Object-Oriented Group Communication for Java Grid Middleware

LAURENT BADUEL †, FRANÇOISE BAUDE ‡, and DENIS CAROMEL ‡

Providing a middleware for Grid computing with an effective and efficient implementation of the group abstraction at programming level could ease software development and reduce the communication overhead. Performances of programs on distributed memory parallel machines are highly dependent of the efficiency of interprocess communications. Parallel programming environments often offer poor support for high level communication models, especially in object-oriented contexts. Our research deals with high level group communication in such architectures and the programming models it provides.

1. Introduction and context

Grid applications deal with intensive computations and management of huge amount of data which have to be transferred and processed on multiple resources.

For few years, the interest in using Java for high-performance computing has increased. The RMI mechanism is the standard point-to-point communication mechanism, and is adequate for client-server interactions, via synchronous remote method calls.

In a high-performance computing context, asynchronous and collective communications should be accessible to programmers, so the usage of RMI is not sufficient.

According to the Object Group design pattern [1], we name a group a local surrogate for a set of objects distributed across networked machines to which can be assigned the execution of a task.

2. The *ProActive* middleware

ProActive is a Java library for parallel and distributed computing, also featuring mobility and security. As *ProActive* is built on top of the Java standard API, it does not require any modification to the standard Java execution environment, nor need of a special compiler, pre-processor or modified virtual machine.

A distributed application is then composed of medium-grained entities called *active objects*. Each active object has its own thread of control and is granted the ability to decide in which order to serve the incoming method calls. Method calls sent to active objects are asynchronous with transparent *future objects*. and synchronization is handled by a mechanism known as *wait-by-necessity* [2].

Let us take a standard Java class `A`. The following instruction creates a new active object of type `A` on the (remote) JVM identified with `node`, (for instance `rmi://lo.inria.fr/node`):

```
A a = (A) PA.newActive ("A",params,node);
```

Further, all calls to that remote object will be asynchronous.

```
v = a.foo();   // Asynchronous call
```

† Tokyo Institute of Technology,
#201 West Building 7, 2-12-1 Ookayama Meguro-ku, Tokyo 152-8552, Japan
Email: `baduel@smg.is.titech.ac.jp` Tel & Fax: +81 3 5734 3881

‡ INRIA - CNRS - University of Nice Sophia-Antipolis,
2004 Route des Lucioles, BP93, 06902 Sophia Antipolis Cedex, France
Email: `First.Last@sophia.inria.fr` Tel: +33 4 92 38 75 56 Fax: +33 4 92 38 76 44

```
...
v.bar();    // wait until v gets its value
```

Compared to traditional futures, *wait-by-necessity* offers two important features: futures are created implicitly and systematically, and futures can be passed to other remote processes.

3. Typed group communication

Our group communication mechanism efficiently achieves asynchronous remote method invocation for a group of remote objects, with automatic gathering of replies. Given a Java class, one can initiate group communications using the standard public methods of the class together with the classical dot notation; in that way, group communications remains *typed*. On the class A, here is a group creation:

```
    // A group of type "A" and its 3 members
    // are created at once on the nodes
  Object[][] params = {{...}, {...}, {...}};
  A ag = (A) PAGroup.newGroup("A", params,
                        {node1,node2,node3});
```

Elements can be included into a typed group only if their class equals or extends the class specified at the group creation. Based on Java typing, only the methods defined in the class specified at the group creation can be invoked.

A method invocation on a group has a syntax similar to a standard method invocation:

```
V vg = ag.foo(); // A group communication
```

The *result* of a typed group communication is a group transparently built at invocation time, with a future for each reply. It will be dynamically updated with the incoming results.

Other features are available regarding group communications: broadcast, scatter, gather, hierarchical groups, dynamic group manipulation, synchronization, etc. see [3].

4. Object-Oriented SPMD

SPMD stands for Single Program Multiple Data. SPMD programming is a common way to organize a parallel program. The group mechanism is already a usable and efficient basis to program non-embarrassingly parallel applications using a pure object-oriented paradigm, but some of the features specific to SPMD programming were lacking:

- The identification of each member taking part in the parallel computation.
- The expression of the program run by each member.
- The expression of global synchronization barriers.

By the addition of those properties in the typed group communication, the Object-Oriented SPMD becomes an alternative to the standard message-based SPMD programming model. While being placed in an object-oriented context, the mechanism helps with the definition and the coordination of parallel and distributed activities. The approach offers, through modest expansion of the group API, structuring flexibility and innovative implementation thanks to the benefits from the object-oriented programming model. Details of the implementation and advanced features such as postponed barriers and topology objects are introduced in [4]. The automation of key communication mechanisms and synchronization simplifies the writing of the code for the parallel activities.

5. Conclusion

The resulting OO-SPMD API already forms part of the *ProActive* open-source library, freely distributed through the Object Web consortium for open-source middleware. Our ambition is to have this approach used on real size applications.

Bibliography

[1] S. Maffeis, "The Object Group Design Pattern" in the 2nd USENIX Conference on Object-Oriented Technologies, Toronto, Canada, 1996.

[2] D. Caromel, "Towards a Method of Object-Oriented Concurrent Programming", Communications of the ACM, 1993.

[3] L. Baduel, F. Baude, C. Caromel, "Efficient, Flexible, and Typed Group Communications in Java", JavaGrande ISCOPE, Seattle, Washington, USA, 2002.

[4] L. Baduel, F. Baude, C. Caromel, "Object-Oriented SPMD", International Symposium on Cluster Computing and the Grid, Cardiff, UK, 2005.