynchronous", the interface-side will not wait for any answer from the object-side. It's important to understand this small difference when developing POP-Java application.

4.3 Exception handling

Errors can be efficiently handled using exceptions. Instead of handling each error separately based on an error code returned by a function call, exceptions allow the programmer to filter and centrally manage errors trough several calling stacks. When an error is detected inside a certain method call, the program can throw an exception that will be caught somewhere else.

The implementation of exception in non-distributed applications, where all components run within the same memory address space is fairly simple. The compiler just need to pass a pointer to the exception from the place where it is thrown to the place where it is caught. However, in distributed environments where each component is executed in a separated memory address space (and eventually data are represented differently due to heterogeneity), the propagation of exception back to a remote component is complex.

Figure 5: Exception handling example



POP-Java supports transparent exception propagation. Exceptions thrown in a parallel object will be automatically propagated back to the remote caller (figure[5]). The current POP-Java version allows the following types of exceptions :

- Exception
- POPException

The invocation semantics of POP-Java affect the propagation of exceptions. For the moment, only synchronous methods can propagate the exception. Asynchronous methods will not propagate any exception to the caller. POP-Java current behavior is to abort the application execution when such exception occurs.

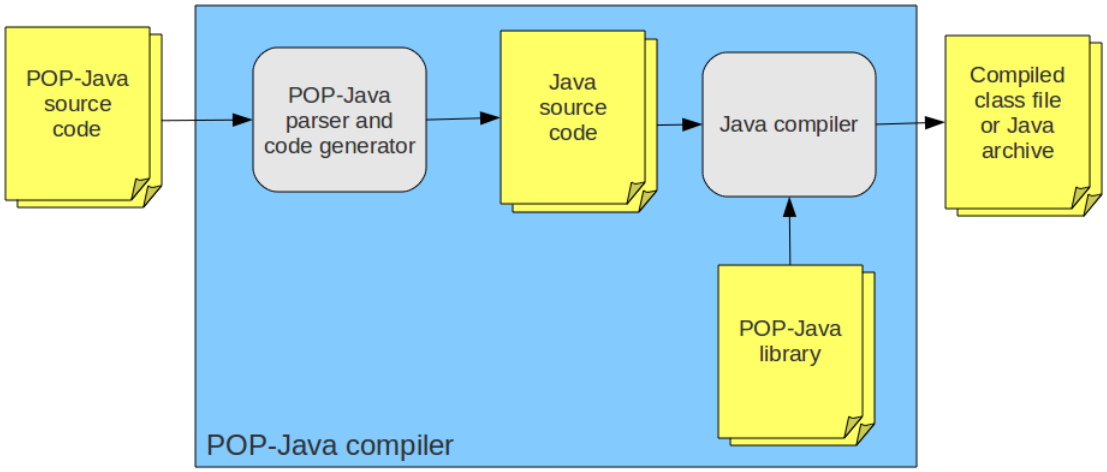
5 Compile and run a POP-Java application

This chapter explain the POP-Java compilation process, the POP-Java application launching process and the tools related to those processes. The structure of this chapter is as follows : The first section explains the compilation process and the special compiler for POP-Java. The second describes the application launching tools. The third one aims to help the programmer to understand the importance of the object map and the object map generator in the POP-Java application launching process. Finally, a full example is explained to pass trough the whole process.

5.1 The POP-Java compiler

POP-Java has its own compilation process. As POP-Java applications are written in the POP-Java language, the programmer have to use the POP-Java compiler to compile its application. The figure [6] shows the POP-Java compilation process. The POP-Java source files will be converted in standard Java source files by a specific parser and code generator. These Java source files will be compiled with the POP-Java library and produce compiled class files or a Java archive file (JAR).

Figure 6: POP-Java compilation process



The POP-Java compiler will parse and generate new source file only for the files with .pjava extension. Due to that, it is very important the main class and the parallel classes have this extension and not the standard .java extension.

The POP-Java compiler is accessible trough the command "popjc". The command syntax is as follow :

```
popjc <options> <files>
```

Options

- -n or –noclean : Do not clean the temporary files generated by the parser and code generator.
- -p or –popcpp : Use the specified additional informations XML file for the compilation (See chapter 6).
- -j or –jar : Compile the source files in a JAR file. Name of the JAR must be specified.
- -v or –verbose : Print additional information during the compilation process.

- -c or –classpath : Add class file or JAR file to the compilation
- -x or –xmlpopcpp : Generate the XML additional information file with the specified source files (See chapter 6).
- -g or –generate : Generate the POP-C++ partial implementation from the specified source file (See chapter 6). Not implemented in the current version.

The files in the command line are the ones to compile (.pjava or .java). The command help is available in the appendix A.

5.2 The POP-Java application launcher

To help POP-Java programmer, POP-Java provide an application launcher that simplify the launch of a POP-Java application. This application launcher is named "popjrun" and is used with the following syntax :

```
popjrun <options> [<objectmap>] <MainClass> <arguments>
```

Here is an explanation of the arguments to provide to the POP-Java application launcher :

- **options** : in the current version there is only one option "-c" or "–classpath" that allow the programmer to add some class path for the execution of the POP-Java application. The different class paths must be separated with a semicolon.
- **objectmap** : this informations is not mandatory. If it's provided, the object map inform the runtime system about the location of the different compiled parallel classes of the application. If it's not provided, the default object map (located under : *POPJAVA_LOCATION*/etc/defaultobjectmap.xml) will be used. More information give in the section 5.3.
- **MainClass** : this is a main class of the POP-Java application.
- **arguments** : there are the arguments of the program

5.3 The POP-Java object map and object map generator

The object map is an XML file that informs the POP-C++ runtime about the location of the different compiled parallel classes of the application. This file can be given to the "popjrun" tool. If the programmer do not give this file, the default object map located in *POPJAVA_LOCATION*/etc/ will be used.

The object map can be generated with the POP-Java application launcher. By using the option "-l" or "–listlong" and giving the class files or the JAR file, the object map will be printed to the standard output. The easiest way to save this file is to redirect the output into the desired file.

Here are the command used for our example :

Compiled classes

```
popjrun --listlong Parclass1.class:Parclass2.class > objectmap.xml
```

JAR file

```
popjrun --listlong parclasses.jar > objectmap.xml
```

5.4 Full example

This section shows how to write, compile and launch a POP-Java application by using a simple example. The POP-Java application used in this example includes only one parallel class. All sources of this example can be found in the directory "examples/integer" from the POP-Java distribution.

5.4.1 Programming

When we start to develop a POP-Java application the main part is the parallel classes. The figure [7] shows the parallel class implementation. As we can see this class use special POP-Java keywords. In the line 1, the parclass keyword specifies that this class is a parallel class. The constructor declaration includes an object description (line 4). The method declarations includes the invocation semantics (line 8, 12 and 16). The method "add" (line 16) receive another parallel object as a parameter and it's transparent for the programmer.

Figure 7: File Integer.pjava

```
1:  @POPClass
2:  public class Integer {
3:    private int value;
4:
5:        @POPObjectDescription(url="localhost")
6:        public Integer(){
7:          value = 0;
8:        }
9:
10:       @POPSyncConc
11:       public int get(){
12:         return value
13:       }
14:
15:       @POPAsyncSeq
16:       public void set(int val){
17:         value = val;
18:       }
19:
20:       @POPAsyncMutex
21:       public void add(Integer i){
22:         value += i.get();
23:       }
24:     }
```

Once the parallel class is implemented, we can write a main class that use this parallel class. The figure [8] shows the code of the main class. The code of the main class is pure Java code. However, this code must be declared in a file with ".pjava" extension to be considered by the POP-Java compiler. The instantiation (lines 3-4) and the method calls (lines 5-9) are transparent for the programmer.

Figure 8: File TestInteger.pjava

```
1:      public TestInteger {
2:        public static void main(String[] args){
3:          Integer i1 = new Integer();
4:          Integer i2 = new Integer();
5:          i1.set(23);
6:          i2.set(25);
7:          System.out.println("i1=" + i1.get());
8:          System.out.println("i2=" + i2.get());
9:          i1.add(i2);
10:         int sum =  i1.get();
11:         System.out.println("i1+i2 = "+sum);
12:         if(sum==48)
13:           System.out.println("Test Integer Successful");
14:         else
15:           System.out.println("Test Integer failed");
16:         }
17:     }
```

5.4.2 Compiling

The POP-Java compiler can generate two kind of compiled code. The first is the standard Java compiled class file (.class). The second is the Java archive (JAR) file. Here are the two commands to compile the example application.

Compiling as .class files

```
popjc Integer.pjava TestInteger.pjava
```

Compiling as a JAR file

```
popjc -j myjar.jar Integer.pjava TestInteger.pjava
```

5.4.3 Create the object map

Before running the example application, the programmer needs to generate the object map. The object map will be given to the POP-Java launcher. This file will inform the POP-C++ runtime system where to find the compiled files. The POP-Java launcher has a specific option to generate this file from the compiled files (.class) or the JAR file (.jar). Here is the command used for our example :

```
popjrun --listlong Integer.class > objmap.xml
```

The command will generate the XML file and print it on the standard output. To save this file, we redirect the output in a file named objmap.xml. This file contains the following XML code (the path specified in the element CodeFile will be different on your computer) :

```
<CodeInfoList>
  <CodeInfo>
    <ObjectName>Integer</ObjectName>
    <CodeFile Type="popjava">
      /home/clementval/pop/popjava-1.0/example/integer/</CodeFile>
    <PlatForm>*-*</PlatForm>
  </CodeInfo>
</CodeInfoList>
```

5.4.4 Running

Once the POP-Java application is compiled and the object map is generated, the application can be run. A POP-Java application is a pure Java application at the end and could be run with the standard java program. In order to make this running easier for the programmer, POP-Java include an application launcher. Here are the command to use to run the POP-Java application example :

POP-Java application compiled as .class files

```
popjrun objectmap.xml TestInteger
```

POP-Java application compiled as .jar file

```
popjrun -c myjar.jar objectmap.xml TestInteger
```

Application output
Here is what we should have as the application output :

```
i1=23
i2=25
i1+i2=48
Test Integer Successful
```

If we have any problems with the compilation or the launching of the application, please see the chapter 10.

6 Developing POP-Java and POP-C++ mixed application

6.1 POP-Java and POP-C++ interoperability

POP-Java can use POP-C++ parallel classes and POP-C++ can also use POP-Java parallel classes. This chapter will explain everything the programmer needs to know to develop mixed POP application.

6.2 Restrictions

As Java and C++ are different languages, there are some restrictions. In this section, all the restrictions or programming tips will be given.

6.2.1 Java primitives

As Java primitives are always passed by value, the is no way to modify a Java primitive in a POP-C++ object. In pure POP-C++ the programmer can deal with the passing by reference but not in POP-Java.

6.2.2 Parameters passing

Some parameters cannot be passed from a POP-Java application to a POP-C++ parallel object and vice versa. The list below explain the restrictions on certain primitive types. The Java primitive types are taken as the basis.

- **byte** : This type does not exist in C++ so it's not possible to pass a byte.
- **long** : The Java long is coded on 8 bytes as it's coded on 4 bytes with C++. Some unexpected behavior can occurred.
- **char[]** : The char array cannot be used in the current version of POP-Java with POP-C++ parallel classes.

6.2.3 Dealing with array

Passing arrays from POP-Java to POP-C++ is a bit tricky. As POP-Java and POP-C++ do not behave the same with array, the programmer must be aware of the way to pass the array. Here is a example of a method with an array as parameter.

The method declaration in POP-C++
In POP-C++, the programmer must give the array size to the compiler.

```
sync seq void changeIntArray(int n, [in, out, size=n] int *i);
```

Method declaration in POP-Java
As POP-C++ will need the size of the array, POP-Java must declare this size as well.

```
@POPSyncSeq
public void changeIntArray(int n, int[] i){}
```

Method call from POP-Java
In the POP-Java application, the programmer need to give the array size in the method call.

```
p.changeIntArray(iarray.length, iarray);
```

6.3 POP-Java application using POP-C++ parallel objects

This section will teach the programmer how to develop a POP-Java application with a POP-C++ parallel class. The same example of the parallel class Integer will be used. For more details about the POP-C++ programming please have a look to "Parallel Object Programming C++ - User and Installation Manual"[2]. In the following example, the main class used is the same as the one shown in the figure[8]. All the sources can be found in the directory "example/mixed1" of the POP-Java distribution.

6.3.1 Develop the POP-C++ parallel class

First, the programmer needs to write the parallel class in POP-C++ as it should be for a POP-C++ application. The figure[9] shows the header file of the parclass Integer. There is two rules to follow when the programmer develop a POP-C++ parallel class for POP-Java usage.

1. The parclass must declare a classuid.
2. The methods must be declared in alphabetics order.

Figure 9: File integer.ph

```
parclass Integer
{
  classuid(1000);
public:
  Integer();
  ~Integer();

  mutex void Add(Integer &other);
  conc int Get();
  seq async void Set(int val);

private:
  int data;
};
```

The figure[10] shows the implementation of the parallel class Integer. There is no important information in this file for the POP-Java usage.

Figure 10: File integer.cc

```
#include <stdio.h>
#include "integer.ph"
#include <unistd.h>

Integer::Integer() {
  printf("Create remote object Integer on %s\n",
                  (const char *)POPSystem::GetHost());
}

Integer::~Integer() {
  printf("Destroying Integer object...\n");
}

void Integer::Set(int val) {
  data=val;
}

int Integer::Get() {
  return data;
}

void Integer::Add(Integer &other) {
  data += other.Get();
}
@pack(Integer);
```

Compilation of the parallel class

Once the parclass implementation is finished, it can be compiles with the POP-C++ compiler. The following command will create an object executable of our parclass Integer.

```
popcc -object -o integer.obj integer.cc integer.ph
```

6.3.2 Create the partial POP-Java parallel class

To be used in a POP-Java application, a POP-C++ parallel class must have its partial implementation in POP-Java language. A partial implementation means that all the methods must be declared but does not need to be implemented.

The figure [11] shows the partial implementation of the parallel class Integer. All the methods are just declared. This partial implementation is a translation of the POP-C++ source code to POP-Java source code.

Remark : In the future version of POP-C++ and POP-Java, the partial implementation would be generated by the compiler. For the moment, the programmer will need to do it by hand.

Figure 11: File integer.cc

```
@POPClass
public class Integer {
  private int value;

  public Integer(){
  }

  @POPSyncMutex
  public void add(Integer i){
  }

  @POPSyncConc
  public int get(){
        return 0;
  }

  @POPAsyncSeq
  public void set(int val){
  }
}
```

6.3.3 Special compilation

To compile the partial POP-Java parallel class, the compiler needs some additional informations. The POP-Java compiler has an option to generate an additional informations XML file. To generate this file use the following command line :

```
popjc -x Integer.pjava
```

This command will generate a file (additional-infos.xml) in the current directory. This file is incomplete. The programmer will need to edit it with the informations of the POP-C++ parallel class. The figure [12] shows the file generated by the POP-Java compiler.

Figure 12: Generated file additional-infos.xml

```
<popjparser-infos>
  <popc-parclass file="Integer.pjava" name="" classuid=""
    hasDestructor="true"/>
</popjparser-infos>
```

The two empty attributes **name** and **classuid** must be informed with the value of the POP-C++ parallel class. The complete file must look like the figure [13].

Figure 13: Completed file additional-infos.xml

```
<popjparser-infos>
  <popc-parclass file="Integer.pjava" name="Integer" classuid="1000"
    hasDestructor="true"/>
</popjparser-infos>
```

All the informations to compile the POP-Java application are now known. Here is the command to compile it :

Compilation as .class files

```
popjc -p additional-infos.xml Integer.pjava TestInteger.pjava
```

Compilation as .jar file

```
popjc -j myjar.jar -p additional-infos.xml Integer.pjava
  TestInteger.pjava
```

6.3.4 Generate the object map

An object map is also needed for a POP-Java application using POP-C++ parallel classes. The programmer can generate this object map with the POP-Java application launcher and the option "–listlong". This option accept also the POP-C++ executable files. Here is the command used for the example application :

```
popjrun --listlong integer.obj > objmap.xml
```

Generated objmap.xml file (path and architecture can differ from the ones shown here):

```
<CodeInfoList>
  <CodeInfo>
    <ObjectName>Integer</ObjectName>
    <CodeFile>/home/clementval/pop/popjava-1.0/example/mixed/
      integer.obj</CodeFile>
    <PlatForm>i686-pc-Linux</PlatForm>
  </CodeInfo>
</CodeInfoList>
```

6.3.5 Running the application

To run the mixed application, the programmer needs to use the POP-Java application launcher. As the application main class is written in POP-Java, only this tool can run this application. Here is the command used to run the application :

```
popjrun objmap.xml TestInteger
```

The output of the example application should be like the following :

```
i1=23
i2=25
i1+i2=48
Test Integer Successful
```

If any problems occurred with the compilation or the launching of the application, please see the chapter 10.

6.4 POP-C++ application using POP-Java parallel objects

A POP-C++ application can also use POP-Java parallel classes. The following chapter shows how to develop, compile and run a POP-C++ using POP-Java parallel objects.

6.4.1 Developing and compiling the POP-Java parallel class

The POP-Java parallel class will be the same as the one shown in the figure[7]. The compilation will be a little bit different. As for a POP-Java application using a POP-C++ parclass, the POP-Java will need some additional informations during the compilation process. These additional informations must be given in a XML file. The POP-Java compiler can generate a canvas of this file with the option "-x". Here is the command we used :

```
popjc -x Integer.pjava
```

The genrated file will be similar to the one shown on the figure[12]. This time the attribute "name" must stay empty as we want to keep the real name of the POP-Java parallel class. The completed file should looks like the figure[14].

Figure 14: Completed file additional-infos.xml

```
<popjparser-infos>
  <popc-parclass file="Integer.pjava" name="" classuid="1000"
    hasDestructor="true"/>
</popjparser-infos>
```

This file can be given to the compiler to compile the parallel class with the following command :

```
popjc -p additional-infos.xml Integer.pjava
```

6.4.2 The POP-C++ partial implementation

As for the POP-Java application using POP-C++ parallel objects, the POP-C++ application will need a partial implementation of the parallel class in POP-C++. The header file will stay the same as the one shown in the figure[9]. The figure[15] shows the partial implementation of the POP-C++ parallel class. Once again, the method are declared but not implemented.

Figure 15: File integer.cc

```
#include <stdio.h>
#include "integer.ph"
#include <unistd.h>

Integer::Integer() {
  printf("Create remote object Integer on %s\n",
                  (const char *)POPSystem::GetHost());
}

Integer::~Integer() {
}

void Integer::Set(int val) {
}

int Integer::Get() {
  return 0;
}

void Integer::Add(Integer &other) {
}
@pack(Integer);
```

6.4.3 The POP-C++ main

To be able to run this application, a main function must be written. The figure[16] shows the file including the main of the POP-C++ application.

Figure 16: File main.cc

```
#include "integer.ph"
#include <iostream>
using namespace std;
int main(int argc, char **argv)
{
  try{
    // Create 2 Integer objects
    Integer o1;
    Integer o2;
    o1.Set(1); o2.Set(2);
    cout << endl << "o1="<< o1.Get() << "; o2=" << o2.Get() << endl;
    cout<<"Add o2 to o1"<<endl;
    o1.Add(o2);
    cout << "o1=o1+o2; o1=" << o1.Get() << endl << endl;
  } catch (POPException *e) {
    cout << "Exception occurs in application :" << endl;
    e->Print();
    delete e;
    return -1;
  } // catch
  return 0;
}
```

The main is very similar to the one used in POP-Java but this time it is written in POP-C++.

6.4.4 Object map

As the current version of POP-C++ is not able to generate the object map for a POP-Java parallel class, the programmer needs to edit the object map manually.

The code below is the canvas of the line to add in a POP-C++ object map for a POP-Java parallel class.

```
POPCObjectName *-* /usr/bin/java -cp POPJAVA_LOCATION
popjava.broker.Broker -codelocation=CODE_LOCATION
-actualobject=POPJAVAObjectName
```

Here is the line for the example (the path will be different on your computer) :

```
Integer *-* /usr/bin/java -cp /home/clementval/popj
popjava.broker.Broker
-codelocation=/home/clementval/pop/popjava-1.0/example/mixed2
-actualobject=Integer
```

6.4.5 Compile and run the POP-C++ application

The POP-Java parallel class is compiled and the object map is complete. The main and the partial implementation of the parallel class in POP-C++ must be compiled. The following command will compile our application :

```
popcc -o main integer.ph integer.cc main.cc
popcc -object -o integer.obj integer.cc integer.ph main.cc
```

Everything is compiled and we can run the application with the "popcrun" tool.

```
popcrun obj.map ./main
```

The output of the application should look like this :

```
popcrun obj.map ./main

o1=1; o2=2
Add o2 to o1
o1=o1+o2; o1=3
```

7 POP-Java plugin

The POP-Java system can be augmented by its users. If the programmer feels the need of a new network protocol or a new encoding protocol, he can create a POP-Java plugin and add it to the system easily. This chapter aims to present the combox and the buffer plugin systems.

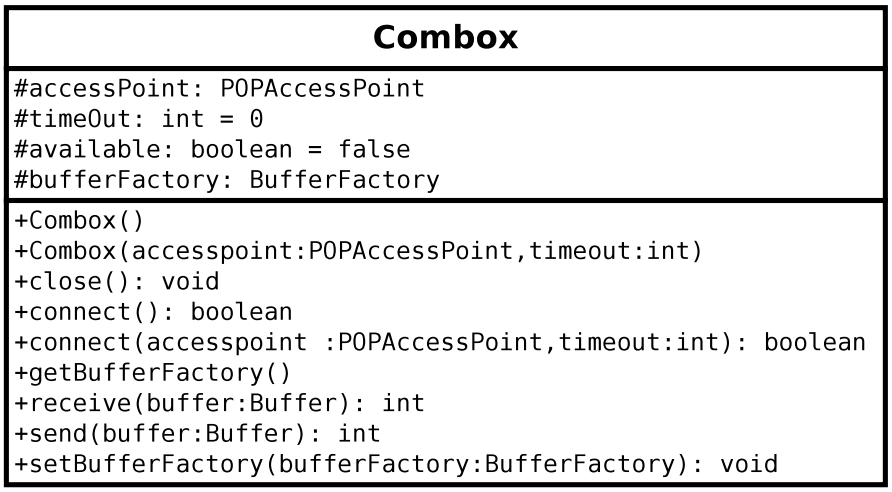
7.1 Combox plugin

The Combox is the component responsible for the network communication between an application and a parallel object or between two parallel objects. In the current version of POP-Java, only the protocol socket is implemented. If the programmer needs another protocol, he can create his own Combox.

To create a new protocol for POP-Java, the programmer needs to create three different classes : a combox, a combox server and a combox factory.

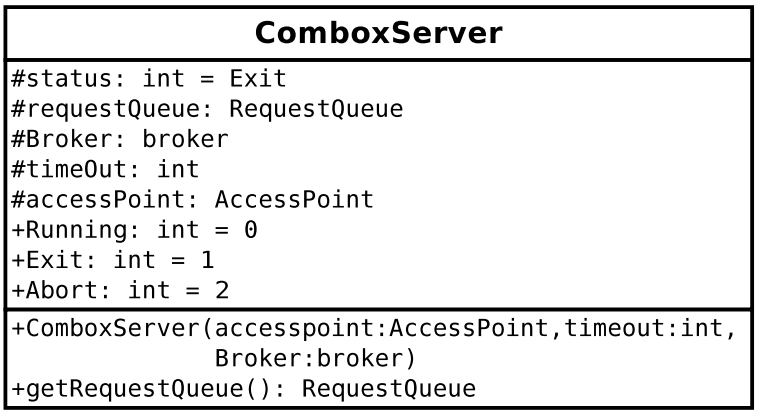
The combox must inherits from the super class ComboxPlugin located in the package popjava.combox in the POP-Java library. The figure[17] shows the ComboxPlugin class signature.

Figure 17: ComboxPlugin class signature



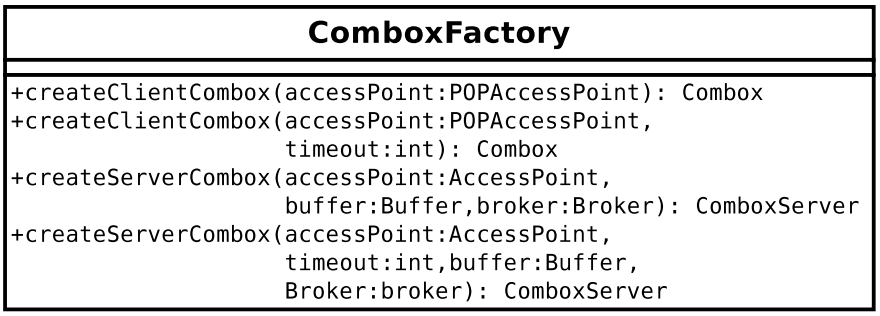
The combox server must inherits from the super class ComboxServer located in the package popjava.combox in the POP-Java library. The figure[18] shows the ComboxServer class signature.

Figure 18: ComboxServer signature



The combox factory must inherits from the super class ComboxFactory located in the package popjava.combox in the POP-Java library. The figure[19] shows the ComboxFactory class signature.

Figure 19: ComboxFactory class signature



Once all the classes are implemented, the programmer needs to compile them as standard Java code and create a JAR file. This JAR file can be added in the system by editing the file pop_combox.xml located in the directory *POPJAVA_LOCATION/plugin*. The XML code below is the current XML file with the socket protocol.

```
<ComboxFactoryList>
  <Package JarFile="popjava.combox.jar">
    <ComboxFactory>popjava.combox.ComboxSocketFactory</ComboxFactory>
  </Package>
</ComboxFactoryList>
```

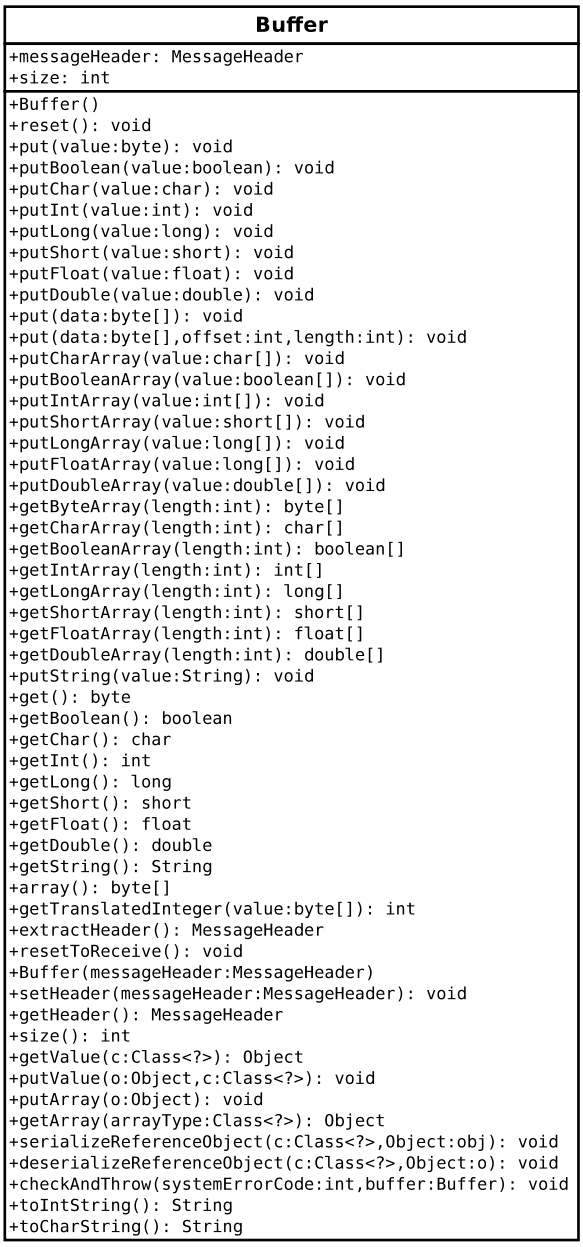
7.2 Buffer plugin

The buffer is the component in charge of the data encoding. In the current implementation of POP-Java, two buffers are available. One is using the RAW encoding and the other is using the XDR encoding. If the programmer needs a special encoding protocol, he can also create his own and add it to the POP-Java system as a plugin.

To implement a new encoding protocol, the programmer needs to create two class. A buffer and a buffer factory.

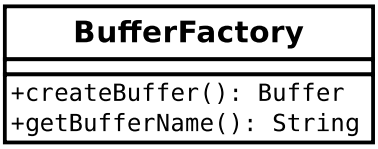
The buffer must inherits from the super class BufferPlugin located in the package popjava.buffer in the POP-Java library. The figure[20] shows the BufferPlugin class signature.

Figure 20: BufferPlugin class signature



The buffer factory must inherits from the super class BufferFactory located in the package popjava.buffer in the POP-Java library. The figure[21] shows the BufferFactory class signature.

Figure 21: BufferFactory class signature



8 Installation

To use POP-Java and POP-C++ on a computer we need to install them. This chapter helps the programmer to perform the correct installation of the POP system on a computer.

8.1 POP-C++ installation

In order to use POP-Java we need to install the latest version of POP-C++. This section will help us to get trough th installation process and make sure the installation is fine for a POP-Java usage.

8.1.1 Requirements

In order to install POP-C++ we need few additional software. The following packages are required before compiling :

- a C++ compiler (g++ or equivalent)
- zlib-devel (package name depends on distribution)
- Gnu Bison (optional)
- Globus Toolkit (optional)

8.1.2 Before installing

Before installation we should make the following configuration choices. In case of doubt the default values can be used.

- The compilation directory that should hold roughly 50MB. This directory will contain the distribution tree and the source files of POP-C++. It may be erased after installation.
- The installation directory that will hold less than 40M. It will contain the compiled files for POP-C++, include and configuration files. This directory is necessary in every computer executing POP-C++ program (default location /usr/local/popc).
- A temporary directory will be asked in the installation process. This directory will be used by POP-C++ to hold file during the application execution (default /tmp).
- Resource topology. The administrator must choose what computer form the grid.

For more informations about the POP-C++ installation and configuration process, please see "Parallel Object Programming C++ - User and Installation Manual"[2].

8.1.3 Installation process

This section will guide us trough the POP-C++ installation process. In the POP distribution we find a directory including POP-C++. First, we need to configure the installation. If we use the configure script without any option, POP-C++ will be installed in the default directory (/usr/local/popc). We can also specify the directory by using the option –prefix.

Default directory

```
./configure
```

Specific directory

```
./configure --prefix=/home/user/popc
```

Once the configuration script is done, we will need to compile the source of POP-C++ for our architecture. For this, we just need to run the make command in the root directory of POP-C++.

```
make
```

Finally, to install POP-C++, we need to run the install target of the make file. This script will guide us trough the installation. To be sure that our installation will fit the requirements of POP-Java, please follow the instructions below.

Answer "y" to the first question. We need to configure POP-C++ services.

```
make install
...
DO YOU WANT TO CONFIGURE POP-C++ SERVICES? (y/n)
y
```

We need to make a special installation so answer "n" to the second question.

```
...
DO YOU WANT TO MAKE A SIMPLE INSTALLATION ? (y/n) :
n
```

The answers to the questions below are up to our configuration but if we don't know our configuration just pass every question.

```
=====================================================
GENERATING SERVICE MAPS...
CONFIGURING POP-C++ SERVICES ON YOUR LOCAL MACHINE...
Enter the full qualified master host name (POPC gateway):

Enter the child node:

Enter number of processors available (default:1):

Enter the maximum number of POP-C++ jobs that can run concurrently
(default: 100):

Enter the available RAM for job execution in MB (default 1024) :

Which local user you want to use for running POP-C++ jobs?

CONFIGURING THE RUNTIME ENVIRONMENT
Enter the script to submit jobs to the local system:

Communication pattern:

SETTING UP RUNTIME ENVIRONMENT VARIABLES
Enter variable name:
```

We need the startup script to use the global runtime service with POP-Java so answer "y" to the question "Do you want to generate the POP-C++ startup scripts?".

```
=====================================================
CONFIGURATION POP-C++ SERVICES COMPLETED!
=====================================================
Do you want to generate the POP-C++ startup scripts? (y/n)
y
```

Depends on our configuration, we can modify the default values of the startup script or just keep them. One important thing is to copy the environment variables on the .bashrc or equivalent file.

```
=====================================================
CONFIGURING STARTUP SCRIPT FOR YOUR LOCAL MACHINE...
Enter the service port[2711]:

Enter the domain name:

Enter the temporary directory for intermediate results:

=====================================================
CONFIGURATION DONE!
=====================================================

IMPORTANT : Do not forget to add these lines to your .bashrc
file or equivalent :
---------
    POPC_LOCATION=/home/clementval/popc
    PATH=\$PATH:\$POPC_LOCATION/bin:\$POPC_LOCATION/sbin

Press <Return> to continue
```

The POP-C++ installation is done. We can now use POP-C++ and also install POP-Java.

8.1.4 System startup

Before executing any POP-C++ application, the runtime system (Job manager and resource discovery) must be started. There is a script provided for that purpose, so every node must run the following command:

```
POPC_LOCATION/sbin/SXXpopc start
```

SXXpopc is a standard Unix deamon control script, with the traditional start, stop and restart options.

8.2 POP-Java installation

This section will guide us trough the POP-Java installation process.

8.2.1 Requirements

In order to install POP-Java, some packages are required. Here is the list of required packages :

- JDK 7 or higher
- POP-C++ 2.5 or higher
- JavaCC (optional)
- Apache ANT (optional)

8.2.2 Installation process

To install POP-Java we need to launch the command **ant** int the POP-Java directory. Once the source code is compiled, launch the installation with the install script: **sudo ./install**. This script will guide us trough the installation by asking us some questions. Be aware that if we install POP-Java in the default location we need the administrator rights. Please use the option "-E" with the sudo command to keep the environment variables.

Here is the output we should have on our shell :

```
[POP-Java installation]: Detecting java executable ...
[POP-Java installation]: Java executable detected under
  /usr/bin/java
[POP-Java installation]: Please enter the location of your desired
  POP-Java installation (default: /usr/local/popj ) :
/home/clementval/popj
[POP-Java installation]: Installing POP-Java under
  /home/clementval/popj ? (y/n)
y
[POP-Java installation]: Copying files ...
[POP-Java installation]: Generating configuration files ...
[POP-Java installation]: Generating object map file for the test suite
[POP-Java installation]: POP-Java has been installed under
  /home/clementval/popj. Please copy the following lines into your
  .bashrc files or equivalent

POPJAVA_LOCATION=/home/clementval/popj
export POPJAVA_LOCATION
POPJAVA_JAVA=/usr/bin/java
export POPJAVA_JAVA
PATH=$PATH:$POPJAVA_LOCATION/bin

[POP-Java installation]: Installation done.
```

At the end of the installation, the script ask us to copy some environment variable declarations in the .bashrc or equivalent file. This step is mandatory to make POP-Java work correctly.

8.3 Test the installation

POP-Java includes a test suite. We can run this test suite to check if our POP system is correctly installed. To run this test suite, we need to launch the "launch_testsuite" script located in the POP-Java installation location.
Here is the output we should get after the completion of the test suite :

```
./launch_testsuite
########################################
#   POP-Java 1.0 Test Suite started    #
########################################
POP-C++ detected under /home/clementval/popc
POP-C++ was not running. Starting POP-C++ runtime global services ...
Starting POPC Job manager service:
POPCSearchNode access point: socket://172.28.10.67:38331
Starting Parallel Object JobMgr service
socket://172.28.10.67:2711POP-C++ started
##############################
#   POP-Java standard test   #
##############################
Starting POP-Java test suite
Launching passing arguments test (test 1/6)...
Arguments test successful
Passing arguments test is finished ...
Launching multi parallel object test (test 2/6)...
Multiobjet test started ...
Result is : 1234
Multiobjet test finished ...
Multi parallel object test is finished...
Launching callback test (test 3/6)...
Callback test started ...
Identity callback is -1
Callback test successful
Callback test is finished...
Launching barrier test (test 4/6)...
Barrier: Starting test...
Barrier test successful
Barrier test is finished...
Launching integer test (test 5/6)...
i1 = 23
i2 = 25
i1+i2 = 48
Test Integer Successful
Integer test is finished...
Launching Demo POP-Java test (test 6/6)...
START of DemoMain program with 4 objects
Demopop with ID=1 created with access point : socket://127.0.1.1:39556
Demopop with ID=2 created with access point : socket://127.0.1.1:60575
Demopop with ID=3 created with access point : socket://127.0.1.1:50088
Demopop with ID=4 created with access point : socket://127.0.1.1:39475
Demopop:1 with access point socket://127.0.1.1:39556 is sending his ID to object:2
Demopop:2 receiving id=1
Demopop:2 with access point socket://127.0.1.1:60575 is sending his ID to object:3
Demopop:3 receiving id=2
Demopop:3 with access point socket://127.0.1.1:50088 is sending his ID to object:4
Demopop:4 receiving id=3
Demopop:4 with access point socket://127.0.1.1:39475 is sending his ID to object:1
Demopop:1 receiving id=4
END of DemoMain program
Demo POP-Java test is finished...

###################################
#   POP-C++ interoperability test  #
###################################
popcc -o main integer.ph integer.cc main.cc
popcc -object -o integer.obj integer.cc integer.ph main.cc
./integer.obj -listlong > obj.map
Launching POP-C++ integer with POP-Java application test (test 1/2)
POPC Integer test started ...
```

```
o1 = 10
o2 = 20
10 + 20 = 30
POPC Integer test successful
POP-C++ integer with POP-Java application test is finishied ...
popcc -parclass-nobroker -c integer2.ph
popcc -o main integer2.stub.o integer.ph integer.cc main.cc
popcc -parclass-nobroker -c integer2.ph
popcc -object -o integer.obj integer2.stub.o integer.cc integer.ph
popcc -object -o integer2.obj integer2.cc integer2.ph
./integer.obj -listlong > obj.map
./integer2.obj -listlong >> obj.map
Launching Integer mix (POP-C++ and POP-Java) with POP-Java application test(test 2/2)
i=20
j=12
i+j=32
Integer mix (POP-C++ and POP-Java) with POP-Java application test is finishied ...
########################################
#   POP-Java 1.0 Test Suite finished   #
########################################
Stopping POPC Job manager service...
Connecting to 172.28.10.67:2711....
POPCSearchNode stopped
JobMgr stopped
```

9 Documentation and examples

9.1 Documentation

All the library is documented under the Javadoc format. This documentation is installed on the computer after the completion of the POP-Java installation. A generated documentation under the standard Javadoc and also under the Doxygen format is available. The "index.html" file is located in the folder *POPJAVA_LOCATION*/doc/index.html gives acces to both Javadoc and Doxygen documentation.

9.2 Examples

The POP-Java distribution includes some examples of POP-Java application. These examples can be found in the folder *POPJAVA_DISTRIBUTION*/example. All examples have a makefile to compile and a special target "run" to run these examples. The following examples are available for the moment :

- Callback example : this example shows the ability of a parallel object to call back the one who called him.
- Integer example : this is a simple example of a parallel object integer (same as the example in POP-C++).
- Mixed1 example : this example is a POP-Java application using a POP-C++ integer parallel object
- Mixed2 example : this example is a POP-C++ application using a POP-Java integer parallel object.
- Multiobj example : this example shows a chaining of parallel object.

10 Troubleshooting

10.1 POP-Java exception

This section lists some of the main POP-Java exception that can occurred during the application execution and gives the cause of the problem.

10.1.1 Cannot bind to access point : socket://your-computer-name:2711

This exception occurred when the application cannot contact the POP-C++ runtime system. To fix this problem, we need to start the POP-C++ runtime system with the following command :

```
POPC_LOCATION/sbin/SXXpopc start
```

10.1.2 Error message : OBJECT_EXECUTABLE_NOTFOUND

This exception occurred when the executable file is not found. This might be due to a bad object map or the deletion of the object executable file. To fix this problem we should generate a new object map with the object executable.

10.1.3 Error message : NO_RESOURCE_MATCH

This exception occurred when no resource match the requirements of a specific object. To fix this problem we should check the object descriptions in the parallel objects. We might have put a too high requirements for a parallel object creation.

10.1.4 Error message : Cannot run program "/usr/local/popc/services/appservice"

If we get an error with "cannot run program" and the path contains the appservice of POP-C++, sou have certainly reinstalled POP-C++ and the configuration file of POP-Java is now wrong. The easiest way to fix this problem is to reinstall also POP-Java. We can also edit the configuration file under *POP-JAVA_LOCATION*/etc/popj_config.xml. The item **popc_appcoreservice_location** must be modified with the good path.

10.1.5 Test suite frozen

If the test suite seems to be frozen, we should abort the test suite and restart the POP-C++ global service with the following command

```
POPC_LOCATION/sbin/SXXpopc restart
```

11 Table of figures

12 References

[1] Clément Valentin, *POP-Java - Technical Report*. EIA-FR, Fribourg, Switzerland, August 2010.

[2] The POP-C++ Team, *Parallel Object Programming C++, User and installation Manual*. Grid and Ubiquitous Computing Group, EIA-FR, Fribourg, Switzerland, 2010.

[3] Dr. Thuan-Anh Nguyen, *An object-oriented model for adaptive high performance computing on the computational GRID*. EPFL, Lausanne, Switzerland, 2005.

[4] W. Allcock, J. Bester, J. Bresnahan, A. Chervenak, L. Liming, S. Meder, and S. Tuecke. *GridFTP Protocol Specification*. GGF GridFTP Working Group Document, September 2002. http://www.globus.org/research/papers.htm.

[5] K. Czajkowski, I. Foster, N. Karonis, C. Kesselman, S. Martin, W. Smith, and S. Tuecke. A resource management architecture for metacomputing systems. In *Proc. IPPS/SPDP '98 Workshop on Job Scheduling Strategies for Parallel Processing*, pages 62–82, 1998.

[6] I. Foster and N. Karonis. A grid-enabled mpi: Message passing in heterogeneous distributed computing systems. In *Proc. 1998 SC Conference*, November 1998.

[7] I. Foster and C. Kesselman. Globus: A metacomputing infrastructure toolkit. *Intl J. Supercomputer Applications*, 11(2):115–128, 1997.

[8] I. Foster, C. Kesselman, J. Nick, and S. Tuecke. Grid services for distributed system integration. *Computer*, 35(6), 2002.

[9] Andrew Grimshaw, Adam Ferrari, Fritz Knabe, and Marty Humphrey. Legion: An operating system for wide-area computing. *IEEE Computer*, 32:5:29–37, May 1999.

[10] N. Karonis, B. Toonen, and I. Foster. MPICH-G2: A grid-enabled implementation of the message passing interface. *Journal of Parallel and Distributed Computing*, 2003.

[11] Kuonen P. Nguyen, T. A. Programming the grid with pop-c++. *Future Generation Computer Systems (FGCS)*, 23(1):23–30, January 2007.

[12] Tuan-Anh Nguyen. *An Object-oriented model for adaptive high performance computing on the computational Grid*. PhD thesis, Swiss Federal Institute of Technology-Lausanne, 2004.

[13] Object Management Group, Framingham, Massachusetts. *The Common Object Request Broker: Architecture and Specification — Version 2.6*, December 2001.

[14] A. Roy, I. Foster, W. Gropp, N. Karonis, V. Sander, and B. Toonen. MPICH-GQ: Quality-of-service for message passing programs. In *Proc. of the IEEE/ACM SC2000 Conference*, November 2000.

[15] H. Stockinger, A. Samar, B. Allcock, I. Foster, K. Holtman, and B. Tierney. File and object replication in data grids. In *10th IEEE Symposium on High Performance and Distributed Computing (HPDC2001)*, 2001. San Francisco, California.

[16] Weiqin Tong, Jingbo Ding, and Lizhi Cai. A parallel programming environment on grid. In *International Conference on Computational Science 2003*, pages 225–234, 2003.

[17] V. Welch, F. Siebenlist, I. Foster, J. Bresnahan, K. Czajkowski, J. Gawor, C. Kesselman, S. Meder, L. Pearlman, and S. Tuecke. Security for grid services. In IEEE Press, editor, *Twelfth International Symposium on High Performance Distributed Computing (HPDC-12)*, 2003.

[18] Tiffani L. Williams and Rebecca J. Parsons. The heterogeneous bulk synchronous parallel model. *Lecture Notes in Computer Science*, 1800, 2000.

A POP-Java compiler command

```
POP-Java Compiler v1.0

This program is used to compile a POP-Java program

Usage: popjc <options> <source files>

OPTIONS:
   -h, --help                Show this message
   -n, --noclean             Do not clean the intermediate Java file
                             generated by the POP-Java parser
   -p, --popcpp <xml_file>   Compile a POP-Java parallel class for
                             POP-C++ usage (Need XML additional
                             informations file)
   -j, --jar <filename>      Create a JAR archive with the given name
                             (Need the JAR file name)
   -v, --verbose             Verbose mode
   -c, --classpath <files>   Include JAR or compiled Java class to the
                             compilation process. Files must be
                             separated by a semicolon ":"

OPTIONS FOR POP-C++ INTEROPERABILITY:
   -x, --xmlpopcpp <files>   Generate a canvas of the POP-C++ XML
                             additional informations file for the
                             given Java files. This option must be
                             used alone.
   -g, --generate <pjava>    Generate the POP-C++ partial
                             implementation to use the given
                             POP-Java parclass in a POP-C++ application
                             (NOT IMPLEMENTED YET)
```

B POP-Java application launcher command

```
POP-Java Application Runner v1.0

This program is used to run a POP-Java application or to generate
object map

Usage: popjrun <options> <objectmap> <mainclass>

OPTIONS:
   -h, --help                 Show this message
   -v, --verbose              Verbose mode
   -c, --classpath <files>    Include JAR or compiled Java class
                              needed to run the application. Files must
                              be separated by a semicolon ":"

OPTIONS FOR OBJECT MAP GENERATION:
   -l, --listlong <parclass> Generate the object map for the given
                              parclasses. Parclasses can be a .class,
                              .jar, .obj or .module file. Parclasses
                              must be separated by :
```