An Open Framework for Reliable Distributed Computing

Benoît Garbinato {*Benoit.Garbinato@epfl.ch*}
and
Rachid Guerraoui {*Rachid.Guerraoui@epfl.ch*}

Laboratoire de Systèmes d'Exploitation
ÉcolePolytechnique Fédérale de Lausanne, Suisse

This paper gives an overview of an open object oriented framework for reliable distributed computing named BAST. BAST provides abstractions for building reliable distributed protocols such as atomic commitment and total order broadcast. It is an object oriented framework in the sense that reliable distributed protocols can be built simply through inheritance and callback implementations. It is open in the sense that both existing protocols and their implementations can be customized for specific application needs.

1. ABSTRACTIONS FOR RELIABLE DISTRIBUTED COMPUTING

Reliable distributed systems are challenging to build because programmers have to deal with many complex issues such as reliable communication, failure detection, replication, and consistency management. A *reliable* distributed environment is one that provides abstractions for hiding failures by ensuring continuous availability (liveness), and preventing failures from putting the application in a inconsistent state (safety). Two main paradigms for building reliable distributed systems have emerged over the last decades: the *transaction paradigm* [Bernstein et al. 1987] in databases, and the *group paradigm* [Birman and Van Renesse 1993] for client-server main memory applications. Although useful in the development of distributed systems, current reliable distributed environments such as group communication toolkits or transactional monitors, are hardly extensible or customizable for specific application needs. The limitations of those systems are mainly due to their underlying distributed protocols, which are usually designed and implemented in an *ad hoc* manner.

Address: IN-Ecublens, CH-1015 Lausanne, Switzerland.

The aim of the BAST[1] research project was to design and implement a set of minimal abstractions that can easily be composed to support both client-server main memory and database-like applications [Guerraoui and Schiper 1995].

2. PROTOCOLS AS FIRST CLASS CITIZENS

As far as reliability is concerned, the intrinsic robustness of protocol implementations is an important issue and reusing well-tested code is a key feature. Because relationships between protocols can be quite complex, structuring this complexity to reuse existing code is fundamental. In BAST, protocols are the basic structuring components of distributed systems and they are manipulated as *object classes*. Such classes make it very easy to use, extend, and customize abstractions provided by BAST. Manipulating protocols as objects allows to structure reliable distributed systems according to their needs in protocols, and simplifies the building of new complex protocols [Garbinato et al. 1996; Garbinato and Guerraoui 1998].

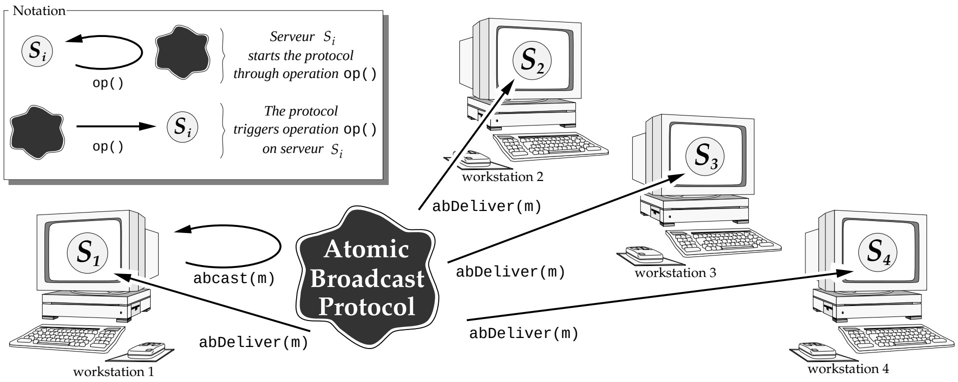

Fig. 1. Replication with Protocol Class `ABCObject`

3. READY-TO-USE PROTOCOLS

In BAST, a *distributed protocol* π is a set of coordinated interactions between distributed objects which aim at solving *problem* π. A protocol class π`Object` implements the behaviour of distributed objects capable of executing protocol π, i.e., of solving distributed problem π.

Protocol objects allow to put complex protocols to work with little effort. To illustrate this, consider an application which objects are running on a network of four workstations, and suppose we want object S to be available even when some workstations crash. We replicate object S on all four workstations, i.e., S is a replicated object and is manipulated as a group of replica objects $g_s = \{S_1, S_2, S_3, S_4\}$. So, users can go on with their work although some other workstation crashed. We also need total order broadcast communications on group g_s to ensure that replicas keep a consistent state. What we need here is a protocol that solves the *atomic*

[1]In the Egyptian mythology, *Bast* is a cat-goddess.

broadcast problem. BAST provides such a protocol class, namely `ABCObject`, which defines operation `abcast()` and callback `abDeliver()`. By subclassing `ABCObject` and by making each replica S_i an instance of that new class, we actively replicate S: callback operation `abDeliver()` of such subclass is implemented in order to update each S_i in a deterministic way. Operations `abcast()` and `abDeliver()` can also be used to atomically transfer the most recent state to a replica that is recovering after a crash [Guerraoui and Schiper 1997].

Figure 1 shows what happens after some user action on workstation 1 generated an update for replicated object S. The application does not directly update local replica S_1; instead, it builds an update message `m` and invokes operation `abcast()` on S_1, passing it `m`. This operation starts an atomic broadcast protocol involving replicas S_1, S_2, S_3 and S_4. Eventually, callback operation `abDeliver()` is triggered by the protocol on each replica, including S_1. This operation is implemented in `ABCObject`'s subclass and performs the actual update, using message `m`.

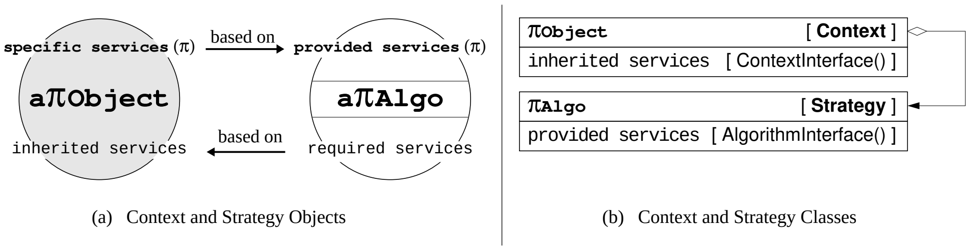

Fig. 2. Strategy Design Pattern in BAST

4. AN OPEN FRAMEWORK

There are basically two ways to build new protocol classes with BAST: (1) by customizing a protocol class implementing a *distributed pattern*, or (2) by composing existing protocols using the *Strategy design pattern*. The first method is adequate when the newly built protocol can be seen as an instance of another more general protocol (the distributed pattern). We do not illustrate this method here; details can be found in [Garbinato and Guerraoui 1998]. The second method is adequate when the new protocol relies on protocols that are already available in BAST, as we shall see now.

In BAST, we separate protocol classes from their implementations and manipulate the corresponding distributed algorithms as first-class objects. This allows us to "*define a family of algorithms, encapsulate each one, and make them interchangeable*" [Gamma et al. 1995, page 315], following the Strategy pattern. Distributed algorithms are thus encapsulated into *strategy* objects, which are used by *context* objects, i.e., protocol objects. Consequently, each π`Object` protocol class is independent of the algorithm supporting protocol π, which makes protocol composition fully flexible. Algorithm objects of class π`Algo` are privately used by the corresponding π`Object` protocol objects and hold the actual implementation of distributed algorithm π.

Figure 2a sketches the way protocol objects and algorithm objects interact. On the left side, protocol object `a`π`Object` offers the services it inherits from its superclasses, as well as the new services that are specific to protocol π. The actual algorithm implementing protocol π is not part of `a`π`Object`'s code; instead, `a`π`Object` uses services provided by strategy `a`π`Algo` (right side of Figure 2a). Whenever an operation related to protocol π is invoked on `a`π`Object`, the execution of the protocol is delegated to strategy `a`π`Algo`. In turn, the services required by the strategy to run protocol π are based on the inherited services of context `a`π`Object`. Such required services identify entry point operations to underlying protocols needed to solve problem π. Figure 2b presents the relationship between classes π`Object` and π`Algo`, using a class diagram based on the OMT notation.

5. CONCLUDING REMARKS

BAST is an object oriented framework of reliable distributed protocols. It provides protocols like total order broadcast and atomic commitment, supporting the development of reliable distributed applications. Other protocols such as group membership or view synchronous communication can easily be built by composing lower-level protocols provided by BAST, such as failure detection, reliable broadcast and consensus. Our implementation of BAST was written in Smalltalk and later ported to Java. It is used for teaching reliable distributed systems, as well as for prototyping new reliable distributed protocols. Our Smalltalk prototype is currently available on the web as freeware, at *http://lsewww.epfl.ch/bast*. A full documentation and several companion papers can also be found there.

REFERENCES

BERNSTEIN, P., HADZILACOS, V., AND GOODMAN, N. 1987. *Concurrency Control and Recovery in Distributed Database Systems*. Addison-Welsey.

BIRMAN, K. AND VAN RENESSE, R. 1993. *Reliable Distributed Computing with the Isis Toolkit*. IEEE Computer Society Press.

GAMMA, E., HELM, R., JOHNSON, R., AND VLISSIDES, J. 1995. *Design Patterns, Elements of Reusable Object-Oriented Software*. Addison-Wesley.

GARBINATO, B., FELBER, P., AND GUERRAOUI, R. 1996. Protocol classes for designing reliable distributed environments. In *European Conference on Object-Oriented Programming Proceedings (ECOOP'96)*, Volume 1098 of *Lecture Notes in Computer Science*, Linz (Austria), July 1996, pp. 316–343. Springer Verlag.

GARBINATO, B. AND GUERRAOUI, R. 1998. Flexible protocol composition in Bast. In *IEEE International Conference on Distributed Computing Systems (ICDCS'98)*, Amsterdam (Netherland), May 1998, to appear.

GUERRAOUI, R. AND SCHIPER, A. 1995. Transactional model vs Virtual Synchrony model: bridging the gap. *In Theory and Practice in Distributed Systems*, Springer Verlag, LNCS 938, 1995), pp. 121–132.

GUERRAOUI, R. AND SCHIPER, A. 1997. Software based replication for fault-tolerance. *IEEE Computer 30*, 4 (April), 68–74.