eBay - eCommerce Platform
A case study in Scalability

by

Mohammad Usman Ahmed
mohammad.ahmed@mail.mcgill.ca

Table of Contents:

The Application and its overall architecture

EBay is a highly successful eCommerce platform. The larger category of eBay includes 19 different platforms (e.g. skype, payPal, rent) but we'll be focusing on the search and trade platform. The other sister platforms became part of eBay due to recent acquisitions and in some cases result in architectural mismatch which would be an interesting topic for a separate case study, therefore this case study focuses on the original platform's architecture and its evolution in recent years.

EBay is an eCommerce system where a user can browse to the website eBay .com and search for anything they want to buy, in auction or right away from the buyer, or to post some item for sale which other users can search for as prospective buyers. The users then arrange for payments online (using eBay's payPal system which is a separate system designed solely for that purpose and recently integrated onto the eBay platform) and receive the item by mail.

Like most internet-enabled business systems, eBay is constructed using distributed object technology. It requires scalability, high performance, high availability, and security. It needs to be able to handle large volumes of requests generated by the internet community and must be able to respond to these requests in a timely fashion.

In addition to the end-user application, eBay has now released its new APIs for developers to create buying applications, access eBay web services platform with javascript and flash, enable eBay bidding from anywhere, and create lightweight alerts about platform activity. This allows the developer community to customize eBay applications to their own needs and to release new useful features.

The most recent stats regarding eBay state that:

- It managers 248,000,000 registered users
- It manages over 1 Billion photos
- eBay has nearly 10,000 live applications
- eBay currently has 30 Software Architects in its employ
- eBay averages well over 1 billion page views per day
- The eBay platform handles 4.4 billion API calls per month
- 100,000+ lines of code are added every two weeks
- There are 30,000 software builds per week
- There are more than 44 billion SQL executions per day

These stats give an idea of the large scale of the eBay platform and the growth that has taken place in just a few years since the launch of this web application. The stats also indicate the key quality attribute requirements for this system to be its high availability, manageability, availability, and most of all its scalability. This case study will summarize its core architecture with special focus on its evolution to accomodate the scalability requirement which was not foreseen in its entirety at the time of its initial launch and had to be accomodated later on by a re-designing of the architecture.

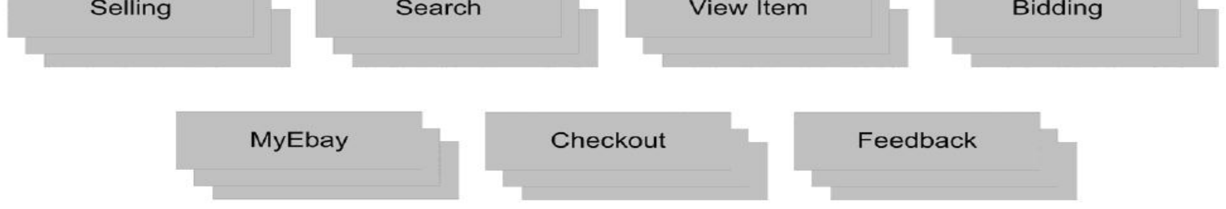


Fig1. An overview of the services provided by eBay

The architecture for a system of this size goes through many iterations. An architectural solution of this level is not only based on the software architecture but also on the system architecture, since the components that populate the system are not just web servers and web clients but it also has databases, security servers, application servers, proxy servers, and transaction servers. The system under discussion has a 3-tier architecture with a web-enabled device (a browser), application and transaction servers, and databases at the data services layer. The following diagram illustrates the relations between these components.

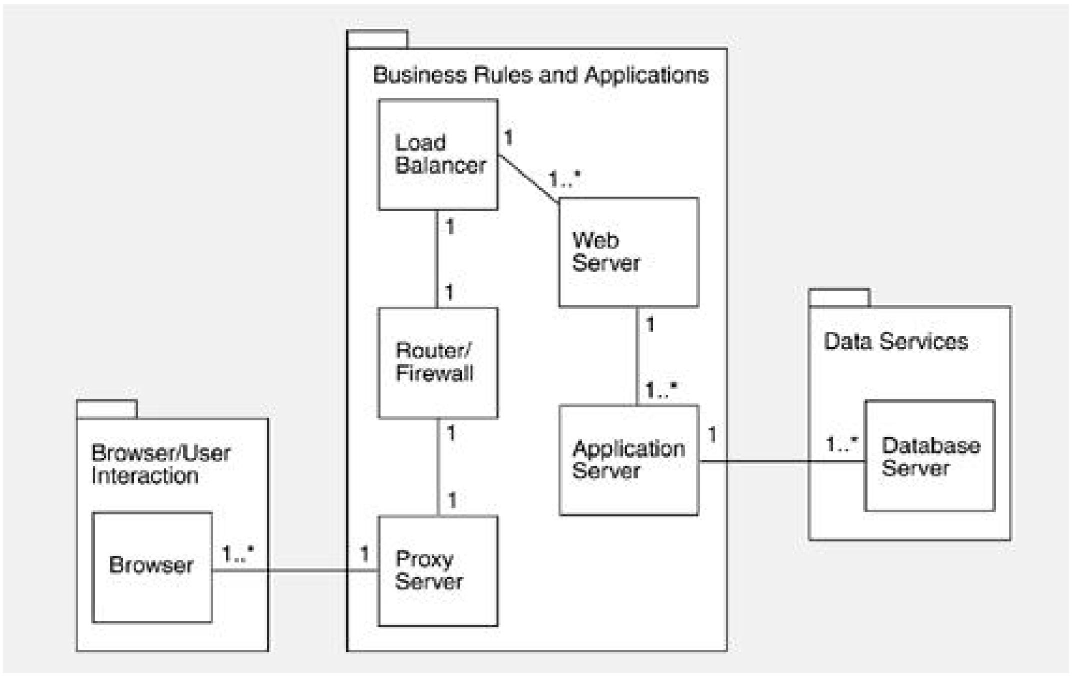


Fig2. Subsystems in the eBay Architecture

Component model and its interactions

Web servers, Web clients, databases, security servers, application servers, proxy servers, transaction servers.

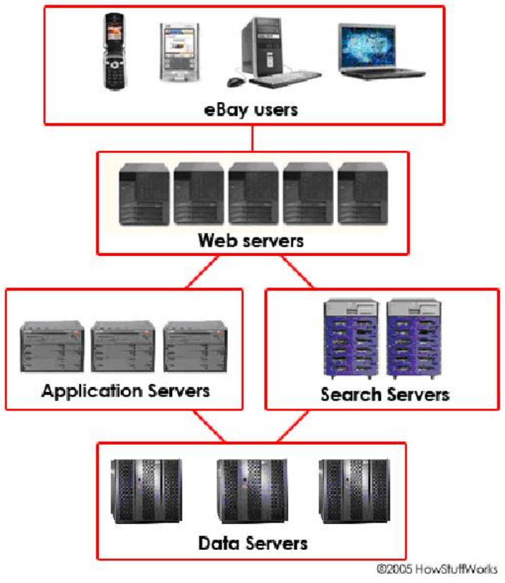


Fig3. Overview of the component interaction

Mapping of components to key quality attributes:

Web browsers for Modifiability
The end user typically interacts with the website through a web browser. Web browsers support user interface modifiability in a wide variety of ways, as the user interface that the browser supports is not hardwired but it is specified via HTML.

HTTPS for Security
Once the user has submitted a request, it must be transmitted to the target web site. This transmission may often be sensitive information such as credit card, and HTTPS (HTTP Secure) is used for this purpose. HTTPS uses Secure Sockets Layer as a subprotocol underneath HTTP. This level of encryption is adequate for the exchange of commercial

information involved in short transactions. The parallel system of payPal provides further secure transactions through the user's bank account for purchases at the marketplace.

Proxy servers for Performance
Requests from individual browsers may first arrive at a proxy server, which exists to improve the performance of the web-based system. The proxy servers cache frequently accessed web pages so that users may retrieve them without having to access the main web site. However, if a user chooses a particular item, with the intention of bidding or selling, then he/she must be shown realtime data. These proxy servers are typically located close to the users, often on the same network, thus saving a tremendous amount of communication and computation resources.

Load balancing for Performance, Scalability, and Availability
A load-balancing component is an integral part of the eBay website, because it supports performance, scalability, and availability. The job of the load balancer is to distribute the "load"—incoming HTTP and HTTPS requests—among a pool of computers running Web servers. The load balancer may simply (and transparently) redirect the request to another computer, or it may respond to the client and instruct it to redirect the request to a different server. This redirection is transparent to the end user, but it could result in an additional roundtrip of communication.
Since the load balancer is acting as a proxy for the pool of computers, we can add to that pool without changing any external interface, thus supporting performance scalability, specifically known as horizontal scaling (adding more instances of a given resource).
In addition, the load balancer may monitor the liveness of each of its computers and, if one of them goes down, simply redirects traffic to the others in the pool. In this way it supports availability.

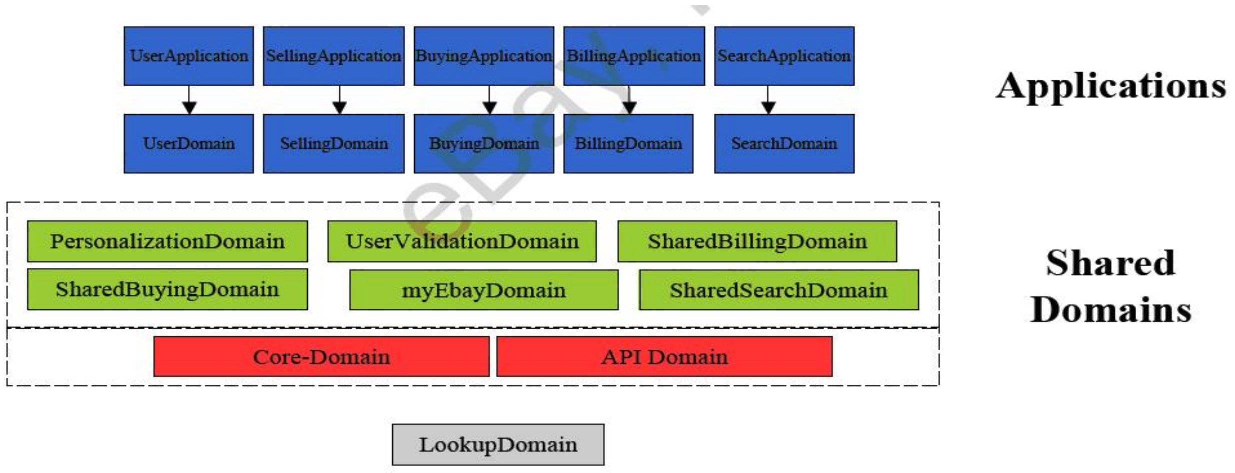
Web Servers for Performance
Next the HTTP or HTTPS request reaches the web server. The web servers are multithreaded, utilizing a pool of threads, each of which can be dispatched to handle an incoming request. A multithreaded server is less susceptible to bottlenecks (and hence long latency) when a number of long-running HTTP or HTTPS requests (such as credit card validations) arrive because other threads in the pool are still available to serve incoming requests. This introduces concurrency at the web server level.
Upon analyzing the request, the web server sends it to an application server that responds using the services of a database.

Application servers for Modifiability, Performance, and Scalability
From the web server the request is forwarded to an application server. These application servers run in the middle business rules and applications architecture as illustrated in the figure above. These servers implement business logic and connectivity, which dictate how clients and servers interact. This allows the databases to concentrate on the storage, retrieval, and analysis of data without worrying about precisely how that data will be used. eBay uses the J2EE application server framework. This replaced the single C++ ISAPI library that existed before and greatly increased the support for scalibility, around 2001.

Databases for Performance, Scalability, and Availability
Finally, the request for service arrives at the database, where it is converted into an instruction to add, modify, or retrieve information. Current database systems frequently use internal replication for performance, scalability, and high availability. They also use caching for faster performance.

Technological Aspects of the eBay Architecture

The databases are segmented into functional areas, for example user hosts, item hosts, account hosts, feedback hosts, transaction hosts, etc. This supports functional decoupling and isolation. There are no client side transactions. Single database transactions through anonymous PL/SQL blocks. There are no distributed transactions. It is all based on careful ordering of db operations, with asynchronous recovery events.
Although the J2EE framework is the bases of the Application Tier, the system uses only the basic functionality of J2EE and keeps the application tier cocmpletely stateless. There is no session state and all transient states are maintained in cookie or scratch database. The application tier is partitioned into the presentation, business and integration tiers.
eBay has built a software-based Integration Tier. This contains both a data access layer (DAL) and a services framework. The DAL is eBay's internally-developed pure Java Object-Relational (OR) mapping solution.
It leverages both component-oriented and service-oriented architecture technologies. eBay has built its own service architecture and uses it to enable integration across disparate technology stacks. An example of this is the open interoperation between C++ and Java technologies via services.
Within the Application Tier there is vertical code partitioning where applications (selling, buying,etc.) depend on domains but there is no interdependancy among shared domains.



Many EJB servers can coordinate transactions that involve multiple objects residing in various processes in a distributed system. If the distributed transaction implementation provided by the EJB server incurs additional remote calls in coordinating transactions, using distributed transactions can slow down an EJB system considerably, inhibiting overall system scalability. Therefore eBay had to achieve The application stack is a very core J2EE set of design patterns. Application objects are the equivalent of session beans, and the business object which are the equivalent of entity beans.scalability by developing their own solution for transactions.
This combined application and data architecture "creates the secret sauce for extreme scalability" for eBay.[4]
The main technologies used in other areas are

- MSXML framework for presentation layer (even in Java)
- Oracle databases, WebSphere Java (still 1.3.1)
- J2EE: use servlets, JDBC, connection pools.

Strengths and Relevant Weaknesses of the Architecture

In addition to the core marketplace services, the architecture is now also able to support external APIs for software developers to create their own software applications that leverage eBay Web Services.
These may be developed in any of the following languages and platforms:
JavaScript, JSON, AJAX, Flash/ActionScript, PHP, Perl, Python, .NET e.g. C#, ASP, VB, or Java, and JSP.
At the current size of the system, modifiability is there but its execution would not be as smooth as one would like. For example, for just adding a small cool widget, on a page that's doing 300 million page views a day, and that involves 5 database round trips, in order for the feature to become part of the system, it would be adding a potential 1.5 billion SQL executions a day to a database. A server that can do 600 million host a day, it would need 3 mid-tier boxes just so that the widget can be added on that page.[2]
Some of the key features of the design are

- Asynchronous event-driven design: Avoided as much as possible any synchronous interaction with the data or business logic tier. Instead, used an event-driven approach and workflow
- Partitioning/Shards: designed the data model so that it will fit the partitioning model
- Parallel execution: Parallel execution used to get the most out of the available resources. For processing users requests. In this case multiple instances of each service can take the requests from the messaging system and execute them in parallel.
- Moved the database to the background: Moved work out of the database into the applications because the database is the bottleneck so that database interactions happen in the background, thereby further supporting availability and performance, along with scalability.

It is really quite interesting how eBay enables data access scalability. The custom O-R mapping is a key feature that helps in that regard as mentioned in the previous section. Furthemore, they use bean managed transaction exclusively, autocommited to the database, and use the O-R mapping to route to different data sources.
The system does not make use of Entity Beans, instead using its own O-R mapping solution. The partitioning of application servers as well as the databases is all based on use-cases. Finally the stateless nature of the system and the conspicuous absence of clustering technologies are some of the key strengths of the architecture that have favoured scalability for eBay.

Component Model Variations

In one of his interviews, Dan Pritcher, one of the lead architects of eBay noted that since at the end of the day the only thing that matters ultimately is that we are able to meet the availability numbers at the transaction volumes we are facing, we should be willing to bend the rules of the architecture.
One of the ways in which it varied from the original design is removal of transactions.
It is interesting to note that EBay .com doesn't use transactions and a lot of it is driven by availability concerns for the majority of their use cases. There's critical data and there's non critical data, and the databases are split for these. This makes it responsive and also maintains site availability.
I present here, an interesting anecdote from Dan Pritcher's interview at the QCon in San Francisco which illustrates how variation in size of a system gives rise to new architectural

decisions that would otherwise not even be thought as possibly affecting the system architecture.
'We had a problem where we couldn't deliver power to our datacenters because municipalities can't even deliver the power infrastructure or build the power infrastructure to do that, which then drives you into problems of "how do I drive up utilization on boxes to the point that I'm not wasting, I don't have idle processors that are consuming power but not doing work for me?". An interesting architectural deployment and software engineering problem, if you asked me 2 years ago "are we going to be thinking about this problem in terms of how to maximize transactions per watt?" ' [2]

Key Quality Atrributes favoured by the eBay Architecture

The impact of quality attribute requirements, such as manageability, scalability, security, and availability, are radically increased when such an application is exposed to potentially limitless numbers of concurrent users on the internet. And even very straight forward problems become extremely hard at such massive scales.
The use of such an application is not evenly spread over time, but often peaks at certain times of the day, yet availability has to be aimed for all hours of the day.

The key quality attribute requirements for this system mentioned above are sumarized in the following table

Table 1. Application Quality Attribute Requirements

Quality	**Requirement**
Scalability	System should support variations in load without human intervention. As more and more people join the internet, more and more users attempt the access the website and request information simultaneously and they must be serviced without fail.
Availability/ Reliability	System should provide 24/7 availability with almost no downtime period.
Security	System should authenticate users and protect against unauthorized access to data. Such access could have disastrous effects for an online marketplace
Usability	Different users should be able to access different content in different categories and multiple users must be able to bid on items at the same time.
Performance	Users should be provided with fast response times. This is especially necessary when a bid is coming to a close and the user wants to ensure that he/she gets the chance to see the last bid.

Together the components described in the architecture section allow the e-commerce system to achieve its stringent quality goals of security, high availability, usability, modifiability, scalability, and high performance

Table 2. How the eBay Architecture Achieves Its Quality Goals

Quality	**How it is achieved in the eBay architecture**	**Tactics employed for reaching the goal**
Scalability	**Allows for horizontal scaling; split databases; load balancing**	**Abstract common services; adherence to defined protocols; introduced concurrency, esp. at data service level.**

Table 2. How the eBay Architecture Achieves Its Quality Goals

Quality	How it is achieved in the eBay architecture	Tactics employed for reaching the goal
Availability / Reliability	Redundant processors, networks, databases, and software; load balancing	Active redundancy; transactions; introduce concurrency. 99.94% availability was achieved
Security	Firewalls; A common user identity management system (Single Sign-On)	Limit access; integrity; limit exposure; Once user signed-on, secure connection maintained while user utilizes the site
Modifiability	Achieved by separation of browser functionality, database design, and business logic into distinct tiers; Use of J2EE	Abstract common services; semantic coherence; intermediary; interface stability; 30,000 lines of code changed every week
High Performance	Load balancing, network address translation, proxy servers	Introduce concurrency; increase resources; multiple copies

Violation of a key quality
Due to eBay's violation of their commitment to the quality attribute of Security, it had to suffer through a number of conspiracies and bad reputation as it drew scrutiny towards its failure at maintaining Security on the web site. Although eBay supports HTTPS for SSL connection, eBay users have the option to log in without using SSL, and the default is to use an insecure login. Even if customers log in using SSL, they are taken to non-SSL pages if they want to change their password or view account balances. This allowed a number of hackers to take over accounts of users with good reputations and then run away with cash after setting up fraudulent auctions.
It is also important to note that Scalability and Manageability are contradicting quality attributes in this case, as deploying a large number of servers and databases decreases the Manageability of the system.

Evolution of the application and its architecture

The architecture had to go through a number of iterations, as each time the demand for the application increased, they had to scale the application and found that the application didn't have enough support for scalability.
Around 1995 it started off without any search capability and just allowed category browsing. As it became more popular, they introduced search with a Microsoft index server and moved to the 3-tiered conceptual architecture with db access tiers (Oracle db introduced) but still had no application server, only a 2-tiered physical implementation. Everything was using a C++ Library running on IIS on Windows. Within 2 years, as the demand increased further, they had to introduce load balancing, servers were grouped into pools, search functionality was moved to a Thunderstone indexing system and the back-end Oracle database was scaled-up to a larger machine and a second database was added for failover. Even at this point the architecture did not support proper scalability as they were still attempting to fix the overload by scaling-vertically (using a larger machine) instead of scaling out (parallel distributed db with separated modules).
After two more years, in 2002, horizontal scalability was finally introduced through database splits and items were split by category. By this time, the application server was hitting compiler limits on number of methods per class and hundreds of developers were now working on the same code. So the application server was replaced by the J2EE application server framework for reusability and separation of duties, and they had to re-write the entire application in J2EE. From their on the journey in the architectural evolution was in the right direction until it reached its current state has has been discussed above.
This shows that in the early days of the internet, companies that were beginning to utilize it for creating a virtual marketplace could not perceive such an enormous growth in such short time nor could the technological advances and the develoment of the various frameworks, such as Java's enterprise edition and EJB have been predicted.

Conclusion

In analyzing the above discussion, we can see that there was no way for the architects of the eBay platform to foresee the rapid development that took place, in the expansion of the world wide web in particular, with millions of people gaining access to it, in just a few years, and also in the development of the frameworks and programming languages that would later be used to improve upon the architecture of the system. The original conception of the Web was as a web of documents, in keeping with its hypertext roots.[5] Whereas e-Commerce views the Web as a web of data, and these different views could not have been resolved completely in the early days of the launch of eBay. Therefore it was necessary that this platform went through this architectural drift/enhancement every few years and it is inconceivable that it could have been prevented entirely.

An interesting point I came across is that the current approaches to scalability introduce a huge level of complexity. Most architects and developers start with simpler approaches that are not scalable, knowing that later they will need to completely re-design the application to meet requirements. Scalability and time-to-market are perceived as two contradicting goals. Therefore most architects prefer not having to implement the scalability patterns on day one, but just being aware of what scalability means. As initially for the first launch, we would be required to make compromises to address time-to-market requirements, but at the same time we should we plan ahead for a change at the appropriate time. Perhaps eBay had to go through a similar phase and the re-design of the architecture for scalability at a later time was the intended plan all along.

Having noted that, we may also ponder on how much it still might have to change to accomodate the future influx of internet users, as more and more people gain ready access to the internet and the real marketplaces continue to move to the cyber domain. Perhaps, just like before, with the problems and issues arising later on, we might also see the database solutions and frameworks developing further to support the problems of the time to come as they did before.

Application workload is growing at an increasing pace, making scalability a prime concern of application designers and administrators and now there is trend starting to shift from tier-based architecture to space-based architecture. Some industry leaders are already commenting that "inherent scalability barriers represent a dead end for today's tier-based business-critical applications"[11]. They are now presenting applications that support *linear scalability*, claiming that the only way to cope with this growth is to switch from the tier-based model to a new architectural approach. In this approach, applications are partitioned into self-sufficient processing units, and Space-Based Architecture (SBA) is presented as the implementation for this approach. Whether SBA really is the solution for the next generation of limitations on application growth is yet to be seen.

References:

1. *eBay - http://www.ebay.com*
2. *Interview with eBay's Dan Pritchett - http://www.infoq.com/interviews/dan-pritchett-ebay-architecture*
3. *A talk by Randy Shoup and Dan Pritchett of eBay : Striking a balance between site stability, feature velocity, performance and cost - http://www.addsimplicity.com/downloads/eBaySDForum2006-11-29.pdf*
4. *Manageability - Nuggets of Wisdom from eBay's architecture - http://www.manageability.org/blog/stuff/about-ebays-architecture*
5. *WWW A Case study in Interoperability - http://proquest.safaribooksonline.com/0321154959*
6. *Web-based architectures - http://www.newt.com/wohler/articles/web-architecture-seminar.html*
7. *Service Oriented Architecture - http://blogs.zdnet.com/service-oriented/?p=675*
8. *EBay's new APIs - http://blog.programmableweb.com/2007/06/12/ebays-new-apis/*
9. *EBay's Java Developer center - http://developer.ebay.com/developercenter/java/*
10. *How eBay works - http://money.howstuffworks.com/ebay.htm*
11. *Xtreme Transaction Processing under SBA terminology - http://natishalom.typepad.com/nati_shaloms_blog/2007/08/xtreme-transact.html*

12. *The Scalability Revolution: From dead end to open road - http://www.infoq.com/vendorcontent/show.action?vcr=174*