An Architecture Proposal for Enterprise Message Brokers

Jörn Guy Süß[1] and Michael Mewes[2]

[1] Technical University Berlin
Einsteinufer 17, D-10587 Berlin / Germany
`jgsuess@cs.tu-berlin.de`
[2] Fraunhofer Society for the Promotion of Applied Research,
Institute for Software and Systems Engineering
Mollstraße 1, 10178 Berlin / Germany
`michael.mewes@isst.fhg.de`

Abstract. In large enterprises, asynchronous communication and messaging are gaining importance as integration solutions between applications. The concept of a message broker has been proposed as a universal mediator at the center of business. This paper gives a distinct definition of a message broker by enumerating enterprise critical criteria and describes a reference architecture to meet these criteria. To arrive at the relevant criteria, the message broker is positioned with respect to other middleware solutions like CORBA and MOM/DAD, and limitations and advantages are pointed out. From this comparison the catalogue of critical criteria is deduced and examples are given of how commercial broker products fulfill or fail these criteria. Finally, a reference architecture based on Java, XML and XSL(T) is described that meets the criteria with respect to configuration, execution and extensibility.

1 Limitations of Object-Oriented Middleware Regarding EAI

Middleware, as envisioned by its promoters, primarily views the enterprise as a large, distributed, object-oriented or component oriented system interacting on the basis of synchronous remote procedure calls: Business is modeled as binary objects, talking to each other over a transparent network. This situation comprises three implicit fundamental assumptions:

- The two partners must know of each other, i.e. their interfaces must be well published and well defined. This implies that interfaces have to be global, stable metadata.
- The two partners must be able to communicate. This requires compatibility to a common network medium and a common means of binary communication.
- The two partners must be available; i.e. transactions must be possible. This requires availability of all interfaces at all time or significant effort invested in exception handling for transaction failures.

Apart from touching on the political implications of data ownership, which arise in every integration effort, the aforementioned factors are also the main obstacles of transferring traditional middleware development to the enterprise scale. Projects are characterized by having local scope, local network or platform and local transactions. As a consequence, the systems that result, although they are distributed to increase performance and availability, are in fact local solutions. The following sections discuss the three aspects of the local character of middleware applications.

1.1 Local Scope

The interfaces for distributed systems are normally architected in a project context, which emphasizes the delivery of end-to-end functionality. The middleware, being "invisible" from both the database and the user interface, and therefore outside of the scope of the process owner, often has its design time reduced to deliver project scope functionality more quickly. As a result, interfaces are narrowed to the task the system in question is being built for. Even if there is a consensus to build cross-company reusable interfaces, the results are complex and unwieldy to use and suffer from the inclusion of too much detail as shown by typical “Jack-Knife-Classes” [1]. For this reason global interface repositories are very rare.

1.2 Local Network or Platform

The middleware approach also lightly assumes that there is such a thing as a binary and network compatible platform. This assumption does not hold for the large number of business critical batch systems, which lack a notion of networks, let alone TCP/IP. The assumption is still hard to cover for the "open platforms", which make up the rest of the business critical systems in the enterprise. Components and ORBs vary, sometimes even if bought from the same vendor, for different platforms or in different versions. And they are regularly not interoperable when provided by different vendors. Since problems in critical interoperability could not be attributed to any one vendor, decision-makers are faced with a stark choice: Either the company opts to choose one product vendor only. This means costly vendor lock-in. Additionally it requires rigid discipline to enforce the standard. Or the company leaves the choice of the middleware product to the project team, so that each system uses its own middleware product. Most companies sensibly opt for the second choice.

1.3 Local Transactions

Finally, to create a business application from objects or components spread out across an enterprise necessitates that the underlying structure can provide distributed transactions spanning several objects or components on different levels. This feature causes a major overhead if implemented. Or it eliminates the structure of the object oriented interfaces if left out, leading to a flat TP-light, two-tier solution. Initially, middleware packages did not even include transactions. Even today, programming transactional middleware in an open manner is cumbersome at best. For this reason, middleware architectures often consist of a well-architected non-transactional information system, and a one-layer transaction system to perform updates, which maps methods 1:1 to stored procedures of the underlying database.