Too Much Middleware

Michael Stonebraker
EECS Department
M.I.T.

Abstract

The movement from client-server computing to multi-tier computing has created a potpourri of so-called middleware systems, including application servers, workflow products, EAI systems, ETL systems and federated data systems. In this paper we argue that the explosion in middleware has created a myriad of poorly integrated systems with overlapping functionality. The world would be well served by considerable consolidation, and we present some of the ways this might happen. Some of the points covered in this paper have been previously explored in [BERN96].

1. Introduction

A typical Fortune 1000 company has a myriad of mission critical data systems on which the enterprise depends. Such systems include ERP systems, sales tracking systems, HR systems, etc. Over the last twenty years, the conventional wisdom was to implement such applications using a client-server computing model as noted in Figure 1. Here, the DBMS (or other storage technology) ran at the server level, accessed by a collection of applications which ran on the client desktop. Client-server computing was enabled by the emergence of desktop PC's which provided a client computing platform, and was pushed by many software and hardware companies, that were selling alternatives to IBM mainframes.

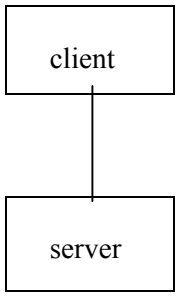

Figure 1. Client-server Architecture

Recently, several factors have rendered client-server computing completely obsolete, but we will focus only on the web in this paper. Basically, it forces every enterprise to move to the architecture of Figure 2. The client level from Figure 1 moves inside a web browser, where at most a portion of the application from Figure 1 can run. Sometimes an ultra-thin client is run, whereby no code exists at the client level. Other enterprises utilize Java, Javascript, or other technology to run some code on the client desktop. In either case, the remainder of the application must be executed in a new middleware tier.

The web also enables cross-enterprise e-commerce applications. In this world, the client is in another organization outside the enterprise firewall, and it is typically impractical to run much (if any) application code at the client level. Again, the thin client architecture of Figure 2 is essential.

This switch from Figure 1 to Figure 2 creates a new tier of computing, commonly called middleware. Moreover, it has created several new classes of products, that run at this level. In Section 2 we will review the major classes of products that fit in this category. Then, in Section 3 we show that the various product classes have dramatic overlap in functionality. Moreover, this overlap will likely increase in the future. We then argue that this product overlap is very bad for customers, leading to complex and inefficient environments. Lastly, in Section 4 we project some scenarios, whereby a consolidation of middleware products could occur.

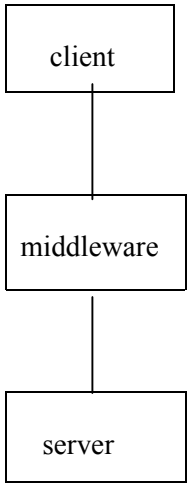

Figure 2. Multi-tier Architecture

2 Middleware Products

In this section we will discuss six classes of middleware products, namely application servers, enterprise application integration (EAI) systems, workflow systems, enterprise portals, data federation systems, and extract, transform and load (ETL) products. For each kind of system, we briefly describe its core functionality and its reason for existence.

2.1 Application Servers

Application servers have been in existence for more than twenty-five years. Historically, they were called TP monitors, the most popular being the IBM Customer Information and Control Systems (CICS). Recently, the category has been renamed application servers, and includes products such as Silverstream, Dynamo from ATG, Web Logics from BEA Systems, Bluestone (now owned by HP), Cold Fusion from Allaire and WebSphere from IBM.

TP monitors came into existence to allow multiple client users to run the same application efficiently on a mainframe computer. Specifically, they provided **application activation**. To perform efficiently, they also provided multi-threading of applications (that were written to allow threading) and connection multiplexing. The latter feature lowered the number of connections that had to be maintained between the middleware layer and the server level, typically increasing efficiency. To provide scalability, many application servers support automatic load sharing over multiple hardware machines. Lastly, most provide a security module including authentication, single site login, and access control.

In summary, an application server is capable of providing code activation and related services. Since the underlying operating systems were slow to provide multithreaded execution, this class

of products developed a foothold. More recently, the architecture of the web created a need for web servers to execute web protocols and CGI programs. This naturally evolved into full-fledged web-oriented application servers.

2.2 EAI Products

A typical large enterprise has more than 5000 major application systems. Such systems are always logically interconnected. For example, when a new employee is hired, he must be inserted into the payroll systems, added to the pension plan, added to the medical insurance system, etc. When a customer changes his telephone options, a change must be made to the billing system as well as to the provisioning system.

Whenever two companies merge, there are two sets of information systems to contend with, one from each of the partners. It is desirable, but rarely possible, to kill one of the two overlapping systems, at least not immediately. Again, multiple interacting application systems are part of the resulting enterprise architecture.

A last example arises in e-commerce. The purpose of a net market is to connect a transacting buyer and seller. Here, transaction information must be communicated from the net market application to the procurement system of the buyer and the order processing and fulfillment systems of the seller. Again, multiple interacting applications are present.

To service the needs of interacting application, a class of products called enterprise application integration (EAI) systems arose. The core functionality is to reliably deliver a message from one application to a second one, which needs to be alerted concerning an action taken by the first one. Such a messaging service is a core functionality of EAI systems.

However, the two communicating systems, which were written independently, **never** agree on the syntax and semantics of application constructs. A transaction to the net market is never the same as the one expected by the buyer's procurement system or the seller's order processing system. Hence, there is a need for **message transformation** between the originator and the recipient.

Lastly, the two communicating applications are never written using the same technology. A home-brew order processing must communicate with a net market based on Ariba. A customer relationship system written using Siebel must communicate with an ERP system from SAP. In this heterogeneous world, a collection of **adapters** is required. A source adapter transforms the source message to a common form while a target adapter changes the common form to that required by the recipient.

The basic functionality provided by EAI systems is message delivery and transformation. Successful EAI products all have adapters for a variety of popular application systems and support a facility to construct new ones for custom legacy systems. Products in this category include Vitria, Crossworlds, CommerceQuest, MQSeries from IBM, and Tibco.

In summary, the requirement for communication between disparate application systems fueled a market in EAI products. These products form another kind of middleware that runs in the middle tier of Figure 2.

2.3 Workflow Systems

Workflow systems have also been in existence for many years. The early systems were oriented toward procurement, and the focus was on process flow. A typical purchase order must be signed by the manager of the originating employee. If it is large enough, it must also be signed by a division manager. If it entails computer equipment, it must be signed by the CIO. A typical large enterprise has tens or hundreds of such rules. The process of obtaining approval for a purchase order entails moving through a collection of approval steps, whose sequencing is governed by business rules.

The best way to think about such purchasing rules is as a "boxes and arrows" diagram, as indicated in Figure 3. Each box represents a processing step, or application. Applications are interconnected by arrows to indicate flow and by decision points to indicate routing. Typical workflow systems embrace a boxes and arrows user interface, which provides a visual representation of process flow.

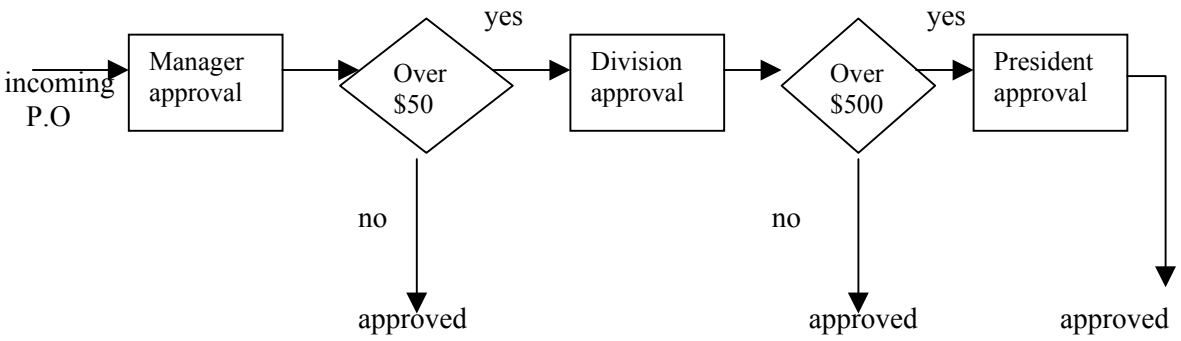

Figure 3. A Boxes and Arrows Diagram

Of course, a workflow system must also have an execution environment. This run time environment must be capable of executing applications, as well as supporting connectivity between one application and the subsequent one in the process flow. Hence diagrams such as Figure 3 are typically compiled into some sort of middleware framework, and executed in the middle tier of the Figure 2 architecture. Products in this category include Versata and Flowmark from IBM.

2.4 ETL Systems

A large enterprise has more than 5000 major application systems that hold data critical to the efficient functioning of the enterprise. There is an obvious need for **data integration**, so that a business analyst can get a more complete picture of the enterprise and how to optimize it.

The conventional wisdom to perform this task is to buy a giant machine located in a safe place (say under Mount Washington). Periodically, desired data is "scraped" from each operational data system and copied to Mount Washington. Of course, the various operational systems never have a common notion of enterprise objects, such as a purchase order. Hence, there is a requirement to transform source data to a common format before loading data onto the Mount Washington machine.

Large common machines came to be known as **warehouses**, and the software to access, scrape, transform, and load data into warehouses, become known as extract, transform, and load (ETL) systems. In a dynamic environment, one must perform ETL periodically (say once a day or once a week), thereby building up a history of the enterprise.

The purpose of this history is to perform data mining, to extract unknown patterns from the data. The common folklore (apparently fictional) is that a convenience store chain employed this technique to ascertain that diapers and beer were usually purchased together (seemingly by males sent out to buy diapers and getting beer as a reward). The story goes that the store co-located the beer and diapers, and that sales of both items increased. This story illustrates the principle of data mining, i.e. uncovering hidden relationships in corporate data. The main purpose of a data warehouse is to allow systematic or ad-hoc data mining.

Early users of data mining were the retail giants, who applied the technique to item-level sales analysis for the purpose of stock rotation. More recently, the technique has been widely embraced. In the future, data mining will only increase, as it is applied to web-oriented click-stream analysis. E-commerce web sites need to know what their customers are doing, and why they are abandoning the majority of their shopping carts, etc.

The requirement for data integration has driven the ETL market, and products are available from Informatica, Ascential, and the major data base vendors.

2.5 Portals

Portal software has also been in existence for many years. Historically, there was a need for executive information systems, i.e. an easy to use interface that allowed retrieval of executive level information from enterprise information systems. Executive information systems (EIS) came into existence to fulfill this need for casual, ad-hoc interaction with corporate data systems.

With the advent of the web, the desire is to make such systems web-oriented. Moreover, the goal is to extend the reach of such systems beyond top-level executives. As such, the goal of the EIS vendors becomes **enterprise self-service,** i.e. casual, ad-hoc access to enterprise systems by end users. Instead of phoning an HR clerk with a desired change in one's employee benefits, who then enters the information into an enterprise system, one wishes to allow the employee to make the change directly using a web-based interface. Hence, the EIS vendors have broadened their focus to employee self-service and renamed their products to be portals.

Hence, portal products must provide an easy to use interface to corporate application systems. Off into the future, portal vendors envision extending their franchise from friendly interfaces for data integration to providing the same easy-to-use interfaces for business intelligence applications. Hence, end users and executives would be able to obtain access to data mining tools, currently the exclusive domain of business analysts. As such, portal software is increasingly providing services similar to ETL and EAI products.

Popular portal products include Plumtree, Viador, and offerings from the major application server vendors and database vendors

2.6 Data Federation Systems

The conventional wisdom is to use data warehousing and ETL products to perform data integration. However, there is a serious flaw in one aspect of this wisdom. Suppose one wants to integrate current (operational) data rather than historical information.

Consider, for example, an e-commerce web site which wishes to sell hotel rooms over the web. The actual inventory of available hotel rooms exists in 75 or so information systems. After all, Hilton, Hyatt and Marriott all run their own reservation systems. Applying ETL and warehousing to this problem will create a copy of hotel availability data, which is quickly out of date. If a web site sells a hotel room, based on this data, it has no way of guaranteeing delivery of the room, because somebody else may have sold the room in the meantime.

The only way to guarantee correct data integration in this environment is to fetch the data at the time it is needed from the originating system. Data integration in dynamic environments requires integration on demand, not integration in advance.

Dynamic data integration has created a market for data federation systems. These products construct a composite view of disparate data systems and allow a user to run SQL commands, including retrieves and updates to this composite view. Then, they translate each SQL command on this composite view to a collection of local SQL commands that "solves" the user's request.

The need for dynamic data integration has driven the market for data federation systems, and products in this category include systems from Cohera (now owned by Peoplesoft), IBM, and Sybase.

3. Commonality

The various kinds of middleware described in the previous section were motivated by very different requirements as indicated in Figure 4. A typical large enterprise has needs in each area, and therefore has purchased at least one of each class of system. Some enterprises have more than one in each category. Although the classes of systems started with different objectives, they have grown to have highly overlapping functionality. This is explored more fully by revisiting each of the system categories, and indicating the current functionality of the best-of-breed participants.

Let us begin with EAI systems. In order to provide transformations, an EAI system must be able to activate functions that provide such transforms. This functionality requires an application server, and most EAI products embed at least a lightweight application server. In addition, many transformations require more than one step. Hence, the best-of breed EAI systems provide multi-step transformations. Hence, good EAI systems provide at least a simple workflow system.

ETL systems must provide transformations and thereby must include at least a simple application server, and multi-step transformations entail workflow. As such, the major difference between ETL and EAI is that ETL focuses on bulk data movement while EAI deals with real-time movement.

Let us move to data federation systems. Since all real world schemas that need to be integrated are heterogeneous, the best-of-breed systems support transformations to a common format during data access. As such, they also provide the function activation necessary to execute such functions. In addition, they require adapters to connect to legacy data systems. Finally, they

support the ability to construct new tables from queries on existing tables. The execution of such queries can invoke transformations to a common schema. As such, they provide, in effect, an ETL system as a portion of their functionality.

System	Purpose
Application server	Application activation
EAI	Application connectivity
Workflow	Process sequencing
ETL	Historical data integration
Portals	Enterprise self-service
Data federations	Dynamic data integration

Figure 4. The Various Kinds of Middleware

Portals also provide data integration, through not as sophisticated as found in federation systems. Moreover, they require adapters to connect to legacy systems, as well as transformations to a common format for disparate data. Such transforms, of course, require function activation, and most portals have an application server bundled into their product offering.

Consider now workflow systems. In order to provide process flow, it is clearly necessary to provide adapters to popular legacy systems. Moreover, workflow requires the ability to execute logic. Hence, most compile into code for an application server or bundle one into their product offering.

Figure 5 summarizes this overlap in functionality among the various kinds of systems.

This overlap of product features appears to be increasing over time. For example, application server vendors, such as BEA, have added EAI capabilities to their products, thereby blurring the distinction between EAI and application servers. This movement will add adapters, messaging and transforms to this class of products. Portal vendors are extending their products beyond data integration and are talking about application integration. This will make them look more like EAI systems

Lastly, data federation systems will have no trouble implementing a messaging system in their products. [BERN90] shows the required message queue implementation. One need only have one application insert into a queue, and a second application read from the queue. Hence, EAI functionality can be efficiently simulated by a data federation system. As such, I expect the functionality matrix to be filled in over time with more yes answers.

There are three major problems with this state of affairs. The first one concerns duplication of effort. Since most major enterprises are running most, if not all, of these classes of products, they are running an example from each row in Figure 5. Since most products have their own transformation system, the enterprise must re-implement each transform several times. For example, if the enterprise needs a transform that converts French francs to US dollars, the chances are great that it will re-implement this transform multiple times.

functionality / system	function activation	transforms	adapters	application messaging	bulk transfer	distributed query	process flow
application server	yes						
EAI	yes	yes	yes	yes			yes
workflow	yes		yes				yes
ETL	yes	yes	yes		yes		yes
portal	yes	yes	yes			yes	
data federation system	yes	yes	yes		yes	yes	

Figure 5. Function Overlap

Of course, it would be great to have an enterprise-wide metadata and transform repository, but few enterprises have this capability. Also, few products are capable of importing transforms from such a repository. Lastly, most products have a proprietary transformation system and do not obey popular transform protocols, such as J2EE.

As a result, there is likely to be considerable duplicated effort.

A second problem is that each class of product has its own:

1) tuning considerations
2) memory management model
3) security model
4) debugging environment
5) development environment
6) crash recovery procedures

In effect, each product requires a system administrator. A potpourri of products makes such system administration more difficult and expensive.

A final problem is that this potpourri of systems tends to generate political fiefdoms, each with its own chosen technology. Moreover, with such overlap in functionality, there are multiple ways to implement any given task, leading to a more complex design task for users. Lastly, there are more kinds of moving parts for management to understand, leading to extra complexity.

In summary, there is duplication of effort and added complexity. The obvious solution to this situation is to have less moving parts. This is the topic of the next section.

4. Reduction of Middleware Components

In this section we present some possible reductions in this middleware chaos. These are ones that are technically logical; no attempt has been made to apply marketplace politics to this situation. Hence, marketplace acceptance of these ideas is unknown.

4.1 Data Federations Subsume ETL

As mentioned earlier, data federation systems offer a pure superset of the functionality of ETL systems. Hence, they should simply subsume ETL systems as a special case. For this subsumption to take place, something like the transformation development environments of some of the popular ETL products must get integrated into one or more data federation systems.

The result of this subsumption is that enterprises will have a uniform environment for physical warehouses (all the data in one place) and virtual warehouses (assemble the data on demand from the owner of the data). Physical warehouses are optimized for historical access, while virtual ones do best on operational data.

4.2 EAI Subsumes Workflow

It is clear that EAI is moving into process flow. There is clearly no need for two products in this area. It appears that EAI offers a broader vision than workflow, and thereby may be able to subsume this category.

4.3 Portals and App Servers Integrate Data Federation Systems

App servers typically allow a user to run transactions that span multiple sites, typically providing a two-phase commit protocol to guarantee data consistency across sites. However, the application program must know where the data resides and how to break down a cross-system task into single-site pieces. A valuable addition to such an app server would be a data federation system to extend app server capabilities.

A similar discussion applies to portal vendors. Again, the application developer must understand how to produce single-site data retrieval tasks. Again, a valuable addition would be a data federation system.

4.3 EAI and App Servers Converge

This is already starting to occur in some of the popular products.

5. Summary

In summary, I see consolidation of the various classes of middleware products into a much smaller set. This consolidation will occur using two principles.

First, broader products will subsume the functionality of narrower products. I expect EAI to subsume workflow and federations to subsume ETL as noted above. Second, I expect products in

various categories to be merged into “super products”. I expect that to occur in EAI, app servers and federation systems. Such super products will increasingly offer all required middleware functionality in a single integrated environment, requiring only one kind of middleware system administrator. Customers will benefit from this consolidation by having a simpler environment to administer and having a single place where transformations have to be written, registered and executed.

References

[Bern90]: Bernstein, P. et. Al., “Implementing Recoverable Queues”, *Proc. 1990 SIGMOD Annual Conference*, Atlantic City, N.J., May 1990.

[Bern96]: Bernstein, P. “Middleware: A Model for Distributed System Services,” *CACM*, Feb 1996.