Component Quality Configuration: the Case of Replication

Vania Marangozova

SARDES Project, INRIA Rhône-Alpes, 655 avenue de l'Europe,
Montbonnot 38334 St Ismier cedex, France

{Vania.Marangozova}@inrialpes.fr

Abstract. In this paper we propose a model for replication configuration in component systems which focuses on two main issues: replication management configuration and component replication. Replication management configuration allows to execute applications in different contexts and thus responds to the growing multiplication of execution platforms. Component replication completes existing work on component system management which consider mainly transactions, persistence and security.

We propose a model of replication management which represents a replication protocol as a component-based service. It allows to develop protocols in an application independent manner and also to develop applications without replication constraints. We detail the model and report on our experience with a prototype implementation where we discuss configuration as well as performance issues.

1 Motivation and Objectives

Largely used for fault tolerance, access performance and scalability, replication is faced today with new challenges in the distributed computing domain. Recent technology developments bring new execution platforms which need specific replication management solutions. Knowing that there is no universal replication protocol, there is a need of *a configurable replication service*. Given the growing diversification of execution platforms, there is a need of *rapid protocol production*.

- *Replication configuration*. The multiplication of execution platforms brings the problem of the execution of the *same* application in different contexts. As different contexts imply different availability requirements, applications need context specific replication management. If there is no possibility for replication configuration, each time a new execution environment is considered, applications will need to be reimplemented.
- *Replication protocol production*. Execution platforms' diversification is an issue not only to applications but also to replication management protocols. They need to be able to satisfy applications' requirements in a growing variety of environments. They need, in consequence, mechanisms allowing the rapid construction of adapted protocols.

In order to respond to the above needs, we propose a component-based solution which focuses on the following issues:

- *Reuse of application business code*. We aim at providing a solution which allows applications to use different replication protocols without requiring business code modifications. Thus, replication management can be configured without application reengineering.
- *Reuse of replication management code*. We aim at providing a solution which allows the use of the same replication protocol for the needs of different applications. Moreover, the proposed solution should allow the reuse of parts of existing protocols for the construction of new ones. The first point overcomes the limitation of protocols' domain specificity and to seamlessly benefit from replication management solutions defined in different application domains. The second point is relevant to the protocol production process as it facilitates protocol development.
- *Non intrusion*. In order to allow the reuse of application business code, as well as of replication management code, our solution aims at defining a model where applications and protocols may be composed in a non intrusive way.

The problem of component-based replication management is part of a larger problem related to the configuration of the system services used by components. This configuration defines the quality attributes of components as it is their use of system services that defines their performance during execution. The issue of replication configuration is also related to the problems of component variability and independent extensibility. In fact, replication configuration is a kind of variability which is needed by applications which have to execute in different contexts. Our objective is to allow this variability without constraining applications' nor replication protocols' development. We are interested in providing mechanisms which allow the reuse and therefore the agile development of replication protocols.

In this paper we describe the component-based model we propose for replication and report on our experience with this model's implementation. After a short overview of related work (Section 2), we introduce the principle of our component-based replication management (Section 3), present the models used for application and protocol management (Section 4) and discuss the results of our prototype implementation (Section 5). We conclude the paper with a discussion on the pertinence of the approach and on the future perspectives of this work (Section 6).

2 Related Work

As the problem treated in this paper is about replication management in the component domain, we can compare our work either to projects focusing on components or to work on replication management.

2.1 Projects focusing on components

Among the projects focusing on component management, those explicitly considering system management are mainly the emerging standards EJB [15] and CCM [12]. However, if the two specifications consider transactions, security and persistence, they do not treat replication management. Some of their implementations do provide replication management for fault tolerance and load balancing but the proposed solutions are ad hoc and are not configurable. Examples of EJB implementations which provide these features are Oracle9iAsS [13] and BEA WebLogic [1]. Examples of CCM implementations with these features are iCMG [10] and EJCCM [3].

2.2 Projects focusing on replication management

Existing works on replication management do consider the replication configuration objective. However, the proposed solutions are not component-oriented, are domain specific, propose a limited set of protocols and do not provide tools for easy protocol construction. For example, the Munin project [2] focuses on distributed shared memory and defines four caching protocols. Applications can choose between those or define their own protocols using low-level mechanisms like locks, memory management primitives, etc. In GARF [7], the protocols manage fault tolerance in connected environments and are based on message passing primitives. Finally, TACT [16] allows to parameter database consistency but applications are obliged to stick to a predefined replication management model.

As far as the issue of protocol production is concerned, it has been considered by very few projects. Most of the research efforts have defined replication and consistency management as a part of the communication layer but have not specified the organization of the related treatments. For example, Subcontract [9] defines replication and consistency treatments in modules called *contracts* whose internal structure is hidden. Globe [14] defines a *replication sub-object* which encapsulates some specific protocol implementation. However, the behaviour of the replication sub-object is defined in very general terms and there are no indications for the protocol programmer.

Among the projects which have investigated in more detail the organization of replication protocols are BAST [5] and PassiveReplicator [6]. BAST models protocols as object graphs where each object encapsulates some algorithmic treatment which can be reused in other protocols. PassiveReplicator defines a three-layer structure based on some generic interfaces for consistency management. Both projects provide models which define explicitly protocols' structure and thus facilitate protocol

construction. However, their solutions focus exclusively on protocols and do not consider the question of the integration of these protocols in applications.

3 Component-Based Replication: the Principle

Our proposition is guided by the principles of aspect separation and of component architectures.

- *Aspect separation*. Aspect separation [11] is needed in order to define applications' business code and protocols' management code in an independent manner and thus achieve the goals of reuse and of non intrusion. In our proposition, aspect separation is achieved thought the use of component *containers* (Figure 1.). Introduced by a majority of component platforms such as the emerging standards EJB and CCM, containers are responsible of the system management of a component. The business code is provided in components and can be reused in different execution contexts thanks to different containers which provide appropriate system management.

  The decision to place replication and consistency management in a container is not sufficient as it does not solve the question of the structure of the corresponding treatments. In order to provide an answer to this problem, we apply the principles of component architectures to the case of replication treatments. The idea is introduced in the following paragraph.

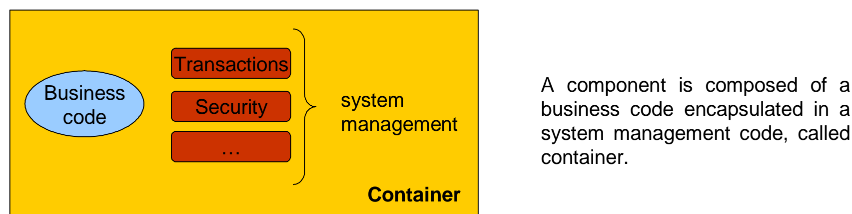

Figure 1. Component containers

- *Component architectures*. Component architectures are used not only in the case of applications but also in the case of replication protocols. By componentising replication treatments we allow to easily produce new protocols and to reuse existing protocols' parts. We base this component-based organisation on the principle that replication and consistency management is itself an application. The only difference is that protocol components treat exclusively replication and consistency issues and in consequence have specific interactions. The model is presented in detail in the next section while its performances are discussed further in this paper.

4 Component-Based Replication: the Model

As defined in the first section of this paper, our work has three main objectives: reuse of applications' business code, reuse of protocols' management code and non intrusion. In order to respond to these objectives we have defined three models : one for applications, one for replication protocols and one for the composition between applications and protocols. The application model is a minimal component-based model which responds to the reuse objective thanks to the inherent component encapsulation. It is presented in Section 4.1. The protocol model is a component-based model obtained through specialisation of the application model. It is described in Section 4.2. Finally, the composition model defines the relations to establish between the architectures of applications and protocols. It is considered in Section 4.3.

4.1 The Application Model

The application component model we propose is inspired of the existing EJB and CCM standards. It is simplified in order to allows us to focus only on issues relevant to replication and consistency issues.

A component is a unit of reuse and of composition. The component type is characterized by a set of interfaces which can be required or provided. A component's required interfaces define the services it uses while its provided interfaces define the services it implements.

Applications are composed of component instances which are assembled by interconnecting their provided and required interfaces. Depending on the connected interfaces, component instances can communicate in a synchronous or in an asynchronous manner. The architecture of the application is expressed in terms of components, interfaces and component interconnections.

To illustrate the model, let us consider the well known example of the "Hello World" application (Figure 2.). It is composed of two component instances: the client `C` and the server `S`. The client is an instance of the component type `Client` which requires the `Hello_itf` interface. The server is an instance of the `Server` component type which provides the `Hello_itf` interface. As the required interface of the client is provided by the server, the two instances can be connected. Thus, when the client calls the `print` method, it will call the server which will execute the corresponding treatment i.e. will print the string parameter on the screen.

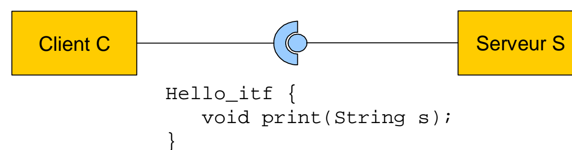

Figure 2. The "Hello World" Application

4.2 The Protocol Model

As mentioned before, protocol development is done using the component model of applications. However, we define the types of protocol components and characterize their interfaces and possible connections. These issues are detailed in the following sections.

Protocol components

We define four component types for the modelling of replication and consistency protocols. Namely, we define copy components, client components, service components and managers.

- *Copy components*. A copy component defines the unit of replication. It models a replicated component or a copy of a replicated component. To model a group of copies, for example, we will represent each copy of the group by a copy component and in the end we will have a group of copy components. The fact that the replication unit is modelled as a component allows to develop protocols that are application independent. A copy component can encapsulate different entities having different levels of granularity. It can be used to model files, directories, databases, database tables, etc.

- *Client components*. A client component models a component which interacts with a replicated component. It is introduced in the protocol model in order to reflect explicitly the fact that copies are modified by their clients and that these modifications may trigger copy synchronization.

- *Service components*. A service component provides some generic service treatment. It is independent of the semantics of a specific protocol and can be reused in different protocols. Examples of service components are timers, timestamp calculators, etc.

- *Managers*. A manager component manages the protocol architecture. It knows the component instances which participate in the protocol, as well as their connections. It may be in charge of some global reconfiguration treatments related to replication and consistency management. For example, it may receive an event announcing the need to create a new copy and react to this event by creating a copy and interconnecting it to other protocol components.

To illustrate the four types of protocol components, let us consider an example of a protocol for fault tolerance (Figure 3.). It models a service which has two copies in order to guarantee tolerance to two crashes. The service is modelled using three copy component instances: one of type `Primary` which represents the primary copy of the service and two of type `Secondary` which represent its secondary copies. The three copies are connected through the `Consistency_itf` interface which is used during copy synchronization.

A component which uses this replicated service is modelled using a client component of type `Client`. The `Client` component is connected to the `Primary` component using the `Service_itf` interface which defines the service. It is also connected to a service component of type `Timer` which is used to measure the timeout during the calls of the service. When the timeout expires, the client contacts the manager component of type `FT_Manager`. The manager component verifies whether the primary is still operational and if not, redirects the client to one of the secondary copies.

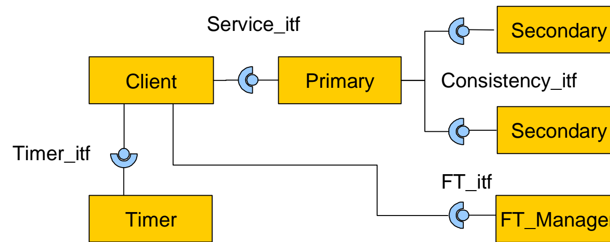

Figure 3. The Fault Tolerance Protocol

Protocol interfaces

As illustrated in the example of the fault tolerance protocol, protocol components are connected in the same way as application components i.e using their required and provided interfaces. However, their interfaces are specialized to replication and consistency management. We identify the following types of interfaces:

- *Reconfiguration interfaces*. Protocol components provide reconfiguration interfaces when they define treatments related to copy creation, destruction or synchronization. In fact, these interfaces define the events to which protocol components react by executing these treatments. For example, a copy creation may be triggered by an event which says that a copy has crashed. A copy destruction may be triggered when the network connections are judged to be sufficiently performant. Finally, a copy synchronization can be triggered after a copy modification.

  The interfaces are called *reconfiguration* interfaces as copy creation, destruction or synchronization are actions involving the creation and/or the suppression of protocol components and interconnections. Copy creation corresponds to a creation of a copy component and to its connection to other protocol components. Copy destruction has the reverse effect. Finally, copy synchronization may involve revisions of the group of copies and require its reconfiguration.

  In the example of the fault tolerance protocol, `FT_itf` is a reconfiguration interface. In fact, it is used by the client when there is a problem with the primary copy. It triggers the reconfiguration which consists in the replacement of the primary copy by a secondary one.

- *Synchronization interfaces*. Synchronization interfaces are typical to copy components. They are used during the phases of copy synchronization and define the informations exchanged between copies. They may take part of the reconfiguration interfaces as, as mentioned in the previous paragraph, synchronization may be related to reconfiguration. In the fault tolerance example, the synchronization interface used by the three service copies is `Consistency_itf`. In this simple example, it is used to periodically make snapshots of the primary copy to the secondary ones.

- *State management interfaces*. Copy synchronization cannot be done if copies cannot exchange information about their states. For example, in the fault tolerance protocol, the `Consistency_itf` interface needs to capture the state of the primary copy and to restore this

state on the secondary ones. In consequence, copy components provide state management interfaces which allow the capture and the restoring of their state. However, these state management interfaces are not implemented by the copy components but by the business components they model. In fact, the state management interfaces are provided after the integration of a protocol in an application by the business components to be replicated.

- *Service interfaces*. Service interfaces are provided by service components. They are required by the protocol components which use services implemented by service components. In the fault tolerance example, the `Timer_itf` interface is a service interface of the `Timer` service component. It is required by the client which uses the `Timer` in order to measure timeouts.

4.3 The Composition Model

The application model produces component-based applications which are developed without considering the management aspects of replication and consistency. The protocol model produces protocols developed without considering application constraints. The composition model is charged to put applications and protocols together in order to produce applications which benefit from the replication management of protocols.

In our proposition we only consider the composition between *one* application and *one* protocol. The composition is defined by establishing relations between application's and protocol's entities. Namely, it establishes relations between components, as well as between interfaces.

- *Component relations*. These relations define the roles of the application's components from the protocol's point of view. First of all, they define which components are to be considered as copy components i.e they define which application components are to be replicated. Next, they define the client components as being the components which use services provided by the replicated components. The rest of the components are ignored as they do not affect replication and consistency management.

- *Interface relations*. Component relations are completed by interface relations which define the role of business interactions from the protocol's point of view. They define which business interactions are to be modified in order to integrate replication and consistency treatments. They define business interactions as being reconfiguration interactions, synchronization interactions or non relevant interactions. In the case of reconfiguration or synchronization interactions, the business interactions are to be intercepted and the protocol's treatments are to be executed. In the case of non relevant interactions, the business interactions are not modified.

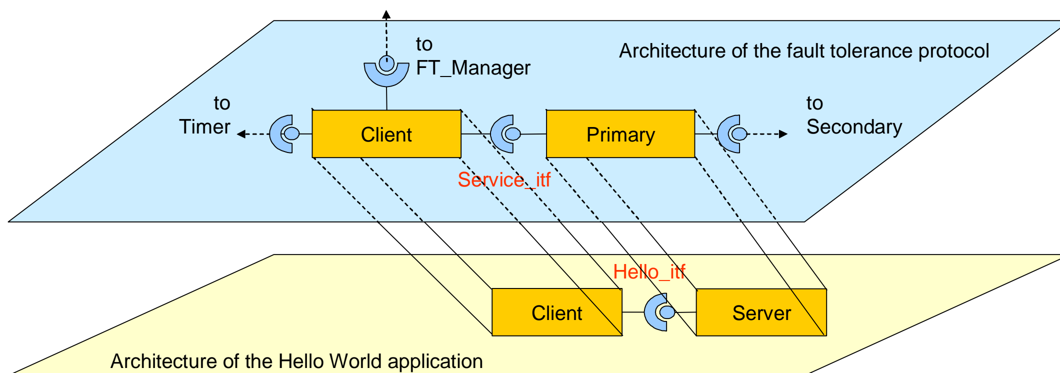

Figure 4. Composition between the "Hello World" application and the fault tolerance protocol

To illustrate the model, let us consider the composition between the "Hello World" application and the fault tolerance protocol (Figure 4.). The component relations to establish are the following. The "Hello World" `Server` component is modelled by the `Primary` protocol component. Thus, the "Hello World" server is made fault tolerant and in consequence will be replicated twice in order to create two secondary copies. As the "Hello World" client is using the server's services, it is modelled by the

`Client` protocol component. As far as the interface relations are concerned, they define the `Hello_itf` interface as being equivalent to the `Service_itf` interface. Thus, for each `print` call, the client will use a `Timer` in order to measure the time needed to complete this call. If the timeout is exceeded, the client will notify the `FT_Manager` component.

5 Component-Based Replication: the Experience

In order to evaluate the proposed model, we have implemented a Java prototype, called FAR[1]. In it, we have provided classes for application and protocol construction., as well as tools allowing the declarative definition of component types, the automatic generation of partial component implementations and tools for application and protocol deployment. Applications and protocols are composed during deployment and pass through the phases of establishment of component and interface relations and of integration of the protocol components in the application's architecture. In fact, the integration produces a flat architecture where the separate levels corresponding to the application and the protocol treatments are lost. Due to space limitations we will not describe in detail this implementation but will only report on our experiments with it.

We have experimented with several applications and with several protocols. The applications include an agenda application, a profile management application and a banking application. The protocols include the fault tolerance protocol, a disconnection management protocol and an entry consistency caching protocol. We have evaluated our proposition and our implementation according to the objectives of configuration and of performance.

- *Configuration objectives*. With FAR, we have validated the objective of business code reuse as we have succeeded to configure one application with different protocols. We have also validated the objective of reuse of protocol management code as we have integrated one protocol in different applications. As far as the non intrusion objective, it is partially achieved. In fact, we can only compose applications and protocols programmed according to the defined models. Moreover, applications cannot benefit from protocol management if they do not provide state manipulation interfaces for the components to be replicated.

- *Performance*. FAR has been implemented without any efficiency objectives. As a result, the overall performance of applications programmed according to our model is slower and consumes more memory than a simple Java application. For example, if a Java RMI call takes about 215 µs, the FAR distant call takes about 830 µs i.e is 4 times slower. If a simple Java object takes about 7 bytes, the equivalent FAR component takes 700 bytes. As a conclusion, for the moment FAR is useful only for prototype usages i.e it can be used in order to compare different application configurations and their overall behaviour after a protocol integration. However, FAR performances can be improved through use of efficient component implementation techniques and through optimization of indirection call chains. Our research team SARDES has gained some positive experience with efficient component implementation in the Fractal project [4] whose goal is to provide a flexible model for administration. As for optimization, SARDES has been working on the promising technique of code injection [8].

6 Conclusions and Perspectives

In this paper we have presented our proposition for component-based replication management. Our model is based on the inherent aspect separation between business and system management code in component platforms, as well as on component architecturing of replication protocols. We have proposed a component-based model for replication and consistency management which allows to program protocols as standard component applications. Thus, protocols are programmed in an application independent manner and can benefit from all the advantages of reuse and maintenance offered by

1. FAR = Framework for Adaptable Replication

component architectures. We have reported on our results with the model's implementation which show that the configuration objectives, namely the possibility to configure an application with different protocols and the possibility to use a protocol in different application contexts, are achieved. As far as the performance issue is concerned, our work can benefit from existing experience on techniques for efficient component implementation and for optimization using code injection.

The problem of replication configuration in the component domain is part of the larger problem of system management (or quality) configuration of component systems which concerns not only replication but also transactions, security, persistence, etc. We have proposed a model which considers replication configuration alone and does not treat relations to other system services' management. This is contrary to approaches such as those of CCM or EJB where other system services (transactions, security and persistence) are considered but not replication. In conclusion, this work needs to be completed by further analysis of the way the proposed replication model and the existing system management models can cooperate.

References

[1] BEA Systems - BEA WebLogic Server. http://www.bea.com/products/weblogic/server

[2] CARTER J.B. Design of the Munin Distributed Shared Memory System. The Journal of Parallel and Distributed Computing, Vol. 11, No. 6, 1995, pp. 219-227.

[3] Enterprise Java CORBA Component Model. http://www.cpi.com/ejccm/

[4] Fractal Home Page. http://www.objectweb.org/fractal/index.html

[5] GARBINATO B., GUERRAOUI R. Flexible Protocol Composition in BAST. In : The 18th International Conference on Distributed Computing Systems (ICDCS), May, 1998, Holland.

[6] GONCALVES T., SILVA A.R. Passive Replicator: A Design Pattern for Object Replication. In : 2nd European Conference on Pattern Languages of Programming. pp. 165-178. 1997.

[7] GUERRAOUI R., GARBINATO B., MAZOUNI K. GARF: A Tool for Programming Reliable Distributed Applications. IEEE Concurrency, Vol. 5, No. 4, 1997, pp. 32-39.

[8] HAGIMONT D. ET DE PALMA N. Removing indirection objects for non-functional properties. In : Proc. of PDPTA 2002, International Conference on Parallel and Distributed Processing Techniques and Applications, Las Vegas (USA), June 2002.

[9] HAMILTON G., MITCHELL G.J., POWELL M.L. Subcontract: A Flexible Base for Distributed Programming. In : ACM Symposium on Operating Systems Principles. 1993. pp. 69-79.

[10] iCMG: Software for CORBA Components. http://www.icmgworld.com/

[11] KICZALES G., LAMPING J., MENDHEKAR A. et al. Aspect-Oriented Programming. In : European Conference on Object-Oriented Programming (ECOOP), LNCS 1241. Springer, 1997

[12] OMG. CORBA & CORBA Component Model. OMG. 2002.

[13] Oracle9iAS Containers for J2EE. http://otn.oracle.com/tech/java/oc4j/content.html

[14] STEEN M., HOMBURG P., TANENBAUM A.S. Globe: A Wide-Area Distributed System. IEEE Concurrency. 1999. pp. 70-78.

[15] SUN MICROSYSTEMS. Entreprise Java Beans Specification 2.0. Sun Mycrosystems. 2001.

[16] YU H., VAHDAT A. Design and Evaluation of a Conit-based Continuous Consistency Model for Replicated Services. ACM Transactions on Computer Systems (TOCS). 2002.