Selecting a J2EE Vendor

by Simon Blake,
the esperanto Group, http://www.esperanto.org.nz

Version .92

Additions and constructive comments are welcome.

This document currently represents the views of one person,
sending me your ideas will make this a better document for all.

Please email me at sblake@esperanto.org.nz.

Table Of Contents

INTRODUCTION

The purpose of the paper

This paper attempts to act as a review guide for organisations doing a seeking a vendor that provides technologies centred around the J2EE brand created and maintained by Sun Microsystems. It attempts to act as a guide for those organisations who wish to use J2EE and want to decide between a number of different potential vendors.

This is a *technical* paper, and is aimed at *technical* people doing *technical* reviews. As such, I have tried to remove or at least point out where decisions would be made for political reasons. It is strongly recommended that technical reviews be wary of including political pressures or reasons and the matrix provided is skewed specifically towards preventing this. A technical review should be as thorough and effective as possible about the technical merits of a particular J2EE vendor's solution.

It is also recommended that your organisation shows the vendor your technical evaluation before the final decision is made - it is unfair to them and to your organisation if you withhold the evaluation, and it ensures that your impartiality and integrity is kept at its strongest level. Do not let them see other vendors technical evaluations as often this breaks contractual agreements for evaluation.

J2EE Specification Used

This paper is based on the J2EE Specification 1.2 Final Release, dated 12/17/1999.

J2EE Vendors

Many J2EE solutions exist, the EJB-SIG located at http://www.mgm-edv.de/ejbsig/bm_ejb.html provides a comprehensive list of those that at least support Enterprise Java Beans, a significant component of J2EE and this should be reviewed by an organisation before taking any further steps - to at least ensure that a decent representation of vendors is taken into consideration.

It is the opinion of this writer that most J2EE solutions are relatively immature - many organisations taking the option to write a product that concentrates on a J2EE product suite rather than addressing many typical customer needs. The J2EE specification, although recognising interoperability concerns, does not require a high level of interoperability with existing technologies and as such is of limited use. J2EE has to be combined with tried and true technology and only then does its true value show.

HOW AN ORGANISATION USES J2EE SOLUTIONS

An organisation that wishes to use a J2EE solution for a stand alone architectural component is unusual, but this is typically how many J2EE projects start off - typically an organisation will automatically choose to do a “single project” using their normal vendor of system software (IBM, BEA, Sybase, Oracle, etc) or with one that has the most market visibility in their area. But there are many different levels to a J2EE based solution and careful thought and considerable should be taken into account before making the final decision on which system to go with, typically a simple pilot done between two or three different vendors is useful if technical considerations need to be proven.

Most organisations who choose J2EE make the choice that their *default* architecture will be J2EE - this does not mean that they will not use Microsoft solutions or CORBA solutions - this just means that when they are requiring development themselves, or when the choice for a purchased product comes in a J2EE format, they are more likely to go with the J2EE solution. Most organisations would be committing IT suicide by not accepting any Microsoft or non-J2EE solutions in their organisations (at least at this point of the Java lifecycle) as there is a considerable number of well tried and tested solutions in such disciplines as ERP, SCM and CRM that do not use J2EE.

This need for integration is also crucial when deciding on what J2EE implementation to choose, and it is a important aspect of the decision making process to decide in the organisation whether interoperability is important for the solution. And this choice typically comes back to the architectural decisions of the organisation. An organisation may have decided for example to use Microsoft as its architecture of choice but must use a J2EE solution for one particular aspect as it is “best of breed” and the organisation wishes to maintain or ensure a competitive advantage.

An organisation that does software development for re-sale must also go through an evaluation process. Although J2EE specifies interoperability at a component level, that interoperability has problems - each vendors does certain things differently and a software vendor must choose one or a small number of platforms which they will support - the support costs of having more than a small number are very high - as they need to have staff that are knowledgeable about their own product as well as the appropriate J2EE product suites.

Choosing a J2EE solution then breaks itself into one of two main categories (there are of course many variations in between these two extreemes):

- **Standalone J2EE systems** - interoperability is not necessary (perhaps a Web application that “interoperates” through sharing a common database backend or via XML) or where isolation of the solution is important.
- **Infrastructure J2EE systems** - where J2EE is a decision for the organisation and interoperability at a level greater than simply database or XML level is important. Typically Remote Procedure Calls (RPC) (and sometimes Message Oriented Middleware (MOM)) mechanisms are important across applications. This organisation has a “buy-not-build” mentality.

MAKING THE DECISION

Developing Using J2EE

Perhaps the most important thing that an organisation can do before deciding on a J2EE vendor is to understand the J2EE paradigm - particularly the two key components, Enterprise Java Beans and the Java Message Service. Much of the rest of the J2EE specification is focused on infrastructure to support these two components.

Enterprise Java Beans

Enterprise Java Beans (EJB) provides an RPC (Remote Procedure Call) mechanism that allows the clients to request and expect a reply in real time for business functions.

Java Message Service

This service provides two key Message Oriented Middleware (MOM) capabilities - Publish/Subscribe and Queue features. These features are designed to be used when delivery of information is required, but not time critical.

For each of the following sections (Standalone and Infrastructure) we will list criterion that are important to each one. Because of cross over and the need to have a sensible reference structure, we will devote an entire section of this document to describing each of these criterion.

Standalone

For a Standalone J2EE system, the decision is typically less crucial - an organisation requiring J2EE is usually after it for component level interoperability and will expend some effort on isolating itself from the idiosyncrasies of the particular container. I typically refer to this as the choice of a “Web Application Server”. There are however a number of factors that should be taken into account when determining a vendor’s product for a Standalone J2EE project.

- Developer Productivity
- Distributed Application Management
- J2EE Compliance
- Load Balancing and Fail Over
- Organisational Stability
- Performance of Applications
- Vendor Stability

Infrastructure

Infrastructure decisions are considerably more complex than Standalone systems and a large number of factors need to be taken into account. Typically organisations such as this have a buy not build mentality and they will purchase in solutions of different kinds, with different platforms and different architectures. As such, interoperability becomes quite important - not just between different architectural infrastructures (such as J2EE vs COM+ for example) but also between different J2EE vendors. Important aspects of this are discussed in the Interoperability section (see “Interoperability” on page 11). I typically refer to this as a Enterprise Application Server and the following areas are of importance in determining this decision.

- Developer Productivity
- Distributed Application Management
- EJB Prohibitions
- Interoperability
- J2EE Compliance
- Load Balancing and Fail Over
- Modular Design
- Organisational Stability
- Performance of Applications
- Support for Common Distributed Problems
- Vendor Stability

All of these factors affect how well a particular J2EE solution will suit an organisation

ANALYSING THE CRITERION

Developer Productivity

Depending on the development methodology that is used, a development shop typically goes through the phases of design, develop, compile, debug, test with each cycle iterating back upon itself.

Ensuring that an organisation has chosen the best development tool - one that matches the J2EE environment, or one that provides enhanced capabilities for use within that environment is crucial. This document does not attempt to provide a review guide for J2EE development tools, but some guidelines are proposed.

General Development Guidelines

the more time the developer can spend inside the one tool, without having to swap around to different tools the better. The greater the number of steps required to develop, deploy, run and debug the application, the greater the chance that something will go awry during the cycle. Choosing a development tool that is capable of doing all of these things internal to the tool is very useful.

Debugging Guidelines

The J2EE specification does not specify how a J2EE solution is to be put together architecturally, but the more modular the design (see “Modular Design” on page 15 for more discussion on this), the easier it is for individual developers to run the crucial components that they are testing inside the development tool - for example, if the developer is debugging a series of Enterprise Java Beans, being able to run only the container (and none of the other services integral to J2EE) will make the debug cycle faster, easier and more likely to be possible on an individually specified machine (for example, a typical NT/2000 machine with 256Mb of RAM). Because there are many elements to a J2EE application (the EJB Container is but one of a number of things - including a Transaction Service, Security, JSP/Servlet engine, Web engine, Naming Service, and Java Message Service) modular design for use

when debugging becomes important.

Many organisations choose J2EE Application Servers that require remote debugging - which is a distinctly less desirable option for this kind of work because it involves so many steps. A developer can save 30-50% of debugging time just by being able to run the required components inside the development tool. Choosing a development tool that supports this, and a vendor that supports this is wise.

Distributed Application Management

When an organisation undertakes to develop or purchase a J2EE application they have a basic understanding the the purpose of doing so is that they will make money - either through increased revenue or cost savings.

A brief overview of the management marketplace looks like this:

Network Management

Network Management, based primarily around SNMP provides a mechanism to manage a series of physical products based on IP address. An SNMP network looks very much like a distributed database with the primary key being the IP address. Tools in this area are very sophisicated - they have been around for a long time, and often a tool can be switched on, left to run and an hour later you can come back and have a complete diagram of your network.

Systems Management

Systems Management (tools such as Unicenter TNG, HP Open View, Tivoli, Patrol, etc) go one step further and recognise that some of these network devices are computers and that they in fact have processes running on them, and that those processes should be able to be managed. Typically such Systems Management tools will also recognise certain processes as being extra special - Patrol for example will understand that a process is part of Oracle and can actually manage “Oracle” as a set of processes rather than just a single process. These products are also very sophisticated and often cross over into the Network Management sphere. A typical user interface for a Systems Management tool will show processes in various colours on various machines running or not running. Systems Management has worked well with 1 and 2-tier applications as typically they can manage a process “blob” - people can tell that the database is working because Oracle is going for example.

Unfortunately, when it comes to three tier applications, things have changed slightly. Modern n-tier applications have a number of key differences from product suites of the past - typically each different “generation” of product has had its own individual problems. We have evolved from 1-tier applications (green screen applications on the mainframe with all logic, presentation, business and database logic in the same application) to 2-tier applications (where business logic was spread somewhere between the “fat” client and the database server) to 2 1/2 tier (with typical web applications that do direct database access) to full three tier applications using business objects.

One key aspect of three tier development, which is assumed given development of J2EE applications is that n-tier applications consist of ***distributed business objects***.

Applications Management

Applications Management is the newest in the management tools area and the tools are fairly unsophicated (with perhaps one or two exceptions). With Applications Management, you need to be able to deploy, configure, start, stop, monitor, tune, upgrade and maintain a distributed application consisting of many different kinds of processes, each which may have certain configurations of business objects operating within certain kinds of parameters inside them. Distributed Appplications Management tools must be able to manage down to the object level to ensure that the application is running with the required performance level.

Given that a J2EE application most certainly fits into the Applications Management area (rather than Systems or Network Management), it is important that a Management tool be looked at when choosing your J2EE solution of choice. If you don't, you'll have to write it yourself, and that can take a huge amount of time and is typically something that you will do for each application - a waste of time when the development team should be attacking that backlog.

Typically when J2EE is selected as the infrastructure, the Applications Management tool must also be able to manage other components that are not related to J2EE - such as COM+ objects or legacy systems. Often such tools will come with APIs that will enable this, but it is worthwhile an organisation doing the research into just how customisable such a tool is if it is to fit into their organisation effectively.

Developers and Architects typically forget the applications once it is in the hands of operations - and it is crucial to the success of J2EE projects that this does not happen. If you cannot manage it then it will not work and the project will fail.

One final note in this area - a vendor's tool that allows you to watch what is going on from one EJB server, or even a few EJB servers is not, in any way, a Distributed Application Management facility. Cool consoles do not make the grade - they are not something that an operator is going to use.

EJB Prohibitions

There are certain things that EJBs are not allowed to do, as per the specification. The intention is to keep EJBs portable and you should beware of vendors that cannot provide you the ability to do these things, yet still stay compliant with the specification.

For CORBA vendors, these things are wrapped up as CORBA objects and exposed in that manner - thus not breaking the EJB prohibitions and yet maintaining interoperability for transactions and security.

The main areas of prohibition that are of particular interest to this document are as follows:

- EJBs are not allowed to access native libraries. This means that if you have a C or C++ set of libraries that is your only interface to a specific product then you must find another way. This is done for portability, stability and security reasons.
- EJBs are not allowed to display any GUI. This prevents EJBs from not being portable across to platforms that have no GUI interface (such as the AS/400 or simi-

lar)
- You cannot access files or directories with the java.io packages. This is quite a brutal requirement, and is specified to encourage the developer to use a database to store and retrieve data from, and it could compromise security
- You cannot listen on a socket - this would artificially keep the EJB active. This also prevents JMS from operating effectively in EJB 1.1.

Typically, a Standalone system would not require such interfaces - but an Infrastructure solution would have problems in most of the listed points. Ask your vendor what solution they have for making sure that any of the things that you wish to do that are prohibited can be done without compromise of the EJB standard.

Interoperability

Interoperability is a large topic, as interoperability exists on many levels. Interoperability is also perhaps the most enthusiastically debated area of EJB, which is perhaps rightly so, as the more experienced distributed systems developers understand that a J2EE system more often than not cannot exist in a vacuum inside an organisation. A J2EE solution must interoperate if it is in the Infrastructure category and the mechanism specified by the J2EE specification for interoperability is to use the CORBA infrastructure. This is not unusual because CORBA's main goal is interoperability - CORBA is platform neutral (like J2EE) and language neutral (unlike J2EE).

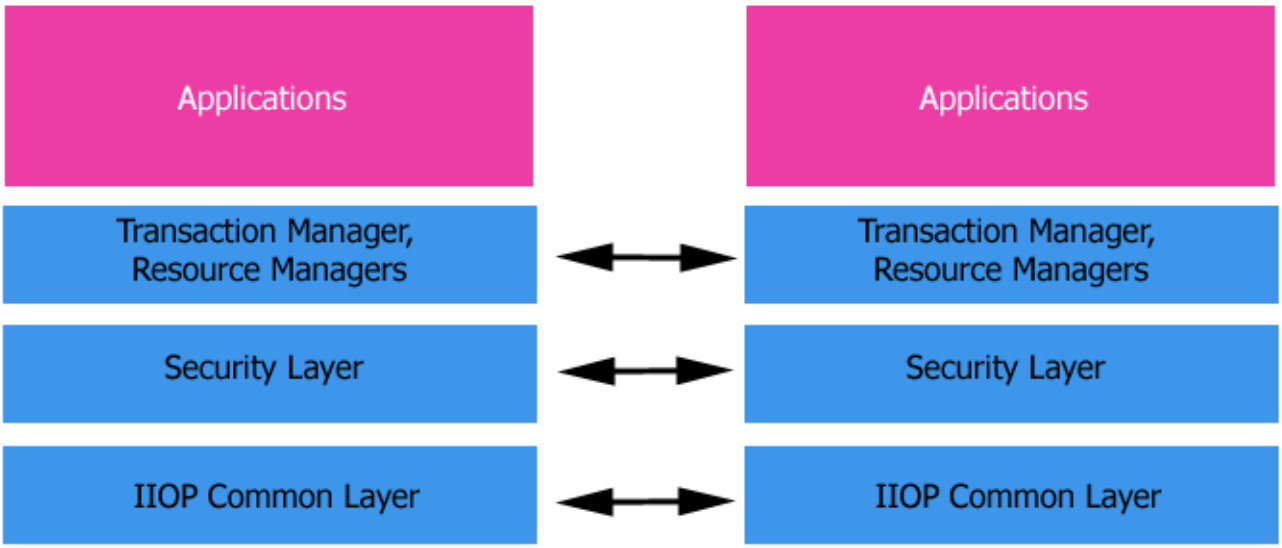
J2EE Interoperability Requirements

J2EE only specifies interoperability at the RMI/IIOP level. To achieve this given the CORBA specifications and given the J2EE specifications, a vendor must support the CORBA 2.3 specification - so ensure that your vendors does (fully). The RMI/IIOP provides method call interoperability, it does not specify or require interoperability at the Security or Transactional level.

Is That All?

Is this important? Lets have a closer look at this topic and see whether your organisation will need this. Below is a picture detailing how a typical distributed application layering

mechanism works.



The diagram shows that at the bottom three layer there exists a requirement to talk. This makes immediate sense in terms of RMI/IIOP - when one set of objects wishes to talk to another then IIOP provides the mechanism by which that occurs. The problem occurs when passing the security and transaction contexts.

What are Security and Transaction Contexts?

The security and transaction contexts are (typically) implicit buckets of data that get tagged on as requests are passed around from object to object. The purpose of the security context is to hold the identity of the user who is making the call - to ensure that that user has the ability to do what they are asking to do. The purpose of the transaction context is to ensure that all transactional resources (databases, queues, distributed objects that require transactional notification (as stateful session beans can), etc) can get "enrolled" in the transaction by the transaction manager.

The problem comes when moving between two different vendors J2EE solutions. If this occurs, then you cannot gaurantee that (for example) only one transaction manager will be involved. Typically Security Managers and Transaction Managers (called Services in CORBA) need to be able to talk to each other to achieve the required goals (particularly true of Transaction Managers). Conversation occurs for Security and Transactions "above" the normal method calls that you will make in your code to ensure that everything works.

If these services are not interoperable then your application will fail.

There are therefore three different levels of interoperability to consider:

- Ensuring call level interoperability using RMI/IIOP and that the vendor is using a CORBA 2.3 implementation. This is mandated for J2EE (except in EJB 1.1, but will be in 2.0).
- If you wish for your transactions to be interoperable, ensure that your vendor has an implementation of CORBA OTS (Object Transaction Service). If this is not crit-

ical to you now but you know it will become so, ensure that your vendor has stated that they will be doing an implementation of it and request the timeline (it is unlikely you will get an accurate one, but at least this is possible).
- If you wish for your security to be interoperable, then ensure that your vendor has an implementation of the CORBA Security Service. Query your vendor in the same manner as above.

Furthermore, both transactions and security are important when interfacing with other languages that are not Java. For a Java application calling a C++ CORBA service - for the C++ CORBA service to be able to know (a) what transaction is being used and (b) who the user is that is requesting the service, these CORBA services must be used.

CORBA Security and J2EE

One note about CORBA Security and J2EE - as of writing, the mechanism by which J2EE implementations interoperate - i.e. how they pass around the required J2EE information, or what information they put in that "bucket" is as yet undefined. The CORBA working group is (given a discussion on EJB-INTEREST) supposed to have the decision completed by August 2000. Ensure the vendor is following this as it affects interoperability for the vendors product.

Interoperability with COM+

Typically it is important to test interoperability with COM+. COM+ uses a very different model to that of J2EE - transactions and security will not interoperate with a normal solution. There exist COM/CORBA bridges that provides this level of functionality, as well as a number of strong Java products which enable the calling of COM+ business logic (but without the Transaction and Security capabilities). Determine what you need in this area and ask your vendors what solutions they are able to provide you.

J2EE Compliance

J2EE is a specific set of requirements, all technologies must be supported. The J2EE specification does not specify how an Application Server supporting these technologies must operate, but beware of fine print. There are a number of products that have claimed J2EE compliance but had one or two features missing - often such disclaimers are glossed over by sales people and are included only in the fine print.

An honest answer from a sales person may not prevent the decision to go with a particular vendor - not having Container Managed Persistence for Entity Beans for example may not be a requirement of your organisation if you have chosen to use EJB like a glorified CORBA implementation and just use stateless session beans. But knowing is important - not having (for example) Distributed Transaction capability, or even a limited Distributed Transaction capability (as many vendors provide) is quite important.

If a vendor for example indicates that they fully support RMI/IIOP, make sure that they support it for outgoing calls and incoming calls - when vendors make a representation to you, get it in writing if it is important - test it if you have the time.

Another factor in determining a vendor for your J2EE needs is to see how quickly they comply with J2EE specifications - are they 8-12 months late? Specifications do not appear

out of the blue, they go through a process whereby J2EE vendors get to particupate in a "closed" pre-draft phase, so they know what is coming up. Much of the technology is not a hugely moving target and so a vendor typically has upwards of six months before the new technology becomes agreed upon. Vendors who are quick with new technologies tend to leap frog each other in quick succession and then come into par for a while, and then play the leap frog game again. If this is the case, J2EE compliance is probably not an issue for them - you aren't going to have to harass them for support for the new standards, the marketplace is ensuring that they do it.

Some vendors will say that the can't use the J2EE brand because they aren't willing to pay the license to Sun. It is worthwhile if you can get the organisation to formally make this representation - and you may wish to check it out with Sun that they have not failed in their J2EE attempt if you can even get that information. [do Sun still do this?]

One final word - the products history. What is the history of releases of this product? Check out the support newsgroups for each of the products and see what kind of support they offer. See what kinds of questions are being asked and what the general feeling is about the product. One area that is worthwhile to go to is the EJB-INTEREST list at http://archives.java.sun.com/cgi-bin/wa?A0=ejb-interest and have a close look at the answers when people suggest vendors to look at.

Load Balancing and Fail Over

A crucial area that is ignored by the specification is the question relating to how scalable and reliable your vendor is. Typically this appears in terms terms of load balancing and fail over.

Load Balancing

This determines how sophisticated your vendor's load balancing capability is. Many vendors don't provide this capability, or the capability has to be hard coded - question your vendor to determine whether load balancing is dynamic - is it affected by actual load on machines? If machines and containers are added and removed from the network does the system automatically re-balance? Does this only occur at the EJB container level or do each of the services of the vendor (Transactions, EJB, JMS, Naming, Security) load balance?

Fail Over

Fail Over determines how a vendors solution will perform when certain components are removed - either through a crash or a specific shutting down. Ask your vendor questions like - how can I cluster my services, do they have to be homogenous or can they be hetrogenous? Can I fail over individual services or do whole "servers" full of capability have to be failed over? i.e. what level of granularity do you get? Does your load balancing go all the way down to bean level for instance? Do stateful session beans (if you are using them) fail over? How do they perform if this mechanism is in place?

As each vendor does this differently, it will prove to be a task to answer these questions - the matrix provides the above questions on the second sheet.

Modular Design

A modular designed determines how componentised the solution that you are evaluating is. The J2EE specification states that the following components are required to have a J2EE server:

- Support for HTTP and HTTPS - typically a web server
- Support for the Java Transaction API
- Support for RMI-IIOP (although this is not required as part of EJB)
- A Java IDL Compiler (again, not required as part of EJB)
- JDBC Support - the ability to hand out connections to databases
- The Java Message Service (JMS) - the MOM equivalent of EJB
- An Enterprise Java Beans 1.1 Server/Container
- Support for the Java Naming and Directory Interface
- Support for the JavaMail API
- Support for the JavaBeans Activation Framework (JAF) - which is used by JavaMail

Being able to run individual components of each on different machines dramatically increases the flexibility of your application. If you require a Standalone system, then Modular Design is probably not important, but in a typical Infrastructure system, you may have a number of web servers, appropriately placed EJB containers and JMS servers, one or two Transaction Services, and, depending on your requirements, a scattering of other CORBA objects all participating in the one distributed application. These resources could be partially or fully shared by other distributed applications as they are developed or purchased (see also “Distributed Application Management” on page 9).

Also important is the ability to co-locate certain types of services with each other. Co-locating is when different services are running in the same process space, so method calls are effectively in-process, and therefore faster than if they were going across a network. A properly componentised system will allow components or modules to run co-located if you wish. For a CORBA based solution, this is usually the case, but make sure that it is if that is important to you - as CORBA is able to provide some solutions to certain problems that EJB and JMS on their own cannot (see “Support for Common Distributed Systems Problems” on page 16).

Organisational Stability

The stability of your own organisation should certainly factor into the decision on which vendor to choose. In the corporate marketplace, mergers and re-organisations, centralisation and decentralisation, government regulations, and (if a government department) physical splits into different organisations are typically possible. Take stock of what is going on in your organisation.

If there are many changes taking place in your organisation, then an Infrastructure solution is more likely to net you benefits as many of the issues listed here are going to arise. Mergers typically bring systems into contact with each other that otherwise wouldn’t - if the merger is with a similar company the systems could be similar and the migration at a component level is important - typically some of the functionality from one J2EE solution will be moved across to another J2EE solution (this is of course a simplified view of what could

potentially happen). If the merger is with different (but complementary) companies, then typically systems will just want to interoperate with each other at some level, typically at an RPC level, and this interoperability issues (see “Interoperability” on page 11) become important - not having chosen a vendor that accepts a wider view of the world will cause either your organisation or the one that you are merging with unnecessary grief.

Performance of Applications

J2EE application servers typically encompass many, many hours of work by very talented people - but performance is a sticky question for most vendors.

There have been a number of independent collections of people (as well as vendors) publish benchmarks - but typically these have been done without the vendors permission and on evaluation versions. Most vendors (all that I have seen) specifically state that you agree, when using the evaluation version, not to publish any benchmarks without written permission from the vendor - and this will rarely happen if the vendor doesn’t win the benchmarks. Thus you do not see those publications appear on the web for very long, one of the vendors gets (often rightly so) in contact with the publisher and the article is removed.

Unfortunately this means that performance of J2EE solutions is difficult to ascertain - Sun and a Java Community Process Working Group are working on an Electronic Commerce Performance test (ECperf) which we should see available within the next year which may make this less difficult.

There are a number of rules of thumb that can be applied for most J2EE solutions though, and it is worthwhile for everything but “toy” solutions, or for personal experience you steer away from the following performance deprecating areas:

- RMI/JRMP - This is a slow, cumbersome protocol and it is the default protocol used by RMI. One set of research put the speed at 42% slower than IIOP.
- Immature Co-location - Immature J2EE solutions typically do not expedite calls between co-located objects. This can dramatically slow down your application and it is worthwhile to understand whether or not the vendor has optimised this. This is typical when the vendor has come from a Java background rather than a distributed systems background and has no real underlying infrastructure.

Furthermore, where performance is concerned, it may be that your application requires excellent performance in certain areas and in many other areas of the specification you just don’t care about. You need to weigh up each of the areas of J2EE and determine whether or not each one of them is important to you and only compare those that are.

Support for Common Distributed Systems Problems

JMS

There are certain problems that are far outside the realm of JMS capability. It is important to understand both of these technologies and the limits of them, as well as what kinds of capabilities exist in “other” such products in their area - do not limit what you want to do

to what you necessarily can do with a particular set of vendors, another vendor not on your list may be able to solve your problem. This is particularly true of JMS - JMS provides a simple Queue and Publish/Subscribe mechanism - there are considerably more sophisticated MOM solutions in the marketplace - and they may be of more use. MOM is typically used in Enterprise Application Integration as well, so if EAI is an area of concern to you, look more closely at the tools in this area. If you can get away with the feature set provided by JMS, it is worthwhile to use it as you will be more portable between vendors.

EJB - The New Visual Basic?

As mentioned, Enterprise Java Beans does not provide everything that a typical distributed systems developer will need. EJB could be called the “Visual Basic of Distributed Systems” - which is perhaps a little too cruel, but the intent of the phrase is to indicate two primary areas of concern with “EJB”

1. It introduces non-distributed systems developers to developing n-tier applications. Suddenly n-tier application development is easy enough that every develop who knows a little Java can dip their toes. This is perhaps dangerous if they get into positions of architectural influence and don’t upskill themselves first.
2. There are many solutions to different kinds of problems in distributed systems that distributed systems “old-timers” have come across and understand. A typical CORBA or DCE developer has had to put a much greater amount of thought into their design than EJB, as EJB solves a large number of problems.

What to do?

Given this there are a number of areas that EJB won’t solve for you, and where you will have to look to other technologies - CORBA being the most obvious, but Sockets are possible for situations where CORBA is not available (but remember Transactional Integrity and Security facets of your solution).

Such things include

- Singleton patterns - when you want to ensure that only one instance of an object service exists on your network, where all “things” have to pass through it (for example - controlling access to a resource that has a single thread library, maybe a logging service, etc), EJB cannot help you. You must instead use another mechanism.
- others?

Vendor Stability

There are many J2EE (or at least EJB) vendors. Knowing which ones to even take seriously is important - typically it is the tradeoff between the dynamic (but potentially flighty, not well grounded vendor) and the (perceived, if not actual) established player. Other organisations have “bought” into J2EE and tried to retrofit it with their own suite of products, sometimes to great success, sometimes to great confusion of both their existing customers and their new customers. This may not have happened with just J2EE technology, it could have happened earlier with non-J2EE technology which has been bought in, modified, abandoned, merged, changed, more bought in that merged and changed, and so forth. Typically larger vendors do this all of the time.

If your vendors touts “extra” features over and above J2EE, look at their track record for supporting these products. It may be perfectly well that such products time of life has been and gone, but it always leaves lingering dissatisfaction with certain customers - beware of a few creating a large noise where the majority are happy, yet silent. Also beware the other way around!

Look at the marketplace that you are in - your industry or your geographical area, most countries are not the size of the US when it comes to market and so finding another customer in your country *in your industry* that is willing to talk to you about their success or failures with the vendor’s products is unlikely. There are some general questions you can ask any organisation using the vendor’s product though:

- Is this a buy and forget organisation - do they sell you the product and then forget you until its maintenance time or they have other products to sell? Do their other customers see and/or talk to them very often?
- What is the support like from this organisation? Can you purchase the level of support that you require and do you know that it will actually be delivered? How much dependency is there on third parties in this vendor’s offering which may affect the ability to provide such support? Is there a risk of a schism with third parties which will affect your product choice?
- When being pointed to customers who have made a specific vendor choice that was a year or more ago (things change quickly in this industry!) - ask the customer whether they would have made the *same* choice of vendor given the choice of technologies available today? Do they continue to make this vendor’s choice because they are sufficiently locked into this vendor? Is this the way you want to be? Are you happy with a certain level of vendor lockin?
- How long has this organisation been in the distributed systems business? Do they have consultants or technical staff who can train you and point out to you the difficulties you may encounter, and what works and what doesn’t? If they have done training courses for other organisations, perhaps ask to talk to one of them and ask them for feedback on what the courses were like, what the level of experience is like from the vendor. Often organisations that haven’t been around long or who are light on the ground in terms of experienced staff show up very quickly.

The last factor that I will mention, but which should be more of a business decision and it should therefore be left out of any technical decision is how stable the vendor is financially. How are they doing with their revenues - are they a “dot com” company that just makes loss after loss, or are they making a proper set of revenues - or at least intent on doing so. The key question you want to know is “Is the vendor I choose going to be around in three to five years time?”. I have only included this because it is often an issue under consideration, but it is also generally left outside of the technical evaluation.

WEIGHTING THE FACTORS

Working With Vendors

Vendors will typically work at a number of levels within your organisation - expect this. Vendors will want to work with the IT manager and business people, but they should also assign a technical person to be at your call when you need to know certain information - especially if you wish to prove certain things. Make sure you have this resource available to you - if you don't, then you know that you don't have vendor committment and it is a bad sign of things to come.

Working with Criterion

Although this document does not provide a nice matrix that you can tick off, it does provide (in the ZIP file this PDF should come in) a Excel spreadsheet that contains most of the questions that have been asked - certain questions cannot be answered easily by a weighting factor and so are not included. Because the evaluation should go to a vendor for review, then you should keep such things as Organisational Stability (yours) to yourself.

I have also highlighted key areas that I believe are important in any organisational choice - particularly the question of JDBC 2.0 XA extension support within any J2EE server - without this there is no two phase commit support and therefore load balancing and fail over are meaningless.

The method I would suggest using is to assign a value to each of the points in terms of importance and breadth of the area - Starting your Distributed Application is important, but it doesn't have the same breadth as Load Balancing Support adequate.

SUMMARY

Deciding which J2EE vendor to use for your organisation can be a long, involved task. It can be made much easier by chucking out those vendors early on that don't match with your organisational vision - and this document has given you some mechanisms by which to recommend that. Look for key issues first and drop those vendors based purely on those areas. If, for example, J2EE Compliance is a very real and important issue for you, then don't accept a vendor that still only supports EJB 1.0 or only parts of EJB 1.1 for example. If interoperability is important to you, don't accept a vendor that isn't based on CORBA. If managing your deployed distributed application is important to you, ask your vendor what they have in this area or at least what products do they work with.