# Increasing MOM Flexibility with Portable Rule Bases

Message-oriented middleware (MOM) provides an effective integration mechanism for distributed systems, but it must change frequently to adapt to evolving business demands. Content-based routing (CBR) can increase the flexibility of MOM-based deployments. Although centralized CBR improves a messaging solution's maintainability, it limits scalability and robustness. This article proposes an alternative, decentralized approach to CBR that uses a portable rule base to maximize MOM-based deployments' maintainability, scalability, and robustness.

**Edward Curry**
*National University of Ireland, Galway*

Integration presents a common challenge for the software community. Whether connecting two simple applications or a collection of complex distributed systems, integration can be demanding. At its most basic level, integration involves connecting one system's inputs and outputs to another's. Faced with integrating multiple distributed systems, however, developers need to consider numerous additional concerns. When choosing an integration approach, they must ask themselves how the solution will perform within the operating environment. How manageable is the solution? What is the maintenance burden? Will the solution scale? Is it heterogeneous and flexible, with support for current and future legacy systems? These key factors determine integration's success.

Message-oriented middleware (MOM) is an effective and flexible mechanism for interconnecting systems. It simplifies the integration of multiple applications and enables the creation of flexible and adaptable deployments. MOM provides a diverse range of messaging capabilities, including the point-to-point and publish–subscribe messaging models, message filtering, transactional messaging, and once-and-once-only message delivery. In addition, MOM successfully reduces the interface and temporal coupling that synchronous Remote Procedure Call-based mechanisms experience, resulting in a highly cohesive, decoupled approach to connecting multiple systems.[1]

However, MOM is not a silver bullet for system integration — several challenges still exist. Application/business logic (ABL) drives a MOM's configuration. Because this form of logic frequently changes to meet current business demands, the MOM's configuration will also need to change. A MOM-based integration approach must accommodate these changes in a flexible manner. Here, I examine MOM's role within the integra-

 Published by the IEEE Computer Society 1089-7801/06/$20.00 © 2006 IEEE 

tion process and, in particular, how its configuration can introduce obstacles.

## Integration Challenges

Successfully integrating two systems requires that the developer reconcile or bridge both systems' semantics. In a large-scale heterogeneous integration effort, this is one of the greatest challenges. To achieve a successful integration, all participants must have a common conceptualization of the problem domain. David Orchard highlights some of these issues:[2]

> Imagine a Purchase Order system. A sender sends Purchase Orders to a receiver, who responds with successful completion of the order or failures. The receiver must understand all the nuances and details of the purchase order messages. Any interface or type change — such as changing the authentication structures, changing the timing of the authentication step, changing the purchase order messages, etc. — [will] require that the sender change. And that means a programmer must perform the change.

The issues go beyond simply bridging low-level interfaces and message formats to reconciling the system's ABL, which affects all areas of system development. Most prominently from an integration perspective, it directs the definition of messages: who should receive the information, and how and when.

The MOM domain uses several constructs — queues, topics, journals, destination hierarchies, and so on — for message exchange. The developer configures these constructs to meet the demands of the particular application domain. In many cases, the domain's ABL will heavily influence their configuration and will dictate how an application uses the MOM to exchange messages, creating a tight coupling to a specific destination configuration and coupling the application to the MOM infrastructure. This can influence the messaging solution in ways that affect its flexibility, maintainability, and scalability. To illustrate this form of infrastructure coupling and the challenges it creates, consider the following hypothetical scenario.

## Hypothetical Scenario

A diverse set of domains — including news and weather services, online auctions, marketplaces, enterprise application environments, and a wide range of information dissemination systems — use MOM. The following scenario from the enterprise application integration (EAI) domain illustrates the impediments and issues ABL and MOM can produce. Although the scenario is EAI-specific, the issues apply to a broad range of messaging scenarios. This example assumes the presence of a message standard such as ebXML, the Universal Business Language, or RosettaNet.

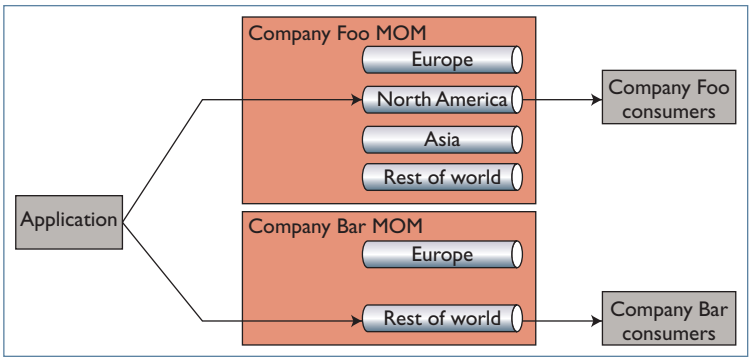


*Figure 1. Direct integration. The application connects directly to each company's message destinations.*

Consider the case of two multinational companies, Foo and Bar. Each company operates its own independent MOM. When designing its communications infrastructure, each company created destinations relevant to its own message consumers' needs and associated ABL. For simplicity's sake, this scenario limits the consumers' needs and the ABL's concerns to the regional divisions of messages. However, a company can use other concerns to route communications along several lines, including departmental divisions, business units, document types (invoice or purchase order, for example), message priorities, and so on. Each additional concern division increases the infrastructure's complexity and the effort that connecting an application requires.

### Direct Integration Example

In this scenario, the application on the left side of Figure 1 produces messages for the consumers on the right; the MOM in the center delivers these messages. For the application to communicate with either company's message consumers, it must place its messages in the relevant destination. Foo has four possible message destinations: Europe, North America, Asia, and a catchall destination for messages outside those regions. Bar has a simpler infrastructure, with only two destinations: Europe and all other regions.

The most straightforward method of integration available is to hardcode the application directly to both communication infrastructures, connecting directly to the destinations in each MOM. This solution's main advantages, as Figure 1 shows, are its

straightforward implementation and decentralized message delivery. Distributed message delivery will increase the overall deployment's scalability, but this approach has limitations.

### Direct Integration Limitations

Direct integration requires developers to "wire up" their applications to each MOM's messaging infrastructure. Although hardcoding two simple messaging infrastructures might be trivial, the process's complexity and cost grows geometrically as the messaging infrastructures' number and intricacy increase.

Hardcoding the application directly to the MOM infrastructure creates tight coupling. This reduces flexibility and manageability. At the same time, it increases the integration solution's maintenance burden, because any modification to the MOM infrastructure will require reciprocal alterations in both the consuming and producing applications. The challenge in this scenario is for the application to integrate with each company's infrastructure in an effective, flexible manner while minimizing the cost of changes to the messaging infrastructure's configuration.

## Configuration-Oriented Integration

To solve the problems that changes to ABL can cause for MOM-based deployments, we must examine state-of-the-art integration practices. Traditional integration involves programmatically connecting two systems. In configuration-oriented integration (COI), configuration files direct a middleware platform to provide the connection between two systems' inputs and outputs. COI is an important technique that can connect multiple systems and accommodate frequent ABL changes in a flexible manner. Service-oriented architectures (SOAs) use the technique to interconnect via enterprise service bus (ESB). Configuring the ESB messaging infrastructure allows services to interconnect within a SOA. The ESB's configuration captures the SOA's ABL messaging infrastructure requirements.

MOM-based deployments such as the hypothetical scenario have several COI options. One common solution captures the messaging infrastructure configuration for MOM-based deployments.

## Content-Based Routing

To overcome direct integration's shortcomings, a middleman such as content-based routing (CBR) can deliver messages between producers and consumers. CBR transports messages to consumers based on the messages' content. This is the messaging counterpart of COI, with routing rules serving as the equivalent of COI's configuration mechanism. The rule base captures the messaging infrastructure's ABL.

A messaging system can offer CBR services either at the MOM infrastructure level or at the application level via a message router within the messaging solution. Numerous commercial and academic MOMs,[3] including Tibco,[4] WebSphere MQ (formerly IBM MQSeries),[5] Oracle Advanced Queuing,[6] Siena,[7] Rebecca,[8] Hermes,[9] and Elvin,[10] offer infrastructure-level CBR.

A downside to infrastructure-level CBR is the lack of support from messaging standards. Each MOM provides proprietary CBR functionality with varying capabilities, from simple message filtering to advanced composite event detection.[9] The widely adopted Java Message Service (JMS)[11] provides limited support for consumer-side CBR by using message selectors to flag messages within destinations. However, this approach still requires consumers and producers to locate the relevant destination to exchange messages. Due to this lack of standardization, implementing an infrastructure-level CBR can lock a system into a proprietary MOM platform.

Another drawback of infrastructure-level CBR is the frequent lack of centralized rule bases. Many implementations have consumers define the routing rules via their subscriptions. This requires consuming applications to change when the infrastructure changes, replicating the high-cost maintenance from which direct integration suffers. To avoid proprietary lock-in and ensure the use of a centralized rule base, a message router can provide CBR services at the application level.

## Centralized Content-Based Routing

The content-based router is a fundamental design pattern that messaging solutions commonly use. The message router contains a rule base that stores routing instructions in order to direct messages to the relevant destinations based on their content. Variations on the pattern include the dynamic router and the more generic message router.[12]

### Centralized CBR Example

Figure 2 revisits the hypothetical scenario, this time using the centralized CBR pattern. Messages take the following steps:

1. The application produces the message and sends it to the router.
2. The router evaluates the message against its rule base to match it to relevant destinations based on its content.
3. The router forwards the message to the relevant destinations.
4. The message consumer receives the message.

The centralization of routing responsibility relieves the tight coupling and inflexibility of direct integration, decoupling the application from the destinations. The messaging infrastructure can now adapt to meet current ABL requirements without requiring the application to change. From the application's perspective, integration is simplified, because it just delivers all messages to the relevant company's router. The cost of integrating with further companies is minimal; no matter how complex the companies' messaging infrastructure, the application simply forwards the messages to their routers. However, this pattern entails some limitations.

### Centralized CBR Limitations

The advantages of the centralized CBR pattern emanate from its centralization of the message-sorting process. However, centralization becomes a weakness when it comes to scalability and robustness. From a message-delivery perspective, a centralized delivery mechanism can form a scalability bottleneck, which affects the messaging solution's robustness.

Companies that replicate their messaging infrastructures at multiple sites might not be able to consolidate all of the routing rules into centralized locations. Such implementations will need to replicate the CBR rule base at each site, increasing the solution's maintenance burden.

The choice of whether to centralize integration greatly affects the messaging solution's characteristics. Centralization simplifies integration, whereas decentralization improves performance. In particular, a messaging solution can improve scalability by performing the routing as close to the producer as possible.[13] An ideal routing solution would maintain the performance advantage of a distributed message delivery while preserving the maintenance benefits of a centralized rule base.

## Decentralized CBR Integration

A decentralized approach shares the centralized rule base with message participants, letting the programs distribute their messages directly to relevant destinations. To implement this, each company must express its rule base in a portable format and establish a mechanism for exchanging the rule base between participants. This would allow routing to take place along the edges of the integration scenario. Figure 3 illustrates this enhancement to the CBR approach.

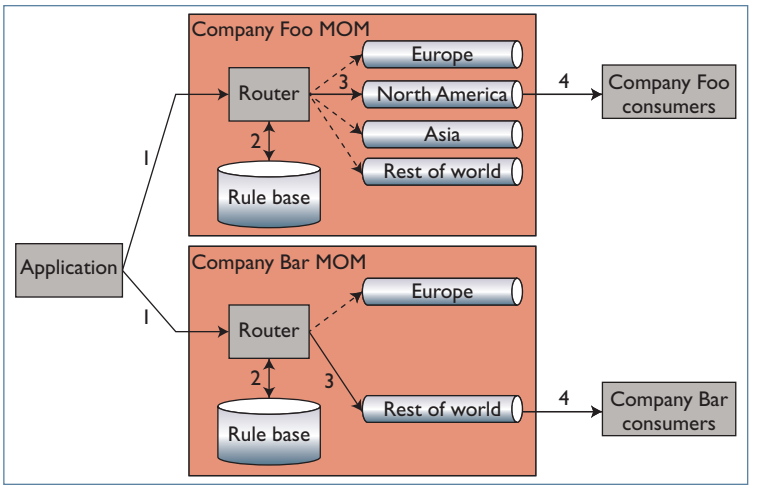


*Figure 2. Centralized content-based routing. An application sends messages to each company's router, which uses a rule base to determine where to send each message.*

Using this new approach, the hypothetical scenario would operate as follows:

1. The message producer receives routing rules from each company's rule base (as in Figure 3a). Routers can also exchange the rule bases (Figure 3b).
2. The message producer creates a message and matches it to the relevant destinations based on the rule base. The producer sends the message directly to its destinations.
3. The message consumer receives the message.

This solution provides the best of both worlds by minimizing maintenance with a centralized rule base while increasing scalability and robustness with decentralized message delivery. It captures the related ABL in the rule base in the same manner as the centralized CBR pattern does. However, the rule base's runtime portability offers several significant advantages.

A portable rule base is particularly useful for systems that replicate messaging infrastructures at multiple sites or for broker networks in which numerous routers possess the same routing responsibility. In these scenarios, which Router B in Figure 3 represents, the system can transfer the rule

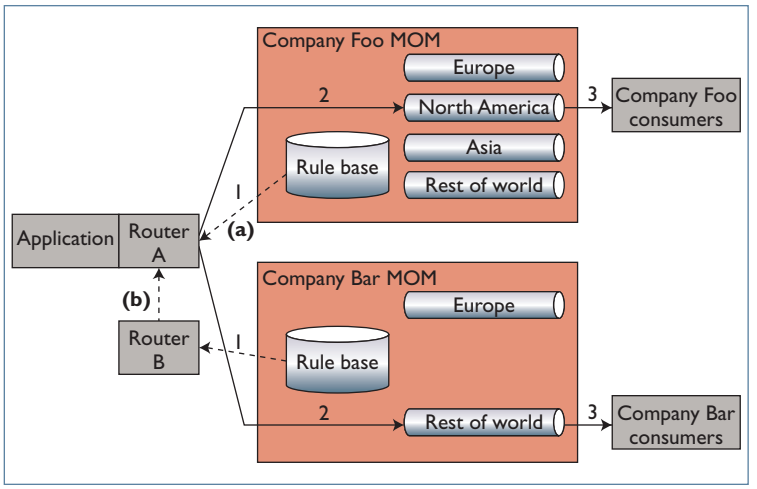


*Figure 3. Decentralized content-based routing deployment. An application's router can receive routing rules either (a) directly from a company's rule base or (b) by communicating with other routers.*

base to each site or router, or between routers, autonomously at runtime. This eases the maintenance burden by consolidating the rule base management into one location. Developers can thus make ABL changes related to the messaging infrastructure in one place, then propagate the changes to any location within the network.

The major drawback of this integration solution is the initial cost of setting up the support framework to facilitate the information exchange.[14,15] For simplistic or low-volume messaging environments, this support framework could be overkill.

A centralized CBR solution needs only standard messaging facilities and a rule-powered router. However, decentralized CBR requires two additional elements: routing rules expressed in a portable format and a mechanism for exchanging the rules. The MOM framework Generic Self-Management for Message-Oriented Middleware (Gismo) supports these requirements.

## Gismo

The Gismo framework provides general-purpose, coordinated self-management techniques for JMS-compatible MOM providers. Gismo opens up the internal dynamics of a MOM provider, enabling it to self-manage at runtime. I designed Gismo using the concepts from Gregor Kiczales's work on metaobject protocols.[16] One of the key aspects of his groundbreaking work is the separation of a system into two levels: a base level to provide system functionality and a metalevel to contain policies and strategies for system behavior. The Gismo metalevel represents and tracks the state, operations, and events that exist within a MOM. Gismo can attach to any base level (that is, any JMS-compatible message provider) noninvasively using the Chameleon framework.[14] As Figure 4 shows, Gismo features three distinct metamodels that cover destinations, subscriptions, and interception concerns. It also contains an event model and a reflective engine.

The *destination* metamodel tracks the existence and basic configuration of destinations, whether queue or topic, within the MOM. If the destination forms part of a destination hierarchy, the model captures additional information to track the relationships between destinations in the hierarchy.

Message consumers drive the main operating requirements and workload for a MOM. As such, the *subscription* metamodel is important for obtaining the MOM's current operational demands. The model monitors client activity by tracking the subscription details of message consumers, including subscription constraints.

Interception is a vital technique for self-managed systems because it offers a flexible mechanism for monitoring, altering, and extending the behavior of a base level at runtime. For instance, interception can inject functionality to execute every time the system adds a new subscriber, or when an application sends a message to a client. The *interception* metamodel details and tracks call interception and functionality injection at specific locations or point cuts within the MOM base level.

Gismo also provides a first-class *event* metamodel that covers all fundamental actions within the MOM environment, including messaging events, administrative events, and time-based trigger events.

The *reflective engine* contains Gismo's reflective self-management capabilities. Its objective is to define locations within the metalevel at which reflective computations occur and to provide a standard interface for reflective policies, which offer an encapsulation mechanism for reflective computational logic. The engine can place those policies either in-line with system execution for synchronous reflection or out-of-line for asynchronous reflection. Once in position, reflective policies perform their duties using the metalevel's full resources — the destination, subscription, interception, and event models — to effect change and reconfigure the MOM provider at runtime.

For the purposes of analyzing decentralized CBR, Gismo's destination metamodel and coordination capabilities hold the most relevance.

### Destination Metamodel

In its basic form, the destination metamodel (DMM) tracks the existence and basic configuration of destinations. This standard structure for destination state storage contains basic information about individual destinations. The core information this structure tracks is as follows:

- destination ID,
- destination name,
- destination type (queue or topic), and
- destination routing condition (optional).

The destination structure can store information on either a queue or a topic, providing the DMM with the ability to represent metainformation for destinations within a MOM. The DMM offers additional storage structures, but those are outside this article's scope.

With the destination model in place, reflective policies can access the model and alter it to meet current messaging requirements. As with any of the metamodels, reflective policies can create, update, and delete destinations and their configurations within the DMM. Once the metamodel has changed, Gismo instructs the underlying MOM provider to update itself. To achieve this, the destination metamodel defines a generic administration interface that includes common administration actions that MOMs take. The current implementation of the DMM provides realization support for three JMS-compliant providers: OpenJMS, ActiveMQ, and SonicMQ.

### Open Metalevel Interaction Protocol

Gismo facilitates the coordination of self-management techniques. The Open Metalevel Interaction Protocol (OMIP), which follows the guidelines previous work set out for metalevel coordination,[15] provides access to the system's internal self-management operations. It allows message deployment participants, whether clients or providers, to exchange configuration information, such as the destination metamodel, with each other, providing the final piece for a decentralized CBR approach to integration.

### Decentralized CBR with Gismo

Gismo implements the decentralized CBR pattern by expressing the routing instructions from the rule base within the DMM using the `condition` attribute. Participants can now easily exchange the DMM, which contains the rule base, by using OMIP. Figure 5 shows a sample DMM for the hypothetical scenario.

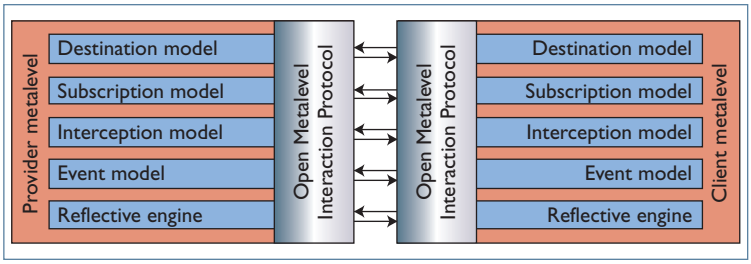


*Figure 4. Gismo metalevel architecture. MOM clients and providers exchange configuration information via the Open Metalevel Interaction Protocol (OMIP).*

Using the DMM, information producers can obtain the purpose of each destination, surmise its relevance to ABL, and route messages to the relevant location. The current incarnation of decentralized CBR assumes a strict consistency model, with each client requiring the latest version of the DMM before it can participate in messaging activity.

By facilitating the coordination of self-managing messaging systems, Gismo allows decentralized CBR systems to regulate themselves, providing another piece in the puzzle of a balanced and flexible MOM solution. Future opportunities include using Gismo to coordinate MOM self-management techniques, thus increasing the flexibility of messaging solutions. However, if coordination techniques such as portable rule bases are to succeed, the developer community must reach a consensus on their definition. To this end, including Gismo mechanisms such as the destination metamodel and OMIP within industrial messaging standards would facilitate their widespread use in messaging solutions.

### Acknowledgments

I thank the reviewers for their constructive comments.

### References

1. G. Banavar et al., "A Case for Message Oriented Middleware," *Proc. 13th Int'l Symp. on Distributed Computing*, Springer-Verlag, 1999, pp. 1–18.
2. D. Orchard, "Achieving Loose Coupling," *Dev2Dev*, 2 Feb. 2004; http://dev2dev.bea.com/pub/a/2004/02/orchard.html.
3. P.T. Eugster et al., "The Many Faces of Publish/Subscribe," *ACM Computing Surveys*, vol. 35, no. 2, 2003, pp. 114–131.
4. D. Skeen, "An Information Bus Architecture for Large-Scale, Decision-Support Environments," *Proc. Winter 1992 Usenix Conf.*, Usenix Assoc., 1992, pp. 183–196.

```
<DestinationMetaModel>
    <Destination id="Foo_Inbox1" name="Europe" type="queue">
      <condition>region="Europe"</condition>
    </Destination>
    <Destination id="Foo_Inbox2" name="NorthAmerica" type="queue">
      <condition>region="North America"</condition>
    </Destination>
    <Destination id="Foo_Inbox3" name="Asia" type="queue">
      <condition>region="Asia"</condition>
    </Destination>
    <Destination id="Foo_Inbox4" name="RestOfWorld" type="queue">
      <condition>region NOT IN ("Europe","North America", "Asia")</condition>
    </Destination>
</DestinationMetaModel>
```

**(a)**

```
<DestinationMetaModel>
    <Destination id="Bar_Inbox1" name="Europe" type="queue">
      <condition>region="Europe"</condition>
    </Destination>
    <Destination id="Bar_Inbox2" name="RestOfWorld" type="queue">
      <condition>region NOT IN ("Europe")</condition>
    </Destination>
</DestinationMetaModel>
```

**(b)**

*Figure 5. Destination metamodels for the hypothetical scenario. Routing rules determine message destinations for (a) company Foo and (b) company Bar.*


5. L. Gilman and R. Schreiber, *Distributed Computing with IBM MQSeries*, Wiley & Sons, 1997.
6. D.K. Bradshaw et al., *Oracle 9i Application Developer's Guide—Advanced Queuing*, Oracle, part no. A96587-01, Mar. 2002; http://download-uk.oracle.com/docs/cd/B10501_01/appdev.920/a96587/toc.htm.
7. A. Carzaniga, D.S. Rosenblum, and A.L. Wolf, "Design and Evaluation of a Wide-Area Event Notification Service," *ACM Trans. Computer Systems*, vol. 19, no. 3, Aug. 2001, pp. 332–383.
8. L. Fiege et al., "Supporting Mobility in Content-Based Publish/Subscribe Middleware," *Proc. ACM/IFIP/Usenix Int'l Middleware Conf.* (Middleware 2003), Springer-Verlag, 2003, pp. 103–122.
9. P.R. Pietzuch, *Hermes: A Scalable Event-Based Middleware*, doctoral dissertation, Queens' College, Univ. of Cambridge, 2004; www.eecs.harvard.edu/~prp/doc/prp_thesis.pdf.
10. B. Segall et al., "Content Based Routing with Elvin4," *Proc. Australian UNIX and Open Systems Users Group* (AUUG2K), 2000; www.mantara.com/community/papers/auug2k.pdf.
11. M. Hapner et al., "Java Message Service Specification, Final Release 1.1," specification by Sun Microsystems, Mar. 2002; http://java.sun.com/products/jms/docs.html.
12. G. Hohpe and B. Woolf, *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions*, Addison-Wesley Professional, 2003.
13. P.T. Eugster, et al., "Event Systems: How to Have Your Cake and Eat It Too," *Proc. 22nd Int'l Conf. on Distributed Computing Systems Workshop* (ICDCSW 02), IEEE CS Press, 2002, pp. 625–632.
14. E. Curry, D. Chambers, and G. Lyons, "Extending Message-Oriented Middleware Using Interception," *Proc. 3rd Int'l Workshop on Distributed Event-Based Systems* (DEBS 04), Institute of Eng. and Tech., 2004, pp. 32–37.
15. E. Curry, D. Chambers, and G. Lyons, "ARMAdA: Creating a Reflective Fellowship (Options for Interoperability)," *Proc. 3rd Workshop on Adaptive and Reflective Middleware*, ACM Press, 2004, pp. 226–231.
16. G. Kiczales, J.D. Rivieres, and D.G. Bobrow, *The Art of the Metaobject Protocol*, MIT Press, 1991.



**Edward Curry** served as a research fellow in the Information Technology Department at the National University of Ireland, Galway. His main research interests include self-* and nature-inspired middleware. Curry has a PhD in computer science from the National University of Ireland, Galway. He is a member of the IEEE and the ACM. Contact him at edcurry@acm.org.