A case study of Middleware to Middleware: MOM and ORB interoperability

Jérôme Hugues, Fabrice Kordon, Laurent Pautet, and Thomas Quinot

{hugues, pautet, quinot}@enst.fr
École Nationale Supérieure des Télécommunications
CS & Networks Department
46, rue Barrault
F-75634 Paris CEDEX 13, France

fabrice.kordon@lip6.fr
Laboratoire d'Informatique de Paris 6/SRC
Université Pierre & Marie Curie
4, place Jussieu
F-75252 Paris CEDEX 05, France

1 Schizophrenic middleware: a solution to *M2M*

Diversity in distributed applications leads to diversity in distribution models, and hence in middleware. However, large systems may need different types of middleware and interoperability between them, requiring *Middleware to Middleware* (M2M) capabilities [Bak01]. Such infrastructures supporting interoperability between different distribution models require a specific architecture; to be as independent as possible from the underlying distribution model, it should be generic. Generic middleware relies on the instantiation of a set of functionalities according to a distribution model to create a *personality*, (see Jonathan project [DHTS98]). However, this does not allow interoperability.

We have introduced the *schizophrenic middleware* concept [QPK01] as a solution to both genericity and interoperability problems. Schizophrenic middleware extends generic middleware to simultaneously support multiple interacting personalities within the same middleware instance. Interoperability is provided through dynamic gateways between these interacting personalities. We have implemented a free software schizophrenic middleware, PolyORB[1], to assess the feasibility of this concept. We have also developed several personalities: CORBA, SOAP and the Ada 95 Distributed System Annex (DSA). Thus we experienced inter-ORB interoperability. As a demonstration of the schizophrenic middleware capabilities, we briefly present the PolyORB architecture and describe the implementation of a Message Oriented Middleware on top of it; we then assess MOM and ORB interoperability capabilities provided by this very specific design.

2 Overview of PolyORB

PolyORB decouples middleware functions into *application-level* personalities and *protocol-level* personalities connected to the *neutral core*, on which all personalities rely.

[1] available at *http://libre.act-europe.fr*

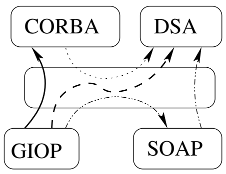

Fig. 1. inter-ORB interoperability

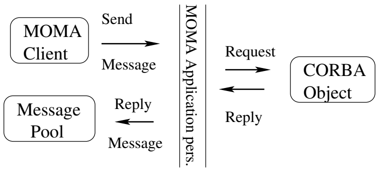

Fig. 2. MOM/ORB interoperability

Application personalities constitute the adaptation layer between application components and middleware. They provide APIs to register application entities with PolyORB's core, and interoperate with the core to allow the exchange of requests with remote entities (like CORBA DSI and POA mechanisms or JMS APIs).

Protocol personalities handle the mapping of requests (representing interactions between application entities) onto messages exchanged through a communication network, according to a specific protocol. The requests are received either from application entities (through an application personality and middleware core), another protocol personality or a node in the middleware instance.

The Neutral Core acts as an adaptation layer between application and protocol personalities. It manages execution resources and provides necessary abstractions to transparently pass requests between protocol and/or application personalities in a neutral way. It is completely independent from both application and protocol personalities, enabling the selection of any combination of them.

Figure 1 shows request paths for various PolyORB configurations, involving CORBA, DSA, GIOP and SOAP personalities. By construction, the core layer acts as dynamic gateway to allow application and/or protocol personalities to mutually use their services. This naturally leads to interoperability: entities registered to an application personality are available to any client using a middleware for which the corresponding protocol personality exists; the combination of two protocol personalities allows the creation of dynamic request proxy.

3 Bulding a MOM on top of PolyORB

Message Oriented Middleware are widely used in messages passing based distributed applications. Typical MOM characteristics are now well-identified: transport protocols, routing policies or advanced services such as naming.

We wanted to show that MOM can be built on top of PolyORB which was first experimented with object oriented middlewares, and that thanks to our design it was able to interoperate with more different distribution models by taking advantage of the schizophrenic architecture. Our MOM, MOMA, is a thick Ada 95 implementation of the Java Message Service API [SUN99].

Mapping MOMA functionalities to PolyORB: In a typical JMS MOM, the API provides primitives to clients to allow them to interact with message pools through requests (e.g. to post and receive messages). In between, a provider supports distribution functions and issues raised by client/server communications, as well as message pools. We can then define PolyORB's personalities that handle these two aspects of MOM: message pools and client API are part of an *application personality*, whereas a *protocol personality* addresses transport mechanisms. The MOMA provider is then the combination of these two personalities and the core middleware, providing services to mask distribution.

Application personality: Interactions between a client and a message pool are similar to method calls on objects in a distributed application. In this respect, we can define an application personality implementing message pools mechanisms. MOMA clients will interact with these entities to send and receive messages. We designed a minimal application personality to implement pools, a simple but effective architecture directly built on top of PolyORB's core that provides the minimal set of functionalities required to support MOM primitives.

Protocol personality: We focus ourself on application level interoperability. Hence, the protocol personality is minimal and provides simple mechanisms to transport requests between a MOM client and a message pool. We then reuse previously implemented protocol personalities (GIOP and SOAP) that were sufficient for our purpose.

4 Assessment of Interoperability

We define interoperability between distribution models as the capability of a middleware to allow entities defined within a given distribution model to interoperate with remote entities from a different distribution model. Such a definition is natural when the platform only involves ORB-like distribution model. We want to extend this notion to architectures mixing MOM and ORB distribution models. We studied application level interoperability and thus restrained our study to MOM/ORB interoperability.

PolyORB design makes MOMA objects available in the same middleware instance than CORBA or DSA objects, hence one entity can interact with other entities, whatever its application personality is. This covers interoperability: a CORBA object can call MOMA pools primitives and interact with a message pool, and the Request Message allows a MOMA client to make a method call on a CORBA object, as presented in figure 2

However, the MOMA client has to carefully format a message. This message, when sent to an 'ORB' destination is transparently transformed by the *MOMA application personality* into a request, which is then forwarded by the *neutral core layer* to the corresponding object. The reply sent by the ORB object is then intercepted by MOMA and bounced to a message pool set by the client. This scenario is a part of the applications that may use both MOM and ORB architectures. We can imagine the dual scenario where a CORBA request is mapped into a MOMA message and then sent to a dedicated pool for processing.

Through these examples, we can see how paradigms can be combined to alleviate some of the problems in the configuration and deployment of a distributed application and propose new features to a distribution model: ORB may benefit from MOM solutions for asynchronous request processing, group communication; MOM may have their request processed by ORB implementation objects. Hence, coupling MOM and ORB facilities in the same middleware architecture brings great flexibility to distributed application design, and provides interesting combination.

5 Conclusion and future work

We have extended PolyORB, our schizophrenic middleware, to support personalization into a Message Oriented Middleware. An initial case study has demonstrated that the canonical architecture of classical MOM fits perfectly in the schizophrenic architecture. Therefore, the design of the corresponding application and protocol personalities was quite natural. As an initial requirement, the application personality provides an Ada message passing interface modeled after JMS. PolyORB thus becomes the first free middleware providing a MOM solution for the Ada community.

PolyORB's schizophrenic design clearly alleviated most of the required work and let us concentrate on MOM API, also verified by CORBA and DSA personalities. For instance, code reuse is between 70% for CORBA personality up to 85% for MOMA.

Moreover, as a consequence of schizophrenic properties, PolyORB enables MOM and ORB applications to interoperate. We have successfully demonstrated this interoperability between CORBA, DSA and MOM applications and we are about to proceed to more general experiments. Moreover, this middleware is not dedicated to a specific protocol and can be configured for any classical protocols like GIOP, SOAP or raw sockets. We are currently extending PolyORB to provide some of the most frequent services available in classical MOM.

References

[Bak01] S. Baker. Middleware to middleware. In *Proceedings of the 3rd International Symposium on Distributed Objects and Applications (DOA'01)*, September 2001.

[DHTS98] B. Dumant, F. Horn, F. Dang Tran, and J-B. Stefani. Jonathan: an open distributed processing environment in java. In *Proceedings of the IFIP International Conference on Distribut ed Systems Platforms and Open Distributed Processing*, 1998.

[QPK01] T. Quinot, L. Pautet, and F. Kordon. Architecture for a reuseable object-oriented polymorphic middleware. In *Proceedings of PDPTA'2001*, Las Vegas, USA, June 2001.

[SUN99] SUN. Java Message Service (JMS), 1999.