

Position paper: Separation of concerns due to extensible container support

Ansgar Radermacher, Sylvain Robert,
Sébastien Gérard, François Terrier

CEA-LIST
CEA Saclay
91191 Gif sur Yvette Cedex, France
{Ansgar.Radermacher, Sylvain.Robert,
Sebastien.Gerard, Francois.Terrier}@cea.fr

Vincent Seignole, Virginie Watine

Thales
Thales Communications
91300 Massy, France

{vincent.seignole, virginie.watine}@fr.thalesgroup.com

Keywords

CCM, CORBA, ADL, connectors.

1. INTRODUCTION

Framework-based components approaches (e.g. CCM, EJB) have until not achieved a breakthrough in the real-time and embedded community. Yet, the interest in component based paradigms is increasing, along with the need to make systems less dedicated, more flexible, and follow standardized software platforms.

The work described here has been done in the context of the European projects COMPARE and MERCED¹ which focus on the development of a flexible component technology for complex embedded real-time systems. A promising approach is the container/component pattern employed by the CORBA Component Model (CCM) [2]. The component is embedded into the *container* through which all interactions with the external world are managed. Some non-functional concerns, such as transactions and persistency are already supported.

However, the *standardized CCM container is not flexible enough*. Transactions are useful for enterprise applications, but we rarely need these in embedded systems which require different services. An example is a configurable synchronization policy enforced by the container, but there is no possibility to add such a service in a consistent way.

Since CCM supports only synchronous method invocations and a specific form of event delivery, another motivation for more adaptability is the wish to use additional interaction mechanisms, for instance an event delivery mechanism provided by the target RTOS. In order to do this, we could configure a CORBA ORB, e.g. by means of pluggable transport protocols. We consider this inadequate, since an ORB has a higher overhead compared to a native communication layer. We don't seek a fixed extension of the set of supported interactions: in the embedded and RT domain, we often need to add interaction mechanisms that exactly fit

domain and target – we require a general procedure how to add specific interaction mechanisms.

We motivated the need for two kinds of adaptability in a container: the flexible management of services and interaction mechanisms. In the next two sections, we deal with these two aspects and present their realization in CCM.

2. Open container

This section deals with an *open container*, i.e. a container tailored by means of plug-ins that support the handling of non functional aspects. We call these plug-ins *container services*. The following considerations are based on the lightweight variant of CCM [1].

The general approach is the *separation of concerns principle*: we keep real-time and quality-of-service (QoS) related code as far as possible out of the component code. Instead, we add the real-time aspects by means of *declarations* that are part of the container configuration. This enhances the reusability: the remaining “business code” is free of specific aspects of a certain target configuration and is therefore reusable.

2.1 Container Services

A major feature of the open container is its flexibility. Instead of providing a predetermined set of container services, it is possible to register a new service with a container using a plug-in architecture.

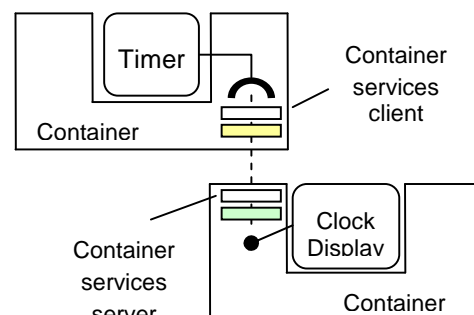


Figure 1: Container Services

As shown in Figure 1, one or more container services can *intercept* the call through a receptacle on the client side. In

¹ <http://www.ist-compare.org> <http://www.itea-merced.org>

the same way, incoming calls can be intercepted at the service side. A container service can also invoke methods on a component facet without being triggered by an incoming call. This is for instance useful for time triggered activation of a certain facet.

The contrary is possible, but discouraged: a component implementation (the executor) might invoke an interface provided by the container service. There are two problems with the latter: the component code depends on a specific container service and it needs additional mechanisms to retrieve a suitable reference to that service.

In the context of the QoS-for-CCM submission [4], we cooperate with Fraunhofer Fokus to standardize the support of non-functional constraints via interception.

3. The CCM connector

In this section, we motivate the use connectors in CCM and roughly sketch their implementation strategy, for more details see [5].

In its native field (Architecture Description Languages [3]), the *connector* is a clearly distinguished entity *dedicated to interaction management*. It enables a *more flexible middleware*, since it supports customized interaction protocols.

Interaction patterns are often bound to a given application domain (e.g. pipes and filters in communication SW). *The way interactions are dealt with is part of the domain expertise*. Thus, building reusable interaction media (connectors) enables to gather and reuse (i.e. capitalize) SW practitioners' knowledge on interactions management.

3.1 Component Model Extensions

3.1.1 Port Responsibilities

Due to the introduction of connectors, ports only define the interface in terms of operations and are not directly responsible for the interaction mechanism that is chosen. The semantics of uses and provides is that of a local method call and a method callback respectively. The implementation of the invoked method in the connector fragment could execute for instance a synchronous CORBA call or emit an event via a specific RTOS primitive.

3.1.2 Templates and Derived Interfaces

The ports of an event bus connector refer naturally to a generic interface for publishing an event to the bus (and another for delivering it to a subscriber). However, the use of generic interfaces is not useful for many other connectors, for instance for asynchronous invocations. It has to be possible to *instantiate* the connector and its ports with a certain interface. We use the terms *fixed* and *adaptive connector*, respectively.

3.1.3 Access through Connector Fragments

In a deployment view the connector is split into two *fragments* that are co-located with the components using them.

Each fragment has to be bound to the component using it. Please note that the *connection between a component and a connector fragment is always a local method invocation*. In order to setup this connection, we need to obtain references of component and connector ports first.

4. Related Work

Wang et al. recognized the possibility that a CCM container can manage QoS properties [6]. The work is done in the context of the CCM implementation CIAO (DOC group). Galik and Bures from the Charles University provide a flexible connector framework [7] with interception possibilities, yet their framework is much more general and not focused on CCM (yet, they use OMG's D+C specification). Their implementation is written in Java whereas we target C++ and embedded C++.

5. Summary

We extended the CORBA container model with regard to two aspects:

1. The possibility to add container services that can intercept the call chain to the component at various places. Thus, we reach a flexible, open container.
2. Adding separately reusable interaction patterns by means of connectors.

Both aspects are configured based on a declarative application description and not in the code. This separation of concerns makes specific configuration choices explicit instead of hiding it into the code and facilitates the reuse of the component in different contexts.

6. REFERENCES

- [1] Lightweight CORBA Component Model – OMG draft adopted specification, Object Management Group, 2003.
- [2] CORBA Components, version 3.0, Object Management Group, 2002.
- [3] A Classification and Comparison Framework for Software Architecture Description Languages, N. Medvidovic, R. N. Taylor, IEEE transactions on software engineering, vol. 26, n. 1, 2000.
- [4] Fraunhofer FOKUS, revised submission QoS for CCM, OMG document mars/05-08-07, 2005
- [5] Enhancing Interaction Support in the CORBA Component Model. S. Robert et al. In IESS Symposium, Manaus, Brazil, August 2005
- [6] Towards an Adaptive and Reflective Middleware Framework for QoS-enabled CORBA Component Model Applications, N. Wang et al., 2000
- [7] Generating Connectors for Heterogeneous Deployment, Ondrej Galik and Tomas Bures, Proceedings of SEM 2005, Lisbon Portugal, September 2005