

IEEE DISTRIBUTED SYSTEMS ONLINE 1541-4922 © 2005 Published by the IEEE Computer Society

Vol. 6, No. 3; March 2005

Department Editor: Olivier Marin, <http://www.cs.vu.nl/~omarin/>, Laboratoire d'Informatique de Paris 6, olivier.marin@lip6.fr

Toward Nex-Generation Middleware?

Fabrice Kordon, *Laboratoire d'Informatique de Paris 6*

Laurent Pautet, *Télécom Paris*



Department Editor: Olivier Marin

A large range of application domains, from real-time embedded systems to grid-computing applications, now require distribution. This trend implies definitions of new or tailored distribution mechanisms dedicated to specific applications and puts a strain on current middleware architectures and development.

Crisis and paradox in distributed-systems development

Let's define a distribution model as a set of mechanisms to handle distribution—for example, distributed object computing, Remote Procedure Call, and message passing. Middleware specifications such as CORBA or Java Message Service propose the API and protocols to support these models. (For example, CORBA supports distributed object computing, and Java Message Service supports message passing.) Middleware implementations provide the circuitry to build and deploy applications.

Because of the increasing number of application domains requiring distribution and the consequent increase in distributed-systems requirements, users might need to extend or restrict a distribution model. Besides, a specific application might require mechanisms such as object persistence, transactional request processing, or light-weight runtimes.

Middleware aims to unify and ease the development of an application on top of as general a distribution model as possible. But, in the middleware area, one size doesn't fit all. For example, most applications require only a subset of middleware services; developers should consider lighter, adapted implementations of a distribution model (see www.zeroc.com/iceVsCorba.html for a comparison between the Internet Communications Engine and CORBA).

Consequently, middleware requires a partial redesign to match the exact application requirements. We call this the *middleware crisis*.

Distributed applications are becoming increasingly complex, and the reuse of existing distributed components is necessary to reduce development costs. Components might be either legacy components or commercial off-the-shelf (COTS) components. Assembling these components regardless of their distribution models is difficult. Moreover, the multiplication of distribution models doesn't help because components might then come from heterogeneous middleware. Middleware intends to separate an application from variations in hardware and operating systems. This new interoperability problem coming from middleware itself is now a serious industrial issue. We call this the *middleware paradox*.

Next-generation middleware should be versatile enough to instantiate the exact required mechanisms of different distribution models. Middleware components that depend on a specific distribution model should be limited to application-level components or to protocol-level components. Many middleware components should remain unchanged from one instantiation to another.

Making middleware versatile

In our experience, four main properties characterize middleware versatility:

- ◆ *Configurability* lets you adapt an infrastructure to an application's real needs. A classical approach consists of defining an architecture in which loosely coupled components are separately configurable. The designer inserts properties into, or withdraws them from, the targeted middleware.¹
- ◆ *Genericity* aims to handle configurability at the distribution model level. This is

a technical challenge for middleware engineering because such platforms must adapt to a large variety of distribution models. A classical approach is to factor out components to override and to reuse and integrate into a generic architecture personalized for the distribution model.²

- ◆ *Interoperability* enables communication between components built on top of different and heterogeneous distribution models. Typically, solutions provide either static or dynamic point-to-point translations of entities from one distribution model to another. This introduces significant overhead; some deployments might impede application scalability.^{3,4}

- ◆ *Quality confidence*, as well as availability and dependability, is a nonfunctional requirement of an application. So, properties such as determinism, safety, liveness, and timeliness must be proved or verified during design at both the application and middleware levels. However, verification of middleware properties is difficult and is usually performed on a limited scale.⁵

Toward solving the middleware problem

Owing to the middleware paradox, middleware design faces a problem similar to that of compiler design or processor architecture in the 1980s.

Compiler theory faced a complexity issue owing to the multiplication of programming languages and processor architectures. To solve that problem, most compilers now come with a flexible architecture separating the compilation steps. A front end analyzes source code and interacts, using intermediate representations, with a back end that assembles machine code.

Processor design was facing growing instruction sets and thus overly complex implementation. To solve the problem, RISC (reduced-instruction-set computing) processors now come with canonical instruction sets. The saved silicon surface lets a chip include more memory or dedicated coprocessor units.

The middleware community must propose a solution similar to those we just described. An approach to handling middleware complexity should define a novel architecture based on a set of canonical middleware components. Figure 1 illustrates one possible architecture.

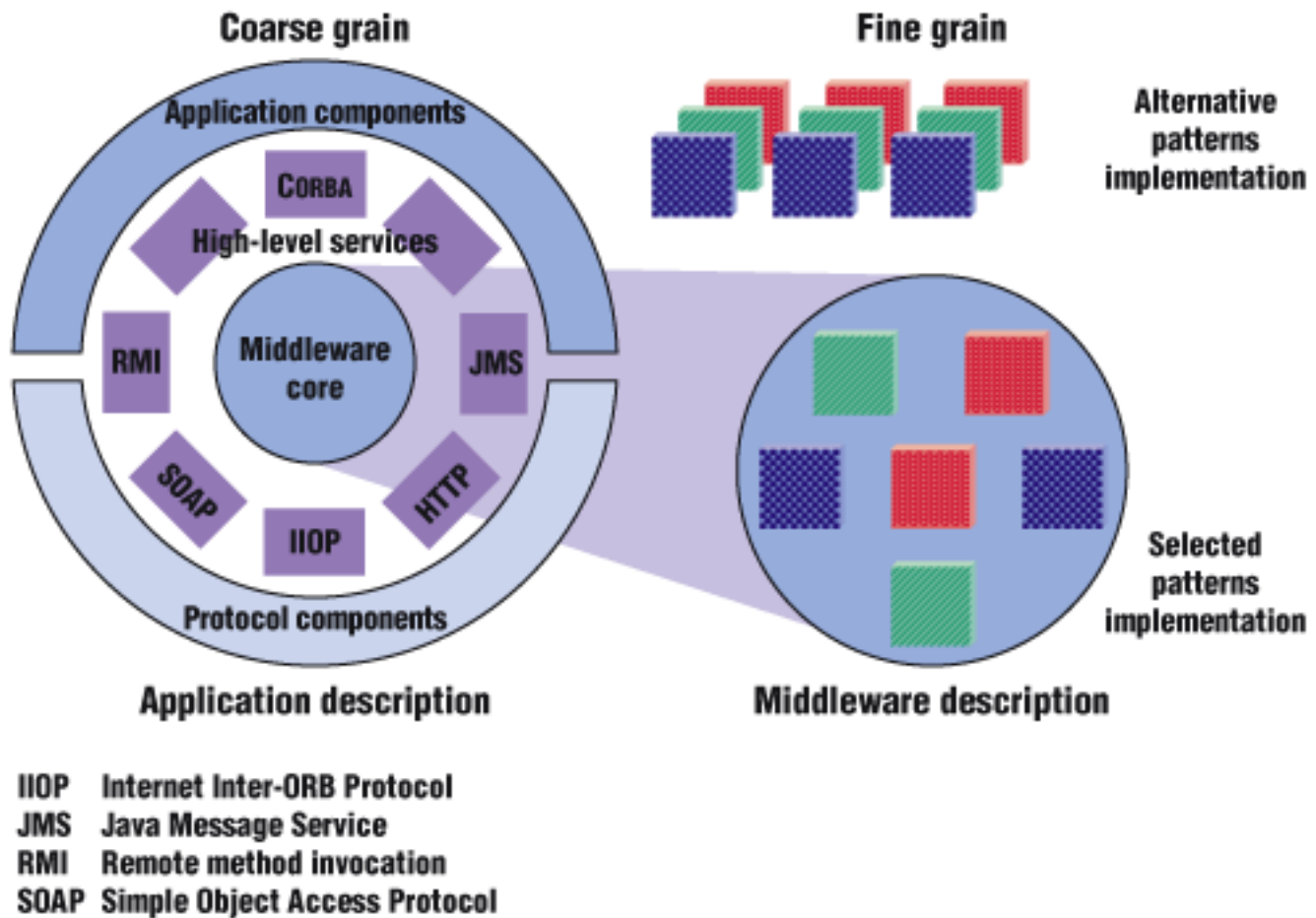


Figure 1. Overview of a middleware architecture.

Middleware offers two interfaces. The upper interface implements the distribution model and is used by the application components. The lower interface consists of communication management and enables interactions with remote nodes. So, the middleware could rely on a core set of fundamental components that handle more sophisticated components such as those implementing the application and protocol interfaces. This coarse-grain level supports both genericity and interoperability:

- ◆ To provide a general scheme to implement a distribution model, you can reuse, override, and extend components from the middleware core. As in some processors where an underlying kernel emulates a complex instruction set, the core acts as the basis for the complex components at the application and protocol levels. Moreover, porting the middleware core to another OS is easier.
- ◆ The application and protocol interfaces all rely on the same middleware core, which therefore can act as a gateway between these layers. For instance, intermediate representations provided by the middleware core can ease the transformation of remote method invocations into Java Message Service

exchanges. The application interfaces act like compiler front ends, while the protocol interfaces act like back ends. The middleware core corresponds to the intermediate language and its management procedures.

The middleware core is classically organized around a scheduler that manages all the middleware components. At this fine-grain level, configuration and quality confidence are easier to achieve:

- ◆ Each component can provide several implementations. Even the scheduler can be either monothreaded or multithreaded. Thus, given a set of constraints (such as the memory footprint), you can select appropriate implementations to build the required configuration.
- ◆ As with reduced processor instruction sets, the middleware core is easier to implement and test (the scheduler being the most complex part). So, providing a reliable core regarding a given set of properties is easier too. Because a reliable core is a prerequisite for reliable middleware, this is a positive step to increase middleware confidence.

Researchers around the world have been experimenting with all the principles we've presented here. One of the first experiments, Quarterware,⁶ defines a set of design patterns that, once specialized and assembled, implement CORBA , remote method invocation, or message passing interface distribution models. Artix (www.iona.com/products/artix) is a commercial product that intends to make interoperable numerous existing systems, representing multiple generations of architecture and technology. PolyORB (<http://polyorb.objectweb.org/>) is free software that tackles behavioral and architectural modeling issues. We think that such research is becoming one of the main trends in the middleware community.

References

1. D.C. Schmidt and C. Cleeland, "Applying Patterns to Develop Extensible ORB Middleware," *IEEE Communications Magazine*, vol. 16, no. 4, 1999, pp. 54–63.
2. B. Dumant et al., "Jonathan: An Open Distributed Processing Environment in Java," *Proc. Middleware '98: IFIP Int'l Conf. Distributed Systems Platforms and Open Distributed Processing*, Springer-Verlag, 1998, pp. 175–190.
3. F. Breg et al. "Java RMI Performance and Object Model Interoperability: Experiments with Java/HPC++ Distributed Components," *Concurrency: Practice and Experience*, vol. 10, nos. 11–13, 1998, pp. 941–955.
4. S. Baker, "A2A, B2B—Now We Need M2M (Middleware to Middleware)

Technology," *Proc. 3rd Int'l Symp. Distributed Objects and Applications (DOA 01)*, IEEE CS Press, 2001, p. 5.

5. J. Hugues et al., "On the Formal Verification of Middleware Behavioral Properties," to be published in *Proc. 9th Int'l Workshop Formal Methods for Industrial Critical Systems (FMICS)*, Elsevier, 2004.
6. A. Singhai, A. Sane, and R. Campbell, "Quarterware for Middleware," *Proc. 1998 Int'l Conf. Distributed Computing Systems (ICDCS 98)*, IEEE CS Press, 1998, pp. 192–201; <http://choices.cs.uiuc.edu/Papers/Theses/PhD.Singhai.Quarterware.html>.



Fabrice Kordon is a professor of computer science at the Université Pierre et Marie Curie and a member of LIP6 (Laboratory of Computer Sciences, Paris 6). He's also the head of the LIP6 Distributed and Cooperative Systems Department. Contact him at fabrice.kordon@lip6.fr.



Laurent Pautet is an associate professor of computer science in the Network and Computer Sciences Department at Télécom Paris. He's the original designer of three free middleware projects (Glade, AdaBroker, and PolyORB). Contact him at laurent.pautet@enst.fr.

Cite this article: Fabrice Kordon and Laurent Pautet, "Toward Next-Generation Middleware?" *IEEE Distributed Systems Online*, vol. 5, no. 1, 2005.