Event-Based Middleware: A New Paradigm for Wide-Area Distributed Systems?

Peter R. Pietzuch[1]
University of Cambridge Computer Laboratory
{Peter.Pietzuch}@cl.cam.ac.uk

Abstract:

Large-scale, internet-wide applications are becoming commonplace. Such systems can consist of many heterogeneous components distributed over several continents. Whereas middleware platforms like CORBA or Java RMI are useful for building distributed applications at a small to moderate scale, we do not have similar tools for large-scale systems. However, without a middleware abstraction it is difficult to deal with the complexity of global systems.

In this paper, we describe a novel kind of middleware, Hermes, that uses event-based communication as its primary paradigm. Nevertheless, it supports standard middleware functionality. It is built on top of an overlay routing network and uses intrinsically scalable algorithms for event dissemination.

1 Introduction

Distributed systems have a number of advantages over conventional, centralised systems. They can support a larger number of users at a smaller cost and the overall availability of the system is higher than of a centralised solution. Increased performance of the system can be provided to the users by adding small, inexpensive components as the systems grows. However, the cost to be paid for these benefits is the increased complexity of the system which has to be managed. This complexity mainly comes from the fact that a potentially large set of autonomous and heterogeneous components are part of the distributed system.

In order to simplify the problem, *middleware platforms* were developed that run on top of heterogeneous operating systems and provide a homogeneous, abstract view of the entire distributed system. Today, most middleware systems like *CORBA* or *Java RMI* are *invocation-based* and thus follow a *request/reply paradigm*: A client requests a particular service from a server by either sending a request message or performing a remote method invocation (RMI) and then receives a reply in return.

Although such a mode of operation works well in a local area network (LAN) context with a moderate number of clients and servers, it does not scale to large networks like the Internet. This is mainly because the request/reply paradigm only supports *one-to-one communication* where a single client interacts with a single server. In contrast, large-scale systems benefit from *many-to-many communication* since the client does not have to decide on the best communication partner. Another problem is the tight-coupling of request/reply middleware: The method invocation is synchronous forcing the client and server to couple at one particular point in time [10]. Such a behaviour is clearly not desirable on the Internet because of the large number of potential communication partners and the dynamic nature of the system with new clients joining and servers failing.

A different underlying communication paradigm for building large-scale distributed systems on top of a middleware seems to be necessary. In this paper, we argue that *event-based communication* [2] is

a viable new alternative for doing this. In an event-based system, events are the basic communication mechanism: First, *event subscribers*, i.e. clients, express their interest in receiving certain events in the form of an *event subscription*. Then *event publishers*, i.e. servers, publish events which will be delivered to all interested subscribers.

As a result, this model naturally supports a decoupled, many-to-many communication style between publishers and subscribers. A subscriber is usually indifferent to which particular publisher supplies the event that it is interested in. Similarly, a publisher does not need to know about the set of subscribers that will receive a published event.

In a world where millions of small and inexpensive devices can be provided with network connectivity and integrated into a distributed system, a vision like the *Active City* could be supported by an underlying event-based middleware. In the active city, houses, offices, public buildings, transportation services, and administrative services are interlinked and can interact with each other. Every device can potentially talk to every other device and provide or request information. A vast portion of the devices are mobile and autonomous so that no central administrative authority can be established. A scalable and flexible middleware is essential in such a world. Event-based communication appears to be much more suited than other traditional request/reply paradigms to cope with these requirements.

In this paper, we express our idea of an event-based middleware architecture by pointing out the requirements and presenting, *Hermes*, an event-based middleware.

2 Event-Based Middleware

Even though several event-based, publish/subscribe systems [6,4,16,9] have been developed over the past years, little work has been done to unite the areas of middleware systems and publish/subscribe communication to provide, what we call, an *event-based middleware*. Traditional publish/subscribe systems have very limited application scenarios like stock quote dissemination [4] or instant messaging [16]. Middleware functionality, such as type-checking of invocations, reliability, access control, transactions, and so on, is often neglected. Our work focuses on providing a scalable event-based middleware that is powerful enough to be the building layer for any large-scale distributed application that would traditionally be implemented with an invocation-based middleware. We envision a world with global e-commerce and business applications, and complex systems like an active city with thousands of components. We have identified a number of important middleware and publish/subscribe features that must be provided in an event-based middleware system.

Scalability is a crucial requirement for Internet-wide applications. A system is only scalable if all its components are, which means that the implementation of the middleware system must not rely on any centralised services. Moreover, algorithms must not keep any global state, and resources like network bandwidth and memory must be consumed efficiently.

Interoperability should allow the integration of a variety of components with the middleware. The event model and the subscription language must be language- and platform-independent. The middleware must not rely on any particular support from the underlying network that is not universally available like IP multicast. However, it should take advantage of available services for performance reasons.

Reliability when delivering events may be one of the *quality of service* (QoS) requirements requested by event clients or servers. The middleware must support a range of QoS guarantees, from ``best-effort'' to ``guaranteed and timely'' event delivery. Fault-tolerance mechanisms such as

persistent events stored in a database allow the middleware to operate in the light of client and server failures.

Expressiveness is an important requirement when specifying events and subscriptions. Subscriptions must allow filtering depending on the content of events (content-based filtering). *Composite event expressions* [15] that detect patterns in the event stream are an intuitive and powerful higher-level abstraction that helps event subscribers to express their information need. Nevertheless, there is always a trade-off between expressiveness and efficiency [5].

Usability means that the middleware is easy to handle. It should cleanly integrate with the application programming language. For instance, events should be typed objects that are mapped transparently to programming language objects. Linguistic supports involves type-checking of events and subscriptions and tools for the construction of complex composite event expressions.

3 Hermes

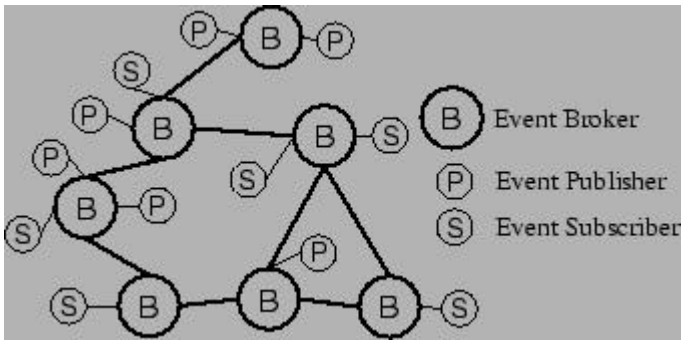

Figure 1: A Distributed Application built with Hermes

In our research group we have developed Hermes [12], a distributed, event-based middleware architecture. It consists of two components, *event clients* and *event brokers*. Event clients can be *event publishers* or *event subscribers*. The main functionality of the middleware is implemented by the event brokers. As a result, event clients are light-weight components that can easily be implemented in any programming language guaranteeing language-independence of the architecture. They communicate with the event brokers by passing XML messages.

Event brokers are interconnected in an arbitrary topology. They use a *content-based routing* algorithm [7] to deliver events from publishers to subscribers. The filtering of event streams depending on the subscriptions is done in a distributed fashion. It happens as close to the event source as possible, thus saving network bandwidth and increasing scalability (source-side filtering). Figure 1 shows a distributed application built on top of Hermes.

In order to provide a scalable event routing algorithm that can deal with node failures, Hermes relies on the service of a peer-to-peer overlay routing network like [13,17]. The event brokers are the nodes of an overlay network that supports a `route(message, destination_id)` function that enables a broker to send a message containing an event to any other broker. This event is then routed via the overlay network, and the event brokers along the path apply their filtering expressions.

Using an overlay network for event dissemination has several advantages: Firstly, events are routed without any need for global state. No event broker in the system must have knowledge of all the subscriptions or event sources. Secondly, the entire system is more robust to error because the overlay network can transparently adapt to node or link failures. Finally, the overlay network is used

to set up *rendezvous nodes* [3]. These special brokers with well-known addresses make sure that published events are disseminated along paths towards all the subscribers that are interested in a particular event [14].

To alleviate the impedance mismatch between events and programming language objects, all events in Hermes are an instance of a particular *event type*. Event types contain *event attributes* and are organised in an object-oriented inheritance hierarchy. When an event client wants to subscribe to an event, it first decides on the event type it is interested in and then provides a *filtering expression* based on the event attributes in this type. Such a *type- and attribute-based publish/subscribe* scheme makes it simpler to decide which events are of interest. Moreover, it enables Hermes to type-check subscriptions at runtime which is not possible in purely content-based publish/subscribe [6].

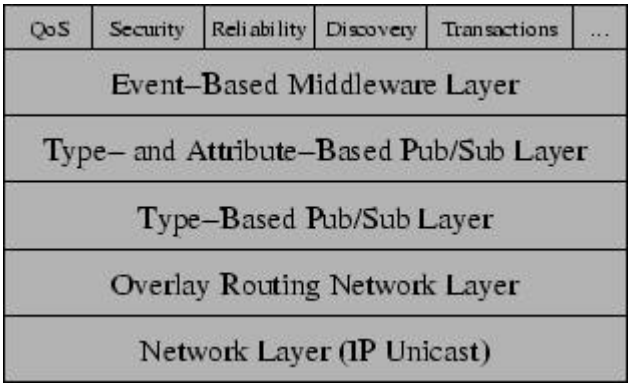

Figure 2: The Layered Architecture of Hermes

The architecture of Hermes follows a layered approach as shown in Figure 2. The middleware is assumed to be deployed on an IP unicast network such as the Internet. The *overlay routing layer* provides a basic communication service between the event brokers. The two *publish/subscribe layers* implement the event dissemination algorithms with source-side filtering. Traditional middleware services like QoS, security, reliability, etc. are implemented in the form of modules on top of the *event-based middleware layer*.

4 Future Work

We are currently working on a full implementation of Hermes within a message-based, discrete event simulator. Since Hermes is aimed at supporting global scale distributed applications, it is important to have a simulation environment that can support at least 10^5 nodes with a large number of event publishers and event subscribers. Our goal is to be able to simulate isolated parts of our active city vision. Performance measurements should support our claims of the overall scalability of the Hermes architecture.

Another active area of interest is the distributed detection of composite events. Previous work [15,11] assumed that either it is possible to put an upper bound on the network delay, or the identity of all event sources is known to the composite event detector. Both assumptions are not valid for a global-scale system. We are experimenting with a heartbeat protocol that gives a confidence interval to the event detector about the likelihood of delayed events that might invalidate the detection of a composite event pattern.

Storing events in a persistent store was identified as a prerequisite for the provision of a reliable event service. It enables an event broker to guarantee the correct delivery of an event to all interested subscribers even when event brokers crash during transit of the event. [1] describes our work on expressing event types using the Object Definition Language (ODL) [8]. These events can then be

directly inserted into an ODMG-compliant object-oriented database. We are planning to extend a Hermes event broker to incorporate a persistent store used for reliably delivering events.

5 Conclusions

In this paper, we have briefly described the potential of the event-based paradigm for building large-scale distributed systems using a novel middleware architecture. We motivate our approach by the fact that wide-area systems are growing in importance as network connectivity is falling in price. As a result, large federated systems consisting of thousands of systems can feasibly be built now in many applications areas, such as global e-commerce or large-scale active environments like the active city. Traditional, invocation-based middleware does not seem to be up to the task. Loose coupling between components in these systems plays a central role because of their dynamic nature and heterogeneity.

The event-based paradigm gives ways of dealing with these new requirements and provides an intuitive model for building systems. The Hermes architecture combines traditional middleware functionality with event-based communication. It uses a peer-to-peer overlay routing network as an abstraction if IP multicast support is not available. Type- and attribute-based publish/subscribe integrates cleanly with programming language types.

However, event-based middleware will only be successful in the future if we do not forget the lessons learnt from decades of middleware research. Supporting traditional middleware functionality like QoS, security, transactions etc. is essential. We feel that our work is a first step in that direction.

Bibliography

1 Jean Bacon, Alexis Hombrecher, Chaoying Ma, Ken Moody, and Walt Yao.
Event Storage and Federation using ODMG.
In *Proc. of the 9th International Workshop on Persistent Object Systems: Design, Implementation and Use (POS9)*, Lillehammer, Norway, September 2000.

2 Jean Bacon, Ken Moody, John Bates, Richard Hayton, Chaoying Ma, Andrew McNeil, Oliver Seidel, and Mark Spiteri.
Generic Support for Distributed Applications.
IEEE Computer, pages 68-77, March 2000.

3 Tony Ballardie, Paul Francis, and Jon Crowcroft.
Core Based Trees (CBT).
In *ACM SIGCOMM'93*, Ithaca, N.Y., USA, 1993.

4 Guruduth Banavar, Tushar Chandra, Bodhi Mukherjee, Jay Nagarajarao, Robert E. Strom, and Daniel C. Sturman.
An efficient multicast protocol for content-based publish-subscribe systems.
1999.

5 Antonio Carzaniga, David S. Rosenblum, and Alexander L. Wolf.
Achieving scalability and expressiveness in an internet-scale event notification service.
1999.

6 Antonio Carzaniga, David S. Rosenblum, and Alexander L. Wolf.
Design and evaluation of a wide-area event notification service.
ACM Transaction on Computer Systems, 2001.

7 Antonio Carzaniga and Alexander L. Wolf.
Content-based networking: A new communication infrastructure.
In *NSF Workshop on an Infrastructure for Mobile and Wireless Systems*, Scottsdale, AZ, October 2001.

8 R. G. G. Cattell, Douglas Barry, Dirk Bartels, Mark Berler, Jeff Eastman, Sophie Gamerman, David Jordan, Adam Springer, Henry Strickland, and Drew Wade.
The Object Database Standard: ODMG 2.0.
Morgan Kaufmann, 1997.

9 G. Cugola, E. Di Nitto, and A. Fuggetta.
The JEDI event-based infrastructure and its applications to the development of the OPSS WFMS.
IEEE Transactions on Software Engineering, 1998.

10 P. Th. Eugster, P. Felber, R. Guerraoui, and A.-M. Kermarrec.
The Many Faces of Publish/Subscribe.
Technical report, EPFL Lausanne, 2001.

11 C. Liebig, M. Cilia, and A. Buchmann.
Event Composition in Time-dependent Distributed Systems.
In *Proc. of the Fourth IECIS International Conference on Cooperative Information Systems*, 1998.

12 Peter R. Pietzuch and Jean M. Bacon.
Hermes: A Distributed Event-Based Middleware Architecture.
Submitted to the Workshop on Distributed Event-Based Systems (DEBS), 2002.

13 Antony Rowstron and Peter Druschel.
Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems.

In *Proc. of Middleware 2001*, November 2001.

14 Antony Rowstron, Anne-Marie Kermarrec, Miguel Castro, and Peter Druschel.
Scribe: The design of a large-scale event notification infrastrucutre.
In *Proc. of the 3rd Int. Workshop on Networked Group Communication (NGC2001)*, November 2001.

15 Scarlet Schwiderski.
Monitoring the Behaviour of Distributed Systems.
PhD thesis, Computer Laboratory, University of Cambridge, 1996.

16 Bill Segall and David Arnold.
Elvin has left the building: A publish/subscribe notification service with quenching.
In *Proc. of AUUG Technical Conference 1997*, July 1998.

17 Ben Y. Zhao, John Kubiatowicz, and Anthony D. Joseph.
Tapestry: An infrastructure for fault-tolerant wide-area location and routing.
Technical report, Computer Science Division, University of California, Berkeley, April 2001.

Footnotes

... Pietzuch[1]

Research supported by QinetiQ, Malvern

Peter Pietzuch 2002-01-17