

# Clash of the Titans

## Part 3: Why WebAS beats Microsoft.NET and J2EE

By Axel Angeli, logosworld.com

**Technical Overview:** First there was Java, then Microsoft.NET, and finally WebAS: these three so-called application development frameworks triggered a renaissance of the concept of the virtual machine. Their common vision is to provide an environment to write robust and reliable transaction-based software that is both independent of the underlying hardware and operating system and easy to maintain. With the ultimate goal of writing robust and reliable software, a framework integrates a myriad of API to be used during runtime combined with integrated tools for rapid development and extreme programming. Choosing a framework for new, individual enterprise or web service developments is not an easy, but one thing we do know: the stakes are high for the vendor that "wins" control of the application development framework. This "clash of the titans" is the battle for control of the development standards and the cross-selling of the applications that are most compatible with it. SAP is a late arriver to this enterprise skirmish. task. Casual observers might scoff at the idea the SAP's WebAs infrastructure is as sophisticated as Java or .NET, but are readers will see in this paper, SAP's WebAS falls short in no key functionality, and debugging, transaction management and deployment control are by far more elaborate in WebAS compared to the corresponding approaches in Java and Microsoft.NET.

### Frameworks as the Next Evolution in Application Software Development

The prominent discussion in cutting edge IT circles is the battle of the frameworks, namely Microsoft.NET, J2EE and WebAS. These three platforms are the next evolutionary steps in application computing, promising to allow better, faster and more reliable software development for business and industry use by covering up (and making "transparent") the tremendous differences of hardware and operating system differences from company to company.

A framework is essentially a fully-functional virtual machine that emulates a high-end, state-of-the-art computer. All enterprise applications are designed to work on top of this virtual machine as if it were the actual hardware platform. This virtual machine is, however, not the whole architecture. It is complemented by a standard development environment that assists the developer in writing efficient, harmonized and error-free code, and by an opulent library of application programming interfaces. So, we define the framework environment in an easy formula:

Framework = Virtual machine + IDE + compiler + API library

Technically, a framework adds a new layer to the classical stack of machine  $\beta$  application communication, which then looks like this:

1.	Application
2.	Application programming interfaces
3.	<i>Framework built from a virtual machine and an integrated development environment</i>
4.	Database system
5.	Operating System
6.	Hardware platform

The idea of a virtual machine is everything but new. Every third generation programming language and most modern operating systems including OS/2, Windows NT and LINUX tried to be such an omnipotent virtual machine. The reason why these approaches did not break through is because they all lacked important features. These new frameworks also bypass the operating system, gives them significant advantages for the user of a program, e.g. addressing a graphic card results in more realistic motion effects on the screen, or an individual printer model has much more valuable features than the operating system designer could have anticipated.

### The Special Requirements on Enterprise Computing Solutions

When talking about enterprise applications, we have a certain vision in mind. We are not talking about sterile, highly formalized, vertical "silo" applications like those found in accounting environments, where a standstill

of the central IT systems may cause same irritation and an extended coffee break, but does not produce a substantial collateral damage apart from the costs caused by repairing the actual error. The same goes for most office-centric applications like purchasing or billing where a delay in fulfillment may well be made up for the lost time shortly after.

When we talk about modern enterprise applications, we are talking about complex, multi-department enterprise structures like found in the process industries. There, a standstill of a fully-automated production process causes real costs. Imagine an ongoing chemical process of some kind - suddenly, the central IT system stops sending directives of what chemicals to produce and in what quantities. There, you cannot simply interrupt the process and wait until the incident is over - otherwise the brew will keep on cooking and reacting and the machine master will end up producing anything he thinks might be right. Not only will this cause a state of production chaos as long as the outage lasts, but if it were necessary to stop the machine completely, a restart might cause tremendous extra costs and efforts for cleaning, disposal of garbage remainders, overtime work, etc., - real money that easily could be as high as several hundred thousand dollars per incident.

## **Influencing Factors for Making an Application Framework Decision**

As a consequence, the key argument for installing an ERP application is reducing the risk and duration of unplanned down times, and maximizing the mean time between failure and the ability to inspect the behavior of the software during runtime in the production environment - while still providing the full range of integrated functionality required by the production process. As it turns out, these same "integration advantages" that led virtually all major companies to install ERP systems also provide the most compelling argument in favor of WebAS as an application framework for SAP users:

- Having a robust reliable underlying technology throughout the enterprise;
- Having the ability to inspect the program source within the productive environment in the runtime debugger;
- Being able to modify the source in the productive environment; and
- Being able to call on a staff of qualified technical people who already have a thorough understanding of the business and a good grasp of the existing enterprise applications and technologies.

In particular, the ability to modify the source code in the productive environment provides SAP users with a powerful feature otherwise only found in scripting languages on hand. Through this functionality, we now have a way to understand the actual program flow and a powerful means to reduce the time we need to react on disorders.

## **Business Competence**

A critical factor in deciding on a certain technology is the amount of competence and the quality of the technical education available for our users. Here we encounter a nearly surreal contradiction between the public perception and the reality. Although a large number of developers with many different backgrounds know about Java and have already done some testing or "sandbox" developments in Java, there are only a very limited number of good Java developers who are truly familiar with Java and have developed solutions during a full life cycle. And amongst those, only a small number have worked in the kind of "high stakes" enterprise environments that large SAP customers are running on. If our basic skill set requires the combination of Java skills, plus business know-how, plus transactional database programming, then the number of qualified people is sparse.

On the other hand, if you work in a multi-national company running on SAP and look around, you will see certainly more than one person with substantial SAP and ABAP know-how, and most ABAP developers will also have a good understanding of the underlying business processes. They are used to working on major, enterprise development projects with a wide scope and very demanding expectations. The situation is similar outside of companies on the free market: if you need competence around any area of SAP, then there is an abundance of solution providers to buy know-how from.

In terms of Microsoft.NET, the situation is somewhere in the middle. Although many people are disparaging about Microsoft products, there is a certain familiarity with Microsoft Windows development tools (especially Visual Basic) around. Due to the lack of true ERP solutions developed on Microsoft.NET, it again falls short against WebAS, but because Visual Basic people are historically closer to the business than to the core development, their value for true successful business development is to be rated higher than Java specialists.

So we see that, from the essential aspect of know-how and experienced resource availability, there is a clear advantage of choosing WebAS over Java or .NET if you're an SAP user. Not only do SAP professionals have, in general, greater business know-how the existing SAP development experience found in the companies leaves the competition far behind. One might object that this will rapidly change as soon as Java becomes even more

popular. But this is not going happen in the near future. There are no widely accepted and standardized, Java-written business and ERP solutions in sight that can compete with the current dominating leaders in today's ERP market - namely SAP, BAAN and PeopleSoft. For Microsoft.NET, the situation is not much different. The existing challengers for SAP & Company are mostly found in the "small market," as SAP goes head to head with Microsoft to win the ERP accounts of the Small and Midsize Businesses (SMB) market. Products like Microsoft NAVISION, Great Plains and SAP's own Business One are still proprietary solutions that are not actively delivered with the source code to customers.

*Takikng all this into account, here is the verdict: until there is at least one large, "killer" enterprise-wide, application built for either Microsoft.NET or J2EE - one that represents a full life cycle development of a major ERP solution and grants the software user the right to adapt the source code on its own discretion, then there will be no realistic break-through for Microsoft.NET or J2EE as a leading technology in the enterprise development platform business - arguably the most important and financially lucrative sector of of the IT business.*

Given this reality, Java and Microsoft.NET will still have their position as niche technologies covering special areas like interface design, hardware near technologies or classical HTTP applications.

Today, there is one clear statement to make: from an SAP business manager's point of view, the only wise and economic decision for new larger enterprise developments is to build upon the existing ABAP and SAP business know-how acquired throughout many years. \*\*\* Axel, if you think that SAP's technology is so powerful that other companies running other ERP packages such as Oracle or PeopleSoft should switch to the Web App Server, you should say so here. Otherwise, we should continue to qualify this as a paper written for SAP users, not all ERP users.

## Robustness

The reliability and stability of an application and the underlying framework is a key demand for successful enterprise applications. In extreme, highlyautomated businesses, people often speak of the magic five nines, the 99.999% availability of an application. This means that the unplanned downtime of the software should be less than six minutes per year - and this covers all causes of malfunction from hardware failure to programming errors. Six minutes is a high standard indeed - this is usually not even enough time to reboot your server! Most businesses are not that mission-critical, and can afford a bit of downtime. Aweekly crash might initially be taken as a welcome coffee break for the production staff, but eventually this getsannoying.

SAP WebAS is a front runner in this area, as the underlying SAP kernel hardly ever reports a crash. Whenever malfunctions of the SAP software are reported, they can typically be traced back to misbehavior of the database system or the operating system. In my experience, there is hardly ever a programming error in the SAP application that can shatter the stability of the running SAP instance. This has been consistent since the first appearance of R/3 in 1992.

Experiences on the robustness of J2EE and Microsoft.NET are still preliminary, because both frameworks are too young, and sufficiently complex enterprise applications to compare it with are still not on the market.

From my own research and analysis I am currently heavily disappointed by the existing Java implementations I've seen. Even on small, clean systems, it was pretty easy to bring the running Java engine down. This mostly happens during installation of a new package, and it is often due to multiple running instances of the Java engine in contention. There has been a lot written about improving stability and resolving contention in Java, but this is the task of the framework designers, not of the application developer, so all these solutions are hypothetical for the average developer, as they need to implemented in the Java kernel and are hence in the hands of SUN Microsystems.

Microsoft.NET, in general, has appeared to me to be more stable.. In fact, any problem with Microsoft.NET I have encountered up until today has been due to errors in Windows itself. However, this is not an adequate excuse for our comparison, given the fact that WebAS is able to shield its functionality completely from the operating system. A crucial question is still whether Microsoft wants to see Microsoft.NET as part of Windows, or as an additional layer independent of it.

## Error Finding During Runtime ("life debugging" and "hot patches")

While producing code according to a prepared concept might require widely the same efforts in Java, ABAP or Microsoft.NET, finding errors or verifying that cose does not. The main difference between ABAP and the other frameworks is the way that code is deployed. ABAP always deploys source code, which is temporarily

translated into temporary runtime code. Microsoft.NET and Java are compiled into an intermediate code. Only this intermediate code is deployed and executed.

Simply put, , ABAP appears like a scripting language to the user, while Microsoft.NET and Java behave like compiler language. This makes a dramatic difference in finding errors or tracing the behavior of a program in productive environment. I cannot emphasize enough that the enormous success of SAP is mainly due to giving the user the possibility to debug the source code in its own complex runtime environment, and to make proper enhancements on its own risk and responsibility. The high quality of the ABAP debugger, along with the ability to patch source code, puts SAP in a strong position to outperform all its contenders.

## Change Management and Deployment

The next key factor in our comparison is how change management is executed and changes are controlled. In this discipline, SAP is in a class by itself. The SAP source code deployment via the SAP Transport Management System (transaction STMS and the *tp* utility) is a feature that is still unbeaten by class and ease of use. For J2EE and Microsoft.NET, the deployment is done in a traditional way via install packages. Hot deployment, i.e. modifying an application without stopping the application itself, is ultimately possible, but source code deployment is not. And standard automation of deployment between development and production systems like the SAP TMS is not in sight for either J2EE or Microsoft.NET.

There are some other new approaches in this direction, like Borland's StarTeam® solution, but these are isolated approaches of individual implementations of the J2EE standard, when what we're talking about is something that doubtlessly belongs in the core of the framework. It appears to me that the importance of this issue is not yet perceived by the designers of Microsoft.NET or J2EE. This unveils a hovering problematic of the whole open source community that drives (especially) the J2EE architecture: the design is still too focused on classical technical aspects like performance or platform independence, because there are too few people within the design teams that come truly out of the enterprise business environment.

Change management and application patches are important for the daily work of any enterprise system, and may well be the deciding factor for its success, because it influences the reaction time both on malfunctions and on software changes imposed by changes in the real business. We all know the old, tragic joke about IT, where a clerk tells the customer: "I would do what you want, but the computer does not allow me to." There is no other part of IT that repeatedly causes so much trouble and latent dissatisfaction than a scenario that forces the business to accord with the software instead of making the software work like the actual business does. What we hence need is the ability to change software in production environment in an easy, fast and controlled manner. That is what TMS does.

While change management is of such importance, hardware independence (an admitted advantage of Java and .NET) is only interesting for a software manufacturer that wants to sell a standard package to many different clients. A typical IT shop decides once in a decade for certain hardware and operating systems and that's what they're stuck with. . \*\*\* Axel, I didn't understand this point, so I took it out. If you can restate it, and you think it's important, then add it back in.

## Managed Applications

What is the temptation of a virtual machine? Why are they becoming so popular all of a sudden? And what are the components of their underlying technical structure? Let's try to gain some insight on this.

A virtual machine (VM) emulates a universal computer, and hence tries to overcome the differences imposed by the individually different hardware and operating system features and restrictions. A virtual machine is not all that different from an operating system and indeed, an operating system like UNIX had been original designed to act as such a VM by providing a universal and unique API for applications to access all peripherals like a printer, scanner, drives etc. Application programs, however, were still executed in the machine's native code. To execute a program under Windows, you would have to compile the code into an .EXE file first.

UNIX tried to overcome this problem (partially) by letting the C compiler be an integrated part of the operating system. This allowed the sharing of source code between different UNIX machines, without knowing about the underlying hardware. But, still, the code had to be compiled into machine code.

The modern VMs of WebAS, Microsoft.NET and Java add a new layer between the actual machine code and the application code. Instead of compiling the source into machine code, a new intermediate code is generated. When the application is called, the intermediate code is interpreted by the runtime. This is not really a new idea. The traditional name for an intermediate code of a VM is *p-Code*. During all times of modern computing, the idea of a VM has been popular and p-Code compilers have come and gone ever since.

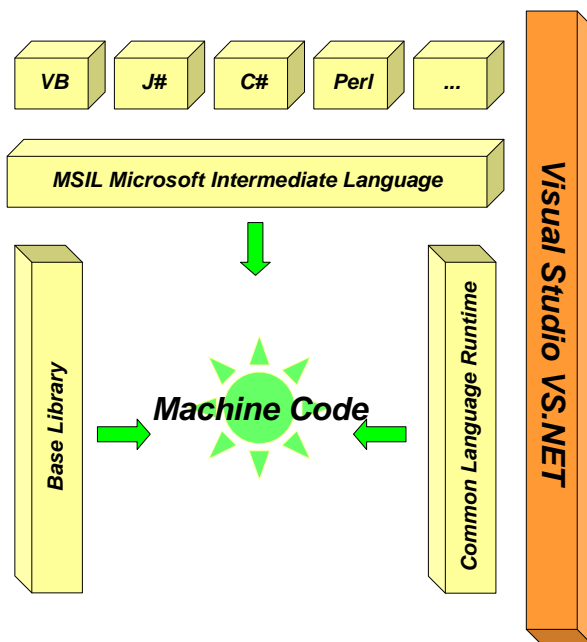
**Figure 1: C#-Code translated into Microsoft Intermediate Language**

---

\*\*\* Axel, I don't see this figure.

The p-Code language which is generated and hence interpreted by the VM should actually be the language of the processor. And indeed, many processor manufacturers, e.g. Motorola, tried several times to design microprocessors that allowed defining new low-level instruction as procedures of the hardware-coded ones. This had been countered by IBM's vision of RISC processors. These processors tried to unload the abundance of features from the processor while implementing a virtual processor as the inner kernel of the operating system. Figure 2 illustrates the aspects of the Microsoft .NET architecture.

## Microsoft.NET Layers



**Figure 2: Architecture of the Microsoft.NET Framework**

---

The question is: why don't we use a simple source code interpreter during runtime? This is mainly due to efficacy aspects. For many applications, there will be hardly a remarkable difference in performance between a compiled and an interpreted code. When it comes to complex applications, the compiler can take up a lot of time consuming optimization of the code. This is especially noticeable if there is a lot of memory operations, allocating and disposing of heap memory.

Such a VM gives access to a complete new variant of application processing: managed applications. Instead of simply running the code and waiting for the result, the VM can now survey each step and each statement during runtime to control its behaviour. Native code compilers submit their code to the runtime like sending a car into tunnel and waiting at the exit for it to reappear. If something unexpected happens, nobody can control it, because

it happens in the dark. The VM controls though the execution. It allows halting the program at predefined breakpoints, capturing each command in order to delay its execution. What does this mean in practice?

- Debugging

The VM allows an easy way to do source code debugging. Because every statement of the p-Code is executed by the VM, the VM can halt the program there, display the full memory state and allow manual interventions at the breakpoint. In SAP and Visual Studio, the p-Code maintains backward references to the source code, so the source code can be stepped through in the debugger. This feature is one of the most appreciated parts of the ABAP workbench and attributes to great success of SAP. Without this debugger, SAP would have probably failed, because it would have excluded the help of the uncounted number of developers sitting at the costumers' site and enhancing the applications for individual needs instead of waiting of a SAP customer representative to kindly and gracefully respond to a problem.

- Transaction management

The VM allows for easy and sophisticated transaction management. In traditional application design, the application programmer had to start a transaction, tell the operating system or transaction manager when a transaction related command is executed, and to individually commit a transaction. This is basically still the case - however, the virtual machine can capture all commands and decide as needed whether the command might be attributed to a transaction, e.g. database writes. These commands will then be collected by the VM and its execution deferred until a transaction commit occurs or the program terminates normally.

## Secondary Decisive Factors

The key factors discussed above may already give enough hints for a proper platform decision for many SAP managers. . But let us now elaborate on the remaining factors and disciplines that may influence a decision for a framework in an individual situation.

- A virtual machine independent of the hardware platform

All three frameworks are designed to be independent of the hardware platform. For Microsoft.NET, this is only true in theory as it still loosely depends on Windows, which runs on Intel-based machines only.

It is a seldom-remarked fact that SAP R/3 has indeed been a virtual machine since its beginning. Built around a practical development environment without gadgets and intellectual vanities, the ABAP workbench and the SAP kernel provided functionalities that are still ahead of J2EE and MicrosoftMicrosoft.NET.

- A virtual machine independent of the operating system

WebAS supports the widest variety of operating systems, including Windows NT-based machines, various UNIX dialects (SUN Microsystems, HP etc.), IBM pSeries (AS/400) or IBM zSeries (s/390).

The implementation of J2EE is actually left to the third-party manufacturers, usually the makers of enterprise servers. Currently, the J2EE implementation included with IBM WebSphere™ supports the widest range of operating platforms. There are other implementations for Windows and Linux, mainly, e.g. Borland's Enterprise AppServer and SUN Microsystem's own SUN ONE application server.

Microsoft.NET currently only supports Windows.. Rumours are spreading that Redmond is working on LINUX-based versions for Microsoft.NET as well.

- A sophisticated and transparent transaction management

Due to the way WebAS is designed, every application runs as a separate logical unit of work, which is nothing else than a transaction. This strategy is apparently different from J2EE and Microsoft.NET, both of which require the user to explicitly denominate the begin and end of a transactional code sequence. This turns out to be a killer argument in favor of WebAS. Transactions are the heart of business database applications. WebAS frees the application developer from transaction management unless he expressively wants to take over control. For J2EE and Microsoft.NET, transaction management is an add-on feature, which needs to be explicitly activated by the application developer.

While every WebAS application runs forcibly within a closed transactional environment, J2EE introduced the optional programming model of the *Enterprise Java Bean (EJB)*. EJBs are object instances that create a closed shell that allows the user to execute an application in its own transactional environment. EJBs need, however, to be explicitly activated and require a nominal administration effort.

- A reliable just-in-time compiler

This has always been a great feature of SAP. Whenever you change the source code of a program, it is automatically compiled (SAP calls it “to generate”) the first time it is executed. Because deployment in SAP means to deploy the source code, this feature is available in every environment. This feature is generally known as a just-in-time compiler.

J2EE uses the Java Development Kit to compile the Java code into the runtime code. The generated code is portable, but it needs to be deployed in compiled form. Ad hoc patches of the source are not possible.

Microsoft.NET works the same way as Java. However, the Microsoft.NET intermediate code is its own low-level programming language, comparable to C++. Generally, it is possible to write programs directly in Microsoft.NET intermediate code.

Both Microsoft.NET and J2EE provide tools that they call “just-in-time compiler” as well. However, these JIT-compilers compile the intermediate code into optimized machine executables. Source code changes need still to be translated explicitly into intermediate code by the developer.

- A full featured programming language

WebAS relies on the good old proven ABAP/4 programming language. You can also develop in the mainstream object-oriented ABAP/Objects, which is a sort of “ABAP light”. But wait! Before you consider drilling your development team to develop in ABAP/Objects, consider that for the sake of object-orientation, a lot of the ease of use and the seamless integration into the development environment had to be thrown over board, and object-orientation is widely useless in combination with a transaction-based runtime like the SAP kernel.

J2EE supports Java, which is, per se, a big minus, because the C-like Java syntax is still light years behind what you would expect of a modern, easy-to-learn, easy-to-read, easy-to-understand programming language.

Microsoft supports several programming languages and encourages third parties to adapt their language compilers to generate Microsoft.NET code. The most popular languages are currently Visual Basic, J# (a Java dialect implementation of Microsoft) and the new language C#, being a best-of-potpourri from C++, Visual basic and Java.

For those who cannot decide between the Microsoft.NET and J2EE, the choice might be Borland Delphi. Using Borland Delphi, you can program in a modern, high-level programming language, one which also lets you compile to Linux or Microsoft.NET.

- Platform-independent APIs for nearly all important and nice-to-have features of a database and operating system (“one face to the developer”)

Microsoft.NET and J2EE provide proper APIs to the underlying operating system, mainly for accessing the file system and designing Graphical User Interfaces (GUI). GUI design is made for all three frameworks either as HTML for access through browsers and the web or through their own, individual screen APIs, which are the SAPGUI for WebAS, Windows for Microsoft.NET and SWING for J2EE.

- A real time updated data dictionary and development repository

Database design and application development are still kept independent of each other in Microsoft.NET and J2EE, while the database and database access is an integral part of WebAS and ABAP, which makes application development a snap. Considering that enterprise applications are definitely based at least 80% on databases, and this alone will count for reducing application development and testing costs by half.

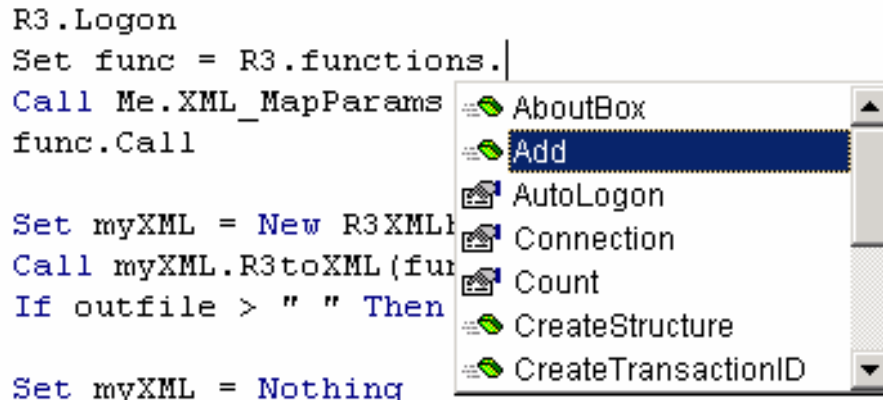
Program reliability is also enhanced in WebAS, because the validity of database structures and database access is checked by the compiler against the repository, while the standard ODBC and JDBC accesses will find errors during runtime only.

- A harmonized and integrated IDE

\*\*\* Axel, define IDE here.

A good development environment must be well interwoven with the target you are developing for. The IDE should assist you in entering the code and alert you if syntax is wrong or semantics do not follow certain specifications, like code that is never reached during execution. When accessing objects or library elements, the IDE should give you sufficient online information to properly enter the statements. Microsoft and most Java editors come with the *intellisense* technology (For an example of Intellisense coding in Visual Basic, see Figure 3). Intellisense displays the possible methods, properties or parameters while you type in the editor. Unfortunately, there is yet nothing comparable in WebAS, and its only entry help is the insertion of templates

and pretty printing the code. It should not remain unmentioned that the ABAP editor lost some of its power since release 4.6, e.g. the editor lost features like column marking (ctrl-Y) and automatic syntax high-lighting.



**Figure 3: Intellisense Coding assistant in Microsoft Visual Basic**

- One-touch cross referencing from the IDE

I am still waiting to see something similar in any other development environment: in ABAP, you click on the name of a database table and it immediately displays its definition, then you click the little spider icon and ABAP displays all programs and database view where this table is used. This alone makes the heart of a developer cheer.

- Application of open communication standards, especially HTTP, XML and SOAP

The sacred value of an enterprise application is its ability to communicate and integrate seamlessly into an enterprise IT infrastructure, which typically have a heterogeneous architecture. Having software components from a multitude of vendors, it is essential from an economic point of view to have as little impedance between two communication partners, where impedance in this context indicates the efforts and costs to establish a connection between two partners and to convert protocols. If both partners speak the same protocol via a permanently available connection, the impedance is low. The more different the protocols of a sender and a receiver are, the more complicated are the programs to convert between the protocols, and the higher is the risk to falsify data with the conversation.

The standards for communication between remote computers are now clearly sorted in favor of HTTP for the machine-to-machine basic protocol, and XML as the markup language to structure the exchanged content. HTTP is one of the base protocols for both J2EE and Microsoft.NET, and as such is supported by their respective kernel APIs. WebAS integrates the HTTP support in a similar manner.

XML as the standard language to structure document data is extremely well supported by the Java community and by Microsoft.NET. For J2EE there is number of open source libraries to found in the Open Source Foundation (OSF), popular ones are XERXES found on <http://xml.apache.org/> or JDOM by <http://jdom.org>. However, XML is not support as part of the kernel and simply “bolting on an XML SAX/DOM parser” is not what you would expect for such an essential element (as stated in a great early article on Microsoft.NET and J2EE by Jim Farley, [http://java.oreilly.com/news/farley\\_0800.html](http://java.oreilly.com/news/farley_0800.html)). XML is so crucial for all modern enterprise applications that not having XML as part of the J2EE standard severely endangers the portability in which Java takes so much pride.

Microsoft implemented all XML utilities in a single library, msxml4.dll, which is part of the Internet Explorer browser. While the OSF tries to strictly adapt the W3C (World Wide Web Consortium, <http://w3c.org>) recommendations, significant enhancements to this standards are provided by Microsoft as usual. It is pointless to wonder if it is Microsoft’s intention to sabotage the universal standard in favor of its own, or if it simply tries to get the most out of the underlying Windows system. Although these differences are on a very abstract level, it might complicate communication between systems based on different standards, as happened when Microsoft transformed Java into J#.

The XML support of WebAS ABAP is there, but needs still major enhancements to step up with the Java or Microsoft.NET world. This is to be expected in releases WebAS 6.4. For the SAP WebAS Java engine, there is always the option to use the OSF libraries.

As far as SAP's support for SOAP,

one often gets the impression that SOAP has been patched into the kernel rather than seamlessly interwoven with it. What I expected from SAP is to integrate HTTP, XML and SOAP support into the ABAP language, not as library or API calls but as language elements, e.g. a SOAP call could be easily requested by modifying the syntax for a Remote Function Call. But at this point, that is not the case.

## **Clash of the Frameworks - Summary Analysis**

It is clear that for new enterprise application development on SAP sites, a proper choice has to be made between Microsoft.NET, J2EE and SAP WebAS.

### **J2EE**

J2EE is based on the steadily growing fan community of JAVA. The number of J2EE engines available is still limited. IBM WebSphere, and Borland Enterprise AppServer have proven that they can deliver a stable and sizable platform for enterprise applications. Others are trying to enter this market.

The general argument for J2EE and JAVA is its portability across all hardware platforms. Let aside the fact that WebAS does the same, there remains still the question: who wants to port a solution across platforms? Unless you are a software manufacturer you usually decide for a platform, develop your application and retain this application for decade or longer. Once the platform is decided, there is seldom a need to put the application onto another environment.

### **Microsoft.NET**

Microsoft web servers are a common standard in a majority of enterprises, which makes Microsoft.NET the apparent choice for smaller web server based applications. Because most companies run their company network on Microsoft server, one can also assume a certain know-how about the Microsoft.NET framework within a company, definitely at the same level as for J2EE, and probably higher. This will reduce the education cost for the staff, that is responsible for the daily operation monitoring.

Microsoft.NET has no integrated database support, they are accessed via ODBC or as SQL client requests. The Visual Studio environment has support for every element of development, but the ease of use and full integration of the SAP development environment is still far away.

The specification of the Common Language Runtime is open, so that Microsoft.NET allows third parties to develop compilers that compile into the CLR code. Until now only Borland's Delphi uses this opportunity.

The large number of different programming languages is also an argument in favour of Microsoft.NET. The most attractive appeal, however, is the abundance of already existing COM applications and COM utilities that can be used immediately in any Microsoft.NET

### **SAP WebAS**

SAP WebAS is a solution out of one hand - SAP AG's. The implementation of the SAP kernel for each supported platform is done by SAP itself and therefore guarantees utmost fidelity to specification. The SAP R/3 kernel, which is exactly the same kernel as the one used by the WebAS, has been steadily enhanced since its first publication in 1993. Since SAP release 4.0, its feature-rich kernel, which also includes the ABAP compiler and ABAP runtime environment, can be regarded as stable. The only significant addition to the kernel has been the HTTP engine, along with the web development features.

On the negative side, there are mainly licensing issues. The current license scheme appears to be relatively high for special developments, so the Web AS investment is definitely out of range for smaller enterprise applications. Outside of the SAP world, the know-how of the development environment is poor. While training people in ABAP is comparably easy, it is still expensive to find good ABAP developers.

SAP WebAS has outstanding features that had been visionary when they appeared the first time in R/3 and that are still avant-garde in enterprise computing. The advantage of WebAS compared to the competing frameworks is the full integration of the development environment, repository and data dictionary with the runtime framework.

Because everything is integrated and interwoven, a developer that has once become familiar with the environment gains easy access to all development related information without having to refer to third-hand documentation. Once you have opened the source code of an ABAP program, you can easily click your way through the complete repository. This saves time and speeds up the development process. But there is more: this

encourages the developer to explore the environment, which leads to better acquaintance with the program and in turn leads to better, more reliable programs.

There is one thing to mention: all the arguments in favour of WebAS are related to the classical kernel, not the SAP J2EE engine which comes as a sidecar to the WebAS, but is not integrated into the transaction-based environment. The SAP J2EE engine, designed for use with certain mySAP applications, is a pure J2EE engine that deliberately neglects the best features of the traditional SAP R/3 kernel.

## Making Practical Decisions

When writing the first part of this white paper, I was consulting with an SAP implementation partner, who had to choose the right development platform for a new production controller central (PLC). The choice was to be made between a Java-based application server (IBM WebSphere Application Server, WAS), a Microsoft centric solution relying on Microsoft.NET, and SAP WebAS as the development platform. If that firm were to choose SAP WebAS, the additional question was whether to choose the classical ABAP environment or the new, long-awaited SAP J2EE engine to come out with SAP WebAS 6.20.

Microsoft.NET proved to be an equivalent competitor to J2EE and WebAS with respect to stability and ease of programming. In terms of connectivity and integration of peripherals, there is nothing that comes even close to Microsoft.NET due to the overwhelming supply of drivers and low-level APIs through the underlying WINDOWS operating system. Indeed, neither J2EE nor SAP WebAS have generic support for peripheral devices, not even for such common interfaces like the RS-232 serial connector. Both need to rely on native interfaces provided by third party suppliers.

As one understands from many comparisons, both Microsoft.NET and J2EE run head to head. Following very similar technical approaches, both camps accuse each other of copying the other's technology or architecture. e.g. the Java community continues to blame Microsoft.NET for copying their concept of the virtual machine, while Microsoft conversely remarks that the component model architecture of Java is a pure copy of Microsoft's decade-old COM technology. As a software engineer, I cannot give a clear preference to either one of the two technologies. In some aspects J2EE has some fine advantages, in other ones Microsoft.NET makes the a better case.

In this recent platform decision, we ruled out Microsoft.NET for purely economic reasons. Platform restriction would have required additional training of the hardware teams, because they were already educated on Linux. Together with the licensing costs and risk, the decision fell against Microsoft.NET. Technically, the decision was rather even.

So we had to decide whether we wanted to follow the mainstream and change horses from SAP and ABAP to J2EE, or to find out whether SAP's classical development environment is good enough, or even better, than the new stuff. From the way the latter phrase is formulated, you can already guess what we determined: that the belief that Java and J2EE is a superior and more modern development environment than what existed before does not hold proof when the actual performance and power is tested. Surprisingly for some, WebAS falls nothing short of J2EE in any one area and beats J2EE in some essential aspects.

The deciding argument in favor of SAP WebAS has been the efficiency in writing new code, counting the time from the vague design, repeatedly improved prototypes to the full deployment in a productive environment. Here it proved to be an enormous advantage that SAP is still the only high end solution that provides for a fully integration of source code compiler, runtime APIs and runtime debugger. \*\*\* But Axel, if a company chooses the Web AS platform with ABAP, what happens when the SAP J2EE engine comes out with 6.2? Will all the advantages of the many years of ABAP development be lost then? a comment or two is in order.

## Conclusion: Author's Choice

All three frameworks represent modern software development tools and runtime environments for professional business development. Microsoft.NET and J2EE are very similar in concept and design, where Microsoft.NET compensates the non-existing portability with a wealth of features and seamless integration of most elder Windows-COM+ components. Neither integrate a database and a repository as part of their virtual machine, which may be an essential disadvantage when it comes to creating reliable and complex applications quickly, and in addition, raises the learning curve sharply. .

WebAS differs from this architecture mainly by integrating a database layer in its virtual ABAP machine that simply shields any database connectivity from the application developer, making the underlying database fully transparent to the developer.

Another substantial difference is that any SAP program runs inherently within a transactional container, taking away the need to add any special code for transaction management from the developer. Microsoft.NET and J2EE both require the developer to explicitly demark the transaction boundaries. In an effort to standardize the development and deployment of transactional component, J2EE introduced the concept of Java beans that wrap the actual program in a transactional container forming the transaction boundary.

The difference between all three frameworks are mostly minor, and especially Microsoft.NET and J2EE are so similar in design and architecture that any decision between these two should be influenced by a thorough cost calculation, looking mainly on hardware, licensing and education costs. WebAS goes a different path in some crucial sectors, mainly with database integration, transaction management and deployment technology. These differences will be the decisive factor pro or against WebAS.

Microsoft.NET is an option for smaller companies and applications that require special solutions. Typical environments that point to Microsoft.NET are those attaching peripherals that support Windows drivers only, or those that want to make use of existing COM solutions, e.g. printing form with dedicated tools like Crystal Reports.

J2EE and Java are en vogue, but the arguments in favour of J2EE are not often convincing. Unless you are a software manufacturer, the portability argument is not persuasive. A decision for J2EE would mainly mean a decision in favour of IBM WebSphere, the only enterprise server that bases all developments on J2EE.

WebAS holds up well compared with the two other popular development frameworks. My personal choice is clearly in favor of SAP WebAS and ABAP for new, enterprise developments, both for web oriented and traditional server applications. \*\*\* Again, Axel, will this change with the release of the latest WebAs with SAP J2EE? It's not like you can stay on version 6.1 forever.....

Readers who want to see an illustrated breakdown of the three frameworks, should continue on to the appendix.

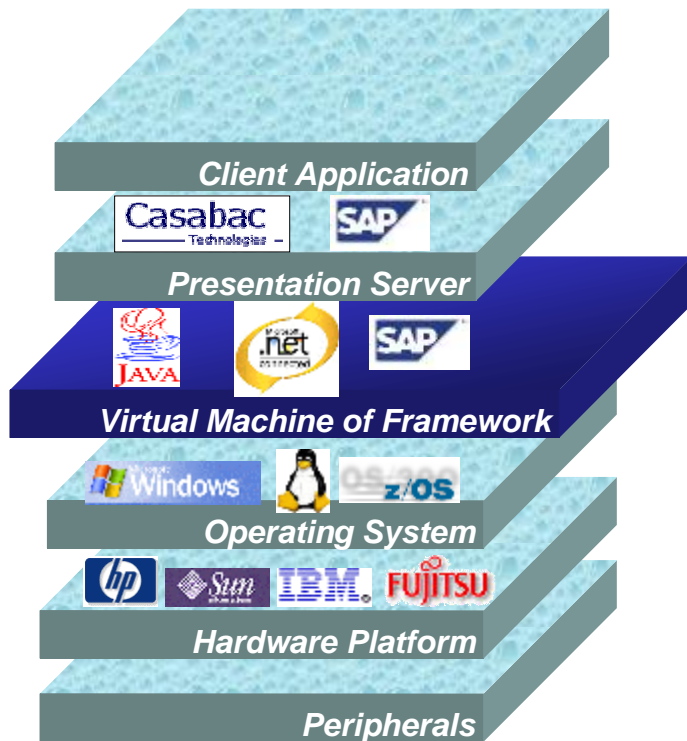
Axel Angeli is a senior SAP and EAI advisor and principal of logosworld.com, a German-based enterprise specializing in coaching SAP and EAI project teams and advising IT management on implementation issues. Axel has been in the IT business since 1984, and throughout his career, he has always worked with cutting edge technologies. Axel's SAP experience stems back from the good old R/2 days, and he is an expert on SAP's Netweaver technology and any kind of ABAP development. A speaker of several languages, Axel specializes in coaching and leading large multi-national teams on complex projects with heterogeneous platforms and international rollouts. Known for his intensive and successful trouble-shooting experience, Axel has been nicknamed by his colleagues as the "Red Adair" of SAP projects. He is the author of the best-selling tutorial "The SAP R/3 Guide to EDI, IDocs, ALE and Interfaces."

**Appendix: An**

**Illustrated View of WebAs, .NET, and Java Functionality**

**Figure 5: How Virtual Machines Fit into Application-Machine Integration**

## Virtual Machine Layer



**Figure 4: Frameworks are a New Layer in Application-Machine Interaction**

.NET, Java, and SAP WebAS Functionality Comparison

	WebAS	.NET	J2EE
<b>Client application</b>	SAPGUI Browser any DCOM client Java applets	Windows Browser any DCOM client Java applets	Swing Browser any DCOM client Java applets
<b>Presentation server</b>	WebDynpro		Casabac et.al.
<b>Virtual Machine</b>	ABAP	CLR	JVM
<b>Component Model</b>	BAPI	.NET Component DCOM/COM+	Java Beans
<b>Debug and IDE</b>	ABAP Workbench	Visual Studio Live debug of deployed apps only via remote	Eclipse, NetBeans Live debug of deployed apps only via remote (implementation dependent)
<b>Just-in-Time Compiler</b>	Yes Tracks source code changes	Limited only from MS.NET intermediate language	Limited only from Java binaries
<b>Transaction manager</b>	WebAS R/3 kernel) Transactions are inherent	CLR Common Lang. Runtime (MTS, COM+)	JTA Java Transaction API JTS Java Transaction Service
<b>Programming</b>	ABAP IV ABAP Objects Java	C# Visual Basic J# et.al.	Java
<b>Virtual Machine</b>	ABAP	CLR Common Lang. Runtime	JVM Java Virtual Machine
<b>Database</b>	Integrated in kernel Database is part of kernel	ODBC, SQL Database not part of kernel	JDBC, SQL Database not part of kernel
<b>Deployment Tracking</b>	STMS Integrated tracking and consolidation of source code and DDIC changes between all systems	Packages No automated consolidation	Packages No automated consolidation
<b>Platform</b>	Many Windows Linux Unix zSeries (/390) pSeries (AS/400)	Windows	Many Windows Linux Unix zSeries (/390) pSeries (AS/400)

Figure 5: WebAS, J2EE and Microsoft.NET Compared

Axel's Critical Evaluation of .NET, J2EE, and SAP WebAS Features

Requirement	.NET	J2EE	SAP (WebAS)
Legend: <b>P</b> = poor <b>PP</b> = good, but limited <b>PPP</b> = excellent      n/a = not applicable			
<b>Hardware neutral</b>	n/a	<b>PP</b>	<b>PPP</b>
A virtual machine independent of the hardware platform	CLR – Common Language Runtime  Currently, the Microsoft suites do support INTEL processors of several kinds; rumors say that there will soon be versions for the Apple Macintosh	JVM – Java Virtual Machine  The JVM is usually adapted by the hardware or OS manufacturer. This leads to heavy differences in the quality of the JVM depending on the implementation.	ABAP Runtime  SAP delivers its virtual machine since 1993 on different platforms. Since 1998 (release 4.0) the VM is stable on most hardware platforms.
<b>OS neutral</b>	n/a	<b>PP</b>	<b>PPP</b>
A virtual machine independent of the operating system	.NET is currently ultimately dependent on the underlying Windows OS;	J2EE has implementations for Windows, Linux, UNIX and many mainframe processors;	SAP still supports the widest range of OS
<b>Transaction manager</b>	<b>PP</b>	<b>P</b>	<b>PPP</b>
A sophisticated and transparent transaction management	.NET inherently supports transaction by means of the built in MTS (Microsoft transaction server).	J2EE supports transaction via Java beans.	In SAP, every action is a transaction by default.
<b>Deployment</b>	n/a	n/a	<b>PPP</b>
An integrated deployment monitoring and modification tracking	J2EE and Microsoft.NET only provide semi-automatic deployment management via install packages.		The reputed SAP transport management system records every modification within the development environment and allows a 100% track of modifications and deployments.
<b>Change Management</b>	n/a	n/a	<b>PPP</b>
Standardized and integrated change and version management	J2EE and Microsoft.NET leave the change management to external tools or third parties. Anything but one unique standard in change management is considered a disaster, because it will not provide quality results.		SAP's well-regarded transport management system that is responsible for deployment records and also any modification made to the production system guarantees a serialized and consolidated synchronization of development and production.

Requirement	.NET	J2EE	SAP (WebAS)
Legend: <b>P</b> = poor	<b>PP</b> = good, but limited	<b>PPP</b> = excellent	n/a = not applicable/available
<b>A just-in-time (JIT) source code compiler</b>	<b>P</b> <p>.NET deploys intermediate code only, not source code, so there is no JIT compiler available.</p> <p>Note: .NET compiles its intermediate language (MSIL) (but not source code) into native code and calls this JIT compiler</p>	<b>P</b> <p>Java deploys intermediate code only, not source code, so there is no JIT compiler available. Deployed p-Code, however, is accessible as soon it is copied.</p> <p>Note: Java can compile Java byte code (but not source code) into native code and calls this the JIT compiler.</p>	<b>PPP</b> <p>When you modify a source or a data dictionary element, the object is automatically compiled during runtime. Because deployment of repository objects means the deployment of the source code, this works also in production systems.</p>
<b>Programming language</b>	<b>PPP</b>	<b>P</b>	<b>PP</b>
A full featured programming language	.NET supports several languages, amongst them are Visual Basic, the Java- like C# and the Java clone J#.	Java supports the Java language only -this is considered problematic by those used to more readable and ubiquitous programming languages.	SAP supports only ABAP, which is still a highly readable and easy to learn language. WebAS also supports Java on its own J2EE, but this is a development parallel to the SAP kernel.
<b>API support</b>	<b>PP</b>	<b>P</b>	<b>PP</b>
Platform-independent APIs for database and operating system access (“one face to the developer”)	The API features of Microsoft.NET are the most complete -small wonder, since it runs on Windows only.	Java supports the JNI, Java Native Interface which allows access to all operating system supported components.	WebAS can call any operating system command line utility via its RFC interface.
<b>Data Dictionary</b>	n/a	n/a	<b>PPP</b>
A real time updated data dictionary and development repository	.NET does not have an integrated data dictionary nor does Visual Studio weave source code with data dictionary elements	Java does not have an integrated data dictionary, nor does any of the major editing solutions build a meta dictionary	Database is integrated in the SAP kernel and completely shielded by it; SAP keeps a sophisticated repository and data dictionary with real-time updated cross references; <p>as a side effect this concept allows a unified view on DDIC objects from within SAP independent of the database engine</p>

**Figure 6: Synopsis of Virtual Machine Properties**