Non-functional replication management in the CORBA Component Model

Vania Marangozova, Daniel Hagimont*

INRIA Rhône-Alpes, 655 avenue de l'Europe, Montbonnot 38334 St Ismier cedex, France
{Vania.Marangozova, Daniel.Hagimont}@inrialpes.fr

1 Introduction

Component-based programming is a promising approach to distributed application development [1]. It encourages software reuse and promotes the separation (as in aspect-oriented programming[5]) between the components' business implementations and the code managing the used system services. One system service of particular importance to the distributed computing domain is replication.

Managing replication as a separate (non-functional) aspect in a component-based environment is a way to provide a generic replication solution which can be applied to a wide range of applications. It allows the association of different replication/consistency protocols with components according to their contexts of (re)use.

If many projects have addressed the issue of managing replication in contemporary middleware (and more specifically in CORBA [3][6][9]), providing configurable replication management in a component-based environment is still an open issue.

This paper presents our approach to component replication, based on the CORBA Component Model (CCM [8]). We show that replication can be managed as a configurable non-functional aspect in a component-based system.

The article is organized as follows. Section 2 presents our motivations and Section 3 provides a rapid overview of CCM. Our infrastructure for replication and consistency configuration, as well as its application to a specific scenario, are presented in Section 4. Sections 5 concludes this article.

2 Motivations

We investigate the integration of a replication service in a component-based infrastructure. We aim at providing a solution which allows the implementation of various replication scenarios for a given component-based application. We are particularly interested in scenarios in which replication is used for cache management, fault tolerance and mobile disconnection.

* This work is supported by the Réseau National des Technologies Logicielles (RNTL Arcad) and by FranceTelecom R&D.

Replication management includes two major points: replication and consistency management. Replication involves the choice and the mechanisms for creating and placing copies on different network nodes. Consistency is concerned with the relations established between these copies. In consequence, in order to support various replication scenarios, the target infrastructure should provide adequate solutions to the following issues:

- Replication configuration (What, when & where?). The replication service should allow deciding *what* entities should be replicable. Moreover, such a solution should allow to decide of the most appropriate moment for replication (*when*) and to control optimal copy placement (*where*).
- Consistency configuration (How?). There is a need for mechanisms enabling consistency management policy configuration depending on the context of replication use (caching, availability during disconnection, etc.).

Therefore, replication and consistency management should be configurable.

3 The CCM platform

CCM [8] is a component model specification from OMG. The experiments reported in this article are based on the OpenCCM implementation [7] of this specification. We overview the CCM features that we used.

The abstract model is concerned with component descriptions. The defined descriptions are richer than standard IDL declarations as they consider both the interfaces used and provided by a component. This is made possible thanks to an IDL extension which introduces the notion of *port*. Ports define connection points (used and provided) for a component. Ports have a type which is an interface. They are used at runtime by clients for business method invocations and during deployment for component connections configuration. Given that the model imposes the static definition of all components' ports, the deployment phase makes all connections explicit.

The deployment model defines the deployment process which involves component implementations' installation (archives defined by the packaging model), component instances' creation, component configuration and port (component) interconnection, and application launching. In OpenCCM, application deployment is done by a deployment program. Therefore, the architecture of the application, defined as a set of components with interconnected ports, is described in the deployment program.

4 Replication management in OpenCCM

The design choices allowing to respond to the needs of an adaptable replication management infrastructure are first discussed. Then, we present an experiment in which we apply the principles of the proposed infrastructure to two replication/consistency scenarios for an simple agenda application.

4.1 Principle

As we have seen in section 2, replication management includes copy creation and placement control, as well as copy consistency management.

Given that copy creation and placement modify the global architecture of an application and that the application architecture is defined in the deployment model, replication configuration can be naturally specified in the deployment model. Replication configuration in the deployment model is described in section 4.1.1.

In our approach, we suppose that consistency between copies can be managed using specialized treatments executed before and/or after normal (business) method invocations. Consistency management implementation relies therefore on invocation interception. This interception is done at the interface level, separately from the component implementation, and is closely related to the component definitions given in the abstract model. Consistency management configuration is described in section 4.1.2.

4.1.1 Replication configuration

By defining what and where components are to be deployed as well as how these components are to be interconnected, the CCM deployment model describes the initial architecture of an application. Even if to date, CCM does not consider reconfiguration (the when aspect), the deployment model is the right place where reconfiguration actions should be defined.

Since replication configuration consists in creating and interconnecting additional entities in the application's architecture, replication configuration can be naturally specified in the deployment model. In fact, a deployment model including replication will have to specify the set of replicable components (what), define the most appropriate moment for replication (when) and the best copy placement (where). The definition of a dynamic copy creation is to be part of the future reconfiguration specification features of the deployment model.

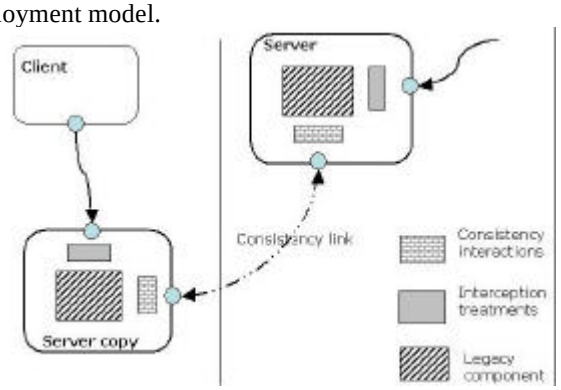

Fig. 1. A simple replication scheme

Replication configuration in OpenCCM is added in the deployment programs. As these programs control component instance creation and interconnection, they are

also given the responsibility of creating component replicas and connecting them to other components. In a cache management system for instance (**Fig. 1**), the deployment program has to connect a client component to a local copy of a server component in replacement of the remote one. As the copies have to be kept consistent, there is a need for connections between the copies. In fact, following a specific consistency protocol, a deployment program has to establish consistency links between component replicas (examples of consistency links are described later).

4.1.2 Configuration of consistency management

Consistency management in a replicated object system generally relies on interception objects triggering consistency actions upon copy invocations. The use of interception mechanisms in a component-based system allows the integration of the consistency management aspect without modification of components' functional code. The consistency actions take the form of pre and post treatments for the business method invocations which continue to be delegated to the initial component implementations.

Consistency protocols define consistency relations between copies and provide treatments to maintain these relations valid. Logically, these treatments require the existence of copy interconnections to propagate consistency actions. These connections are the consistency links mentioned in the previous section and settled during deployment. In the example of a cache management system, consistency actions may consist in the invalidation of a given copy and the acquisition of a fresh one which are implemented using consistency links.

Most consistency protocols require access to components' internal data. In the example of a cache management system, component's data need to be copied upon component's caching or update. The access to components' internal state may be based on global component capture/restoration or on application-specific selector/mutator functions. In our prototype, we preserve the component encapsulation principle by leaving the responsibility of implementing component state access primitives to the component developer. Consistency protocol implementations can thus rely on these primitives and ignore component implementation details.

The previously described interception objects and consistency links are the basis of consistency management. Their implementations depend of course on the specific consistency protocol chosen for a given application.

4.1.3 Implementation

We have implemented such a support for replication and consistency management on top of the OpenCCM middleware.

For consistency management, we implemented our own interception mechanism, but a similar feature should be soon integrated into OpenCCM, in the form of adaptable component containers. Replication configuration only relies on adaptations of deployment programs. Overall, replication and configuration can be managed as an adaptation of the application, relying on CCM features without requiring any extension to OpenCCM.

4.2 Experience

In our experiment, we have used a simple agenda application. In this application, users can connect to an agenda server and register, edit or remove rendezvous from their planning. The application is composed of two components: a client which includes a graphical interface and a server which includes the agenda's persistent data.

We have applied the above infrastructure principle to two replication scenarios. The first one implements a simple disconnection scenario while the second one implements a caching system.

4.2.1 Disconnection protocol

In the case of the disconnection protocol, we proceeded as follows. The agenda's components' implementations remain the same apart minor modifications in the Server in order to make it `Serializable` and to implement a default state management procedure.

The consistency implementation in this scenario distinguishes between master and slave server copies. A slave server is a disconnected copy of a master server. When a disconnection process is launched, the slave server is created on the machine getting disconnected and initialized with the state of the master. The client is connected to this slave server. At reconnection, the possibly diverged slave and master states are reconciled, and the client is reconnected to the master server. Reconciliation is based on a simple redo protocol using a log of disconnected operations (managed in the slave server).

4.2.2 Caching protocol

We have also experimented with a caching system for the agenda application. This caching system allows to deploy copies of the server component on several client machines and to keep consistent portions of the agenda database. We have implemented a version of the entry consistency protocol [2], which follows a multiple-readers/single-writer protocol.

The implementation of the consistency protocol is very close to the one implemented in the Javanaise system [4]. Each method of the agenda server component is associated with a component locking policy, depending on the method's nature (read/write) The interception treatment ensures that a consistent copy is cached before forwarding the invocation to it.

The deployment program specifies the sites where caching should be applied. A master site stores the persistent version of the server component and a client may address either the remote master copy or a local replica.

The consistency links between replicas implement an interface which defines operations for fetching (in read or write mode) an up-to-date component copy and for copy invalidation. There are actually two consistency links between a client and the server: one is used by the client in order to fetch copies (in read or write mode), the other is used by the server in order to invalidate copies and locks held by the clients.

6 Conclusion and future work

We have investigated the integration of replication and consistency management in a component-based platform. We have shown that it is possible to manage replication as an adaptable non functional property in a component-based system.

We have identified two places where the non functional integration of replication management takes place. The first place is at the level of the components' interfaces. We use interception objects in order to capture invocation events and to trigger consistency actions. These interception objects could be replaced by adaptation mechanisms at the container level when they are available (many research groups are working on adaptable containers, both in the EJB and CCM environments). The second integration place is the deployment model. Given that the deployment model is the place where an application's architecture is described and since replication impacts this architecture, it is logical to describe the replication aspect configuration in the deployment model.

As most of the configuration is done by hand, an immediate perspective of this work is to provide the tools which would assist the definition of replication policies.

References

[1] R. Balter, L. Bellissard, F. Boyer, M. Riveill and J.Y. Vion-Dury, ``Architecturing and Configuring Distributed Applications with Olan", *Proc. IFIP Int. Conf. on Distributed Systems Platforms and Open Distributed Processing (Middleware'98)*, The Lake District, 15-18 September 1998

[2] N. Bershad, Matthew J. Zekauskas, and Wayne A. Sawdon. The Midway Distributed Shared Memory System. In Proceedings of the 38th IEEE Computer Society International Conference, pages 528--537. IEEE, February 1993.

[3] G. Chockler, D. Dolev, R. Friedman, and R. Vitenberg. Implementing a Caching Service for Distributed CORBA Objects. In Proceedings of Middleware '00,1-23, April 2000

[4] D. Hagimont, F. Boyer. "A Configurable RMI Mechanism for Sharing Distributed Java Objects". IEEE Internet Computing, Vol. 5, 1, pp. 36-44, Jan.-Feb. 2001

[5] G. Kiczales, E. Hilsdale, J. Hugonin, M. Kersten, J. Palm, and W. G. Griswold. An Overview of AspectJ. In J. L. Knudsen, editor, ECOOP 2001, Object-Oriented Programming, volume 2072 of LNCS. Springer-Verlag, June 2001.

[6] R. Kordale and M. Ahamad. Object caching in a CORBA compliant system. USENIX Computing Systems, 9(4):377404, Fall 1996

[7] The OpenCCM Platform, http://corbaweb.lifl.fr/OpenCCM/

[8] Corba Components – Volume I, OMG TC Document orbos/99-07-01.

[9] Chris Smith, Fault Tolerant CORBA (Fault tolerance joint initial submission by Ericsson, IONA, and Nortel supported by Alcatel), ftp://ftp.omg.org/pub/docs/orbos/98-10-10, October 20, 1998.