



WHITESTEIN
Technologies

Whitepaper

Agents in a J2EE World

by Stefan Brantschen & Thomas Haas

v1.4
2002-03-13

Whitestein Technologies AG | Poststrasse 22 | CH-6300 Zug
Tel. +41 (41) 711-1566 | Fax +41 (41) 711-1560 | <http://www.whitestein.com>

Copyright © 2002 by Whitestein Technologies AG
All rights reserved.



Introduction

At the Threshold These days, software agent technologies are right at the threshold between research and wider industrial use. Surveys and conferences¹ show that only few successful real-world applications using software agent technologies have been implemented on a larger scale – quite comparable to object-oriented technologies in the late eighties and early nineties.

Object Technologies Object (-oriented) technologies did not really take off until robust platforms, tools, and methodologies became available, useful “in the trenches” of the day-to-day IT-business. In this respect, the whole Java movement resulted in a major driving force in the middle of the nineties.

No market and customer oriented person wanted to use object-oriented technologies simply for their beauty, hence the finally available robust and performant technologies gave object-orientation its chance to show and prove its advantages. It's not that object-oriented concepts would have suddenly dramatically improved by then – but the matured technology enabled to make use of long-known concepts to build better systems and applications.

Agent Technologies In the real IT-world, also software agent technologies are not, and will not be, used for their sheer beauty. What is needed are better solutions for real problems, solutions that provide customer value, and generate business revenue. So the common understanding is that one should not “sell” agent technologies, but better solutions based on agent technologies.

True, but alas, for making this possible, also agent technologies need their stable and high-performance platforms, tools, and methodologies in order to show and prove their advantages.

And, as we know from many years of practice and experience with our customers, maybe the *managers* and *marketers* do not want (or need) to know that agents are doing their work within a system or application, but the *IT people definitely do want to know*, and “autonomous and self-configuring software components” somehow have the potential to send a shiver down the spine of system administrators – unless we can build an initial and then growing level of trust in this new technology, just as with any new technology.

As experience shows, this trust is much easier built if there is a sound basis of known technologies and proven products, if possible from well-known vendors, including migration options. These days, after the time period of hypes and back to reality of normal and tough business, to build the needed level of trust is nearly impossible, if everything presented is new, made by small companies or university labs, or even “smells like research.”

And, last but not least, also the providers of agent-based solutions want to build upon proven technologies and products, simply because they need

1 For example: <http://www.agentlink.org/agents-london/>



stable tools to provide professional systems that fit their customers' needs, today and tomorrow, within the project budgets and time frames.

Consequently, an important question is: how can we build agent-based business systems and applications using current state-of-the-art, proven technologies and products? This article describes an approach to build an agent platform based on J2EE application servers.

Challenges

Like all new IT concepts and technologies, software agents are being confronted with the following challenges of the "day-to-day IT reality," which are (or can be) even amplified by the current economic situation, which has become much less friendly for visions and experiments compared to only a few years ago.

Mission-Critical Business Systems

There is a base of installed systems, many of them under heavy load and mission-critical for our customers' operational business, and thus revenue. Often, we find complex networks of well-crafted and configured applications, which work well "as they are," and any experiments with new technologies and products are thus not really welcome.

No "Green Field" Approach

The development of new business systems to provide additional functionality cannot start from scratch. Existing systems have grown over many years, and can usually not just be replaced – in contrary, they must survive longer than ever these days.

Corporate Strategies

Larger companies have longer-term IT strategies, which define and confine a set of approved products and technologies, including also the related industry standards. From a point of view of cost of evaluation and purchasing, support, and maintenance, this makes a lot of sense. However, this makes it also much harder for new types of concepts, technologies, and products to "break into" a company's IT department.

Corporate Know-how and Experience

Companies usually have "accumulated" quite some know-how and experience how to run, administer, and maintain their IT systems and infrastructure in order to support the daily operational business. Each new technology, or even paradigm, will require some changes in people's mindset and in the organization's procedures, tools, and configurations. In the past, companies have experienced this to be a usually costly and longer-term process, often also plagued by disruptive technical or administrative problems. Hence, these days, IT departments have become pretty reluctant to easily jump on a new bandwagon. And with the current challenges – like on-going migration towards web-enabled and messaging-integrated applications, new concepts like web services, and new versions of already well-known products (platforms, databases, messaging systems, etc.) – to be mastered as first priority, IT managers are even more reluctant



to embrace a really new and not yet established technology like software agents – their eyes firmly on the support of the daily operations. IT has been bashed too long for not being able to deliver (remember the “software crisis”?!)

Large Investments Last but not least, companies have made large investments in today’s and yesterday’s technologies and products, both regarding money and efforts. These investments must simply be protected, from an entrepreneurial point of view.

Expectations and Basic Requirements Summarizing, professional IT managers, system administrators, developers, and other IT guys have strong opinions of what they want to expect from a technology that claims to be – or become – one of the keys for future business systems and network infrastructures. They are usually not anymore willing to put up with products that still show an “infancy state,” and the related problems for their daily business.

Thus, for professional business applications, industry-grade technologies must provide

- ❑ basic product *quality*, in particular regarding reliability and failover, performance and scalability, security;
- ❑ business *functionality*, eg. transactions, sessions, persistence;
- ❑ system admin *functionality*, eg. configuration and diagnostic tools;
- ❑ *migration* and *extension* capabilities of existing operational systems;
- ❑ *integration* with, and *interfaces* to, currently operational systems, products, and solutions (both based on today’s and still also yesterday’s technologies!)
- ❑ *re-use* of people’s *know-how* and *experience*, and organizational and administrative procedures and tools;
- ❑ adherence to industry *standards*.

J2EE EJB application servers, described below, represent a state-of-the art technology that fits the above requirements and expectations.

J2EE Application Servers

Introduction “Application servers” are a relatively new type of software, which has recently evolved to manage the complexities associated with developing business systems in today’s Internet world. However, the term “application server” is not precisely defined, and application servers can be made up of some combination of several different technologies, including web servers, ORBs, message-oriented middleware (MOM), databases, etc.



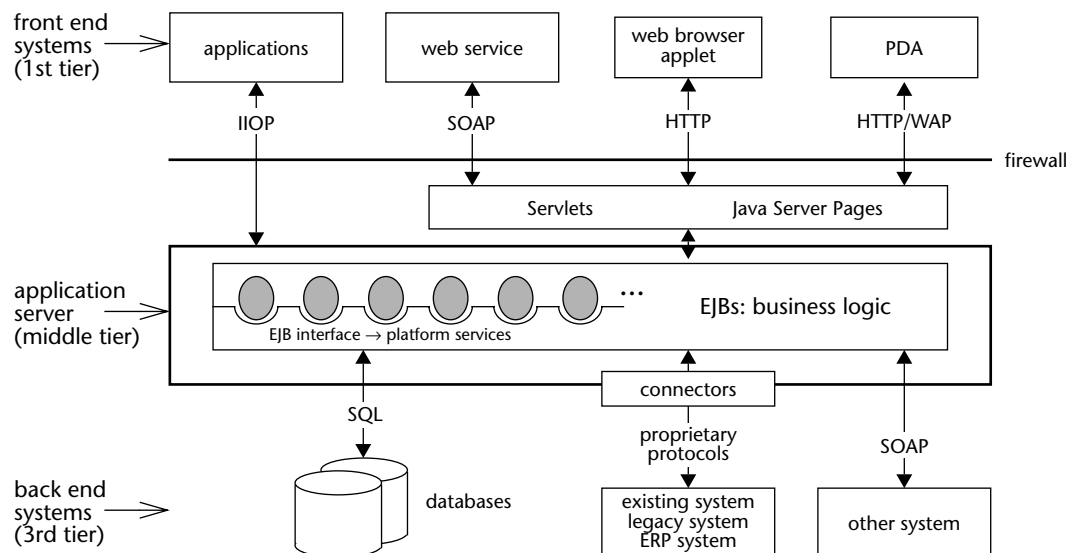
Application Server – Definition In the context of this article, an *application server* is a platform and infrastructure for Enterprise JavaBeans, providing support for a component model, transaction management, and resource/service management.

Enterprise JavaBeans – Definition Enterprise JavaBeans (EJB) is a standard server-side component model for component transaction monitors (CTM) [1].

Important points, discussed in more detail later in this chapter:

- EJB is a J2EE² standard, now at version 2.0. Originally defined by Sun Microsystems, the standard has been adopted and is supported by major vendors including HP, BEA, IBM, SAP, and Oracle.
- EJB is a server-side component model, ie. it allows to define and implement software components that encapsulate business logic and data in portable (business) objects.
- EJB is a component model for component transaction monitors (CTM), a hybrid of traditional TP monitors and ORB technologies, which allow for distributed and transaction-aware business logic.

Application Architecture Applications servers usually represent the middle tier of business applications, where the business logic proper is implemented using EJBs, serving the presentation tier (first tier), and making use of the database backend tier (third tier), as depicted in the following figure.



Common 3-tier architecture The application server provides the business logic in the middle tier.

Please note that all tiers can be distributed over different computers and also different locations.

2 J2EE = Java 2 Enterprise Edition [3]



Products Major application server products include WebLogic (BEA), WebSphere (IBM), HP-AS (HP), Oracle 9i AS (Oracle), iPlanet (Sun), and JBoss (open source).

More Information Please refer to [1], [2], [4], and the respective vendors' web sites for a more comprehensive and precise description of Enterprise JavaBeans and application servers, plus the related specifications.

Relation to Agent Technology

Software Agents Technically speaking, software agents are “autonomous, problem solving software components,” ie. they are also components that implement business logic and state, having a standard interface to their underlying agent platform, which in turn provides common functions and services to the agents, just like the application server does to the EJBs. From this perspective, EJB application servers resemble agent platforms.

Please note that here we're only concerned with the *system-level* aspects, not application level functions and features, like support for ontologies or different high-level communication and coordination protocols.

Distributed Objects **Object View.** Distributed objects like EJBs allow parts of a system to be located on separate computers, possibly in many different locations, where they make the most sense.

Agent View. Also software agents can reside on different platforms, pursuing their tasks in coordinated ways, for which a communication infrastructure is needed, as with EJBs. Consequently, application servers provide the needed functionality for object distribution and communication, which can be used for software agents as well.

Server-Side Components **Object View.** A server-side component model defines an architecture for developing such distributed business objects, and it's being used on (middle-tier) application servers, which manage the components at run-time, and allow their interaction with other components and systems in other tiers.

Server-side component models support a programming style which allows the run-time behavior of the component to be modified at deployment-time. This way, the server administrator can configure and control a component's transactional, security, and persistence behavior.

It's important to note that EJBs are a component *model*, ie. there is a defined way of constructing these components, including their interface to the underlying platform, ie. the application server. This yields a high component portability between different applications and also between different application server products – full portability, ie. without changes, in theory, “very good portability” in practice.



Agent View. Also agents, being “autonomous problem solving software components,” implement part of the system’s business logic and state, and it is obvious that agents can profit to a high degree of a standardized, server-side component model. Of course, agents can potentially be more capable than objects, which are basically purely reactive elements with design-time-defined behavior. Run-time configuration – or even an agent’s change of behavior at run-time based on run-time-negotiated interaction – needs much more related flexibility regarding both the “agent component model” and also the platform, so EJBs are not agents, and applications servers are not agent platforms, but a stable and practice-proven basis is there.

**Component Transaction
Monitors (CTM)**

Object View. Application servers *are* CTMs. They implement robust server-side component models that make it easier for developers to create, use, and deploy business systems. Application servers provide an infrastructure that can *automatically* manage transactions, object distribution, concurrency, security, persistence, failover, load balancing, and resources. And they are capable of handling huge loads and mission-critical work, but also provide value to smaller systems because they are easy to use.

CTMs have evolved as hybrids of distributed object technologies like ORBs (CORBA), Java RMI, DCOM, and traditional TP monitors like CICS or TUXEDO.

Object request brokers (ORBs) have been around for some time. ORBs facilitate connectivity between client applications and distributed objects, by providing means to remotely locate and use the business objects. ORBs thus provide directory services and a communication backbone for distributed objects. However, ORBs provide only a fairly crude server-side component model that places the burden of handling transactions, concurrency, persistence, and other system-level considerations on the shoulders of the application developer: these services are not automatically supported in an ORB, but the developer must explicitly access these services (if available), or even develop them from scratch. That’s where the CTMs (or the application server’s) other “heritage” comes to a rescue: TP monitors.

TP monitors have literally existed for decades,³ and the technology behind them is rock solid. TP monitors systems automatically manage the entire environment that a business system runs in, including transactions, resources, and fault tolerance.

Together, ORBs and TP monitors “team up” to a very useful technology!

Agent View. On a system-level, many of today’s agent platforms resemble very much the description of the ORB above: the agent platforms provide functionality for finding and addressing other agents, including their services (directory), plus inter-agent communication (messaging). What these platforms lack are the TP monitor features, in particular transactions, resource management, security, fault tolerance, and scalability.

3 CICS was introduced in 1968 – this was before Neil Armstrong set foot on the moon...!



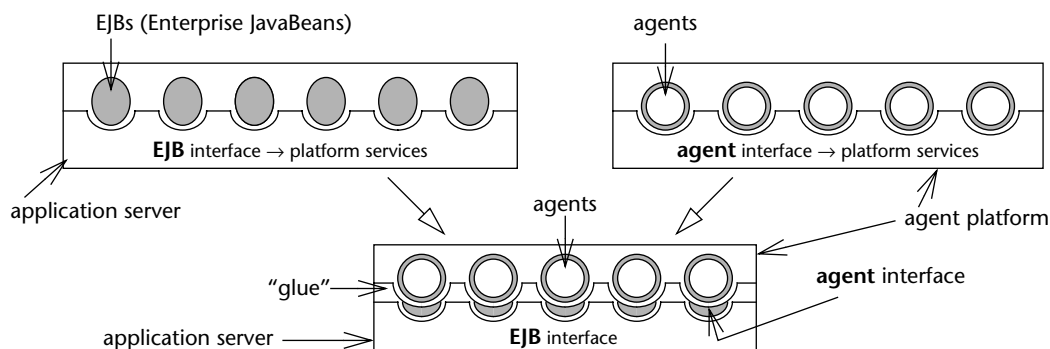
Application Server Based Agent Platform

Introduction To summarize:

- J2EE application servers provide a baseline of functionality that makes it easy to develop distributed business objects, and assemble them into robust and function-complete business solutions that perform well and are highly scalable.
- Next generation agent platforms for business applications must additionally provide “TP monitor features” in order to achieve the same grade of system-level functionality and quality as application servers, to cope with the requirements and expectations of the “real IT world.”

To develop such an advanced agent platform, it would be very difficult, if not impossible, to catch up with the lead of the “application server world” over the “agent platform world,” from a technical system-level point of view. And considering the major investments and accumulated experience, it would be unwise to even *try* to catch up, at least from an entrepreneurial point of view. Hence the concept of combining the agent platform with existing application server technology presented here.

Basic Concept The following figure depicts the basic idea of the application server based agent platform.



Using application servers as basis for an agent platform

On a basic conceptual level, agent platforms resemble application servers. Both provide – or should provide, as discussed before – means for distribution of components (EJBs, agents), the needed communication and cooperation between those components, plus the “TP features” like transactions, resource management, or fault tolerance.

So if EJBs *were* agents, most problems would be solved. But they are not.

Software agents can be described as “autonomous (possibly mobile) software components that are capable of exhibiting flexible problem solving behavior in pursuit of their objectives” (based on reactive, pro-active, and social behavior patterns).



Thus, software agents have characteristics that go beyond the capabilities of EJBs, for example:

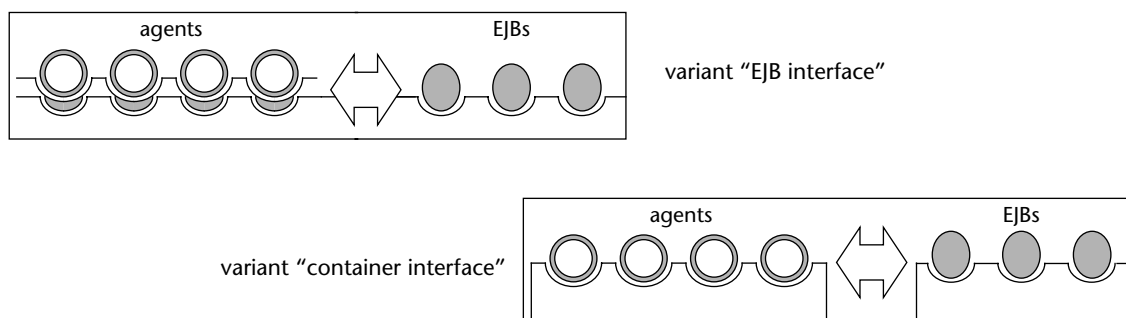
- ❑ EJBs are purely reactive elements, agents can be pro-active.
- ❑ EJBs have technical constraints, eg. EJBs are not allowed to start own (Java) threads, and cannot define own class loaders.
- ❑ EJBs are technically not separated from each other (share name space).
- ❑ EJBs are not mobile, due to limited support for dynamically defining classes.

Hence, even though a major part of the system-level features of an agent platform is provided by an application server, specific agent capabilities must be added via an add-on component, depicted as “glue” in the figure above – “glue” meaning a connector and transformer between the EJB interface provided by the platform, and the interface needed by the software agent.

Platform Architecture

The detailed internal architecture of the application server based agent platform is beyond the scope of this article. Generally speaking, an application server has two possible “hooks” for integrating software agent technology, the EJB interface, and the container interface (see figure):

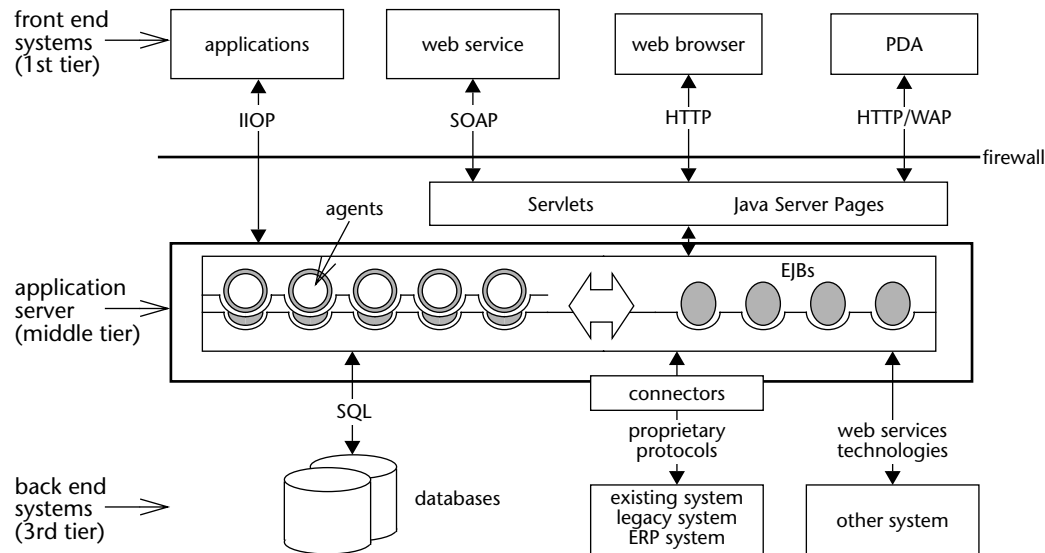
- ❑ In the first variant (EJB interface), agents are constructed using standard EJBs, with the “glue” component adding agent capabilities. These “EJB-based” agents run in the usual way inside the EJB container. As EJBs are standardized components, with a standardized interface to the application server, this solution can be implemented without inside knowledge of the application server. Moreover, the thusly constructed agents are highly portable between different brands of application servers (like EJBs).
- ❑ In the second variant (container interface), a specific agent-container is used, which is running in parallel to the EJB container (and possibly other containers, like for servlets). This allows a much tighter integration into the application server. However, the container interface is neither standardized nor published, rendering the resulting container-product vendor-specific. The agents themselves are again portable.



EJB interface vs. container interface



Agent-Application Architecture The following figure shows the principle architecture of a software application based on the application server based agent platform (variant “EJB interface”).



Common 3-tier architecture The application server based agent platform provides the business logic in the middle tier.

Please note that this technology easily allows to combine agent and “non-agent” technologies, easing integration and interoperability.

Business Applications The application focus of this agent platform is on server-based business applications, ie. the type of applications where one would (or could) use an application server, eg. in e-business, banking, or production planning and scheduling, just to name a few.

Summary

Software agent technologies face substantial technological and organizational challenges in IT departments, ranging from basic product *quality aspects* (reliability and failover, performance and scalability, security), needed business *functionality* (transactions, sessions, persistence), *migration* and *extension* capabilities of existing operational systems, *integration* with currently operational systems, products, and solutions to *adherence* to current industry *standards*.

Today, J2EE application servers represent a major technology that fits these requirements and expectations of the industrial software development. Application servers are platforms for the so called Enterprise JavaBeans (EJB), standardized software components to construct reliable and



maintainable distributed business applications. Application servers are well-accepted in industrial and commercial environments.

Technically speaking, also software agents are components that implement business logic, even though they have additional characteristics like autonomy and flexible problem solving capabilities that go beyond the ones of standard EJBs.

However, from a system-level perspective, EJB application servers resemble agent platforms. Agent-based business applications can be built and operated on application servers, if they are extended to provide the additional capabilities of software agent compared to Enterprise JavaBeans.

This approach yields the following advantages:

- ❑ build upon mature industry-grade products, that are being further developed and maintained by major work forces of their respective vendors;
- ❑ build upon the basic quality features of the underlying J2EE application server (eg. reliability, performance);
- ❑ have the needed and proven business functionality readily available (eg. distributed transactions, persistence);
- ❑ have acceptance with regard to corporate IT strategies, investment protection, know-how protection;
- ❑ adhere to accepted standards (J2EE);
- ❑ be easily integrated with existing systems and environments;
- ❑ provide a path to enhance and migrate current solutions.

Thus, application server based agent systems have a great potential to (pro-) actively support agent concepts and technologies to finally find their deserved position in the real IT world outside the labs, in a pragmatic, “down-to-earth” way.



Appendix A References

- [1] Richard Monson-Haefel: Enterprise JavaBeans
 O'Reilly
- [2] EJB Specification
 <http://java.sun.com/products/ejb/>
- [3] J2EE Specification
 <http://java.sun.com/j2ee/>
- [4] JBoss Website (open source)
 <http://www.jboss.org>