



Re-Published: 05-Nov. 2005

## *Is the .NET Framework Ready to Challenge Java and J2EE for Enterprise Application Development?*

by James Pinpin, Director of Software Development

### *Introduction*

Although the underlying approaches are different, it should come as no surprise that .NET concepts and features are uncannily similar to Java™ and J2EE. Being a significant number of years ahead of .NET, the Java language and J2EE platform have already proven to be robust and versatile for developing and deploying enterprise-caliber applications. So considering the amount of time that .NET has been around, is it now ready for primetime enterprise application development? Specifically, have .NET's designers learned from the Java/J2EE world's struggle with the data access issue?

### How Do the Two Technologies Stack Up?

Java and J2EE provide a common programming language and development platform for various operating systems. On the other hand, .NET provides a common framework for many programming language but specifically for the Windows operating system. (Writer's note: .NET's approach has widened with the .NET specification becoming available to the open source community. For example, open source offerings such as Mono are now available that allow for .NET to run on Linux and UNIX.)

It's interesting that these fundamentally different approaches resulted in very similar architectures, concepts, and features. The following table compares some of the key language and core framework features that Java and .NET provide. Arguably, there are differences in the implementation details between the two, but the overall concepts, goals and benefits are quite parallel.

**Table 1 – Comparison of Java and .NET language and framework features**

Concept/Feature	J2EE	Java
<b>Interpreted and Intermediate Code</b>	Byte Code	Intermediate Language (IL)
<b>Runtime Environment</b>	JVM with HotSpot Compiler	Common Language Runtime (CLR) with Just in Time (JIT) Compiler
<b>Memory Management</b>	JVM Managed Heap with Garbage Collection	CLR Managed Heap with Garbage Collection

<b>Foundation Classes</b>	Java Base Classes	.NET Base Classes
<b>Packaging and Deployment</b>	Java Archives (.jar files)	.NET Assemblies (.dll or .exe files)
<b>Legacy Code Integration</b>	JNI	Managed/Unmanaged Code
<b>Interface/Design Philosophy</b>	Object-Oriented	Component-Oriented
<b>Interception</b>	AOP	.NET Component Services

A comparison of the components and technologies of J2EE and .NET reveals the same parallelism, suggesting that .NET is poised to take on enterprise application development and challenge Java/J2EE in the marketplace.

**Table 2 – Comparison of J2EE and .NET enterprise technologies**

<b>Component/Technology</b>	<b>J2EE</b>	<b>.NET</b>
<b>Web Services</b>	JAXP, JAXR, JAX-RPC, SAAJ	.NET Web Services
<b>Web Presentation</b>	JSP, JSF, Servlet	ASP.NET
<b>Business Components</b>	EJB	ObjectSpaces, ADO.NET
<b>Distributed Invocation</b>	RMI/IIOP	.NET Remoting
<b>Messaging</b>	JMS, JavaMail	Microsoft Message Queuing (MSMQ), BizTalk, .NET Passport
<b>EIS Connectivity</b>	JCA	.NET Enterprise Services
<b>Transactions</b>	JTA	.NET Enterprise Services
<b>Database Connectivity</b>	JDBC	ADO.NET, SQL Server
<b>Security</b>	J2EE Security, JAAS	.NET Security, ASP.NET Security

Adhering to the community process no doubt lengthened the amount of time it took to build out the complete J2EE specification. .NET has the advantage (and corresponding penalty) of being molded from a unified vision by technologists from a single company. In theory, this should make .NET's evolution more focused and efficient. As J2EE matures, only time will tell whether .NET will catch and even surpass J2EE.

## An Important Unresolved Java/J2EE Issue—Data Access

Of all the technology standards with which the Java community has struggled, the specification of the persistent data layer has met with the least success. Specifically, data object APIs and corresponding data services have been difficult to nail down.

The long list of data layer “standards” – JDBC, EJB, JDO, POJOs, Hibernate, AOP (Aspect-Oriented Programming), and various vendor-specific data objects (including our PDOs) – evidences the need and desire to address this issue. Even the EJB expert group has conceded that much work remains to be done with their proposed massive revamp of the EJB specification (version 3.0).

So what lessons has J2EE taught us about the data layer for enterprise applications?

- **The data layer API must be simple.**
- **The data layer API must be object-oriented.**

These seem like straightforward requirements. After decades of object-oriented development evangelization, it's amazing how the data layer has continued to lag on this. Java developers continue to default to JDBC as the foremost way to code the data layer. This is sufficient for an application with ten data classes, but what if an application has a hundred data classes? In contrast, the EJB specification supported object-oriented development but introduced too much complexity.

- **The data model must be reusable among applications.**

There are significant gains in being able to expose the data layer in a service-oriented architecture where a common data layer interface can be reused across an application suite. This can be easily accomplished with a model driven development approach by capturing the data model as an enterprise object model. Automated code generation or manipulation can then be leveraged for further wins in development time-to-market and code quality.

- **The data objects must be lightweight.**
- **The data layer must provide highly-performance data access.**

Enterprise applications have challenging requirements in terms of performance and scalability. Imposing remote distribution and fine-grained security requirements on data objects has turned out to be too complex during development and too expensive in terms of performance. Most application architects prefer the data objects to remain in the server tier behind web components or distributed business/session objects.

- **The data layer API must provide a flexible but powerful query language.**

It's interesting how each standard has put forth its own QL specification – SQL, OQL, EJBQL, JDOQL, XQuery, and so on. Maybe the specification experts should just keep reusing the proven ones and have the tool vendor's deal with various data source specific ways of forming a query.

- **The data layer should encapsulate or integrate seamlessly the underlying core data services.**

To keep APIs simple externally, the complexity must be provided internally. The data layer API needs to continue to provide core data services such as persistence, transactions, and legacy connectivity. Advanced data services such as caching and synchronization can help simplify application code if kept under the covers.

- **The data layer should support an industrial-grade data source.**

For example, providing a concrete mapping to relational databases is a must. This was one of the early knocks against JDO in its attempt to be a backend agnostic specification. Other data sources such as XML and in-memory databases are providing very important functions with regards to data integration and performance-oriented deployments. However, relational databases remain the backbone for enterprise applications dealing with mission-critical data. So they cannot be ignored.

- **The data layer should support complex distributed deployment configurations.**

In large organizations, data often resides among multiple database servers deployed in multiple data centers. A sophisticated data layer must be able to handle such complex deployment scenarios. For example, abstracting the database connection from the data access classes allows you to switch the data source without changing any application code. This prevents vendor lock-in and simplifies migration. In addition, built-in support for failover and high availability becomes more important in such highly distributed architectures.

## Will .NET Get Data Access Right?

So our prediction is that where Java/J2EE has stumbled, .NET may struggle as well. .NET's API offerings for the data layer are ADO.NET and the upcoming ObjectSpaces. How do these two stack up with respect to the data access issue for applications with complex object models and data intensive requirements? In our opinion: not very well.

ADO.NET is not an object solution; it forces you to deal with row sets. The only information available on ObjectSpaces indicates that it will provide an O-R mapping solution, but offers no details on what the object interfaces will look like or whether they will meet the criteria listed above. See the second article in this newsletter for a deeper comparison of these technologies

## Conclusion

The .NET framework compares well with Java and J2EE from a high-level features check box perspective. .NET also continues to make huge strides in filling its enterprise toolbox to allow its architects and developers to build more complex and demanding applications. And, .NET has the luxury of learning from Java/J2EE's missteps and successes.

A number of our key customers are considering .NET for their next set of projects, especially to take advantage of the low cost of PC servers, indicating a growing acceptance of .NET's capabilities. There seems to be no doubt that .NET will be a successful, ubiquitous development framework, the only question is how soon?

However, we believe that similar to J2EE,.NET applications with complex object models, high request rates, or both will need more than the "standard" solution for data access. O-R mapping and integrated caching can provide the tools to build such applications quickly and enable them to achieve enterprise performance and scalability.

In lock-step with our customers' interest in building new front-office applications with .NET, we have responded with Progress® DataXtend™ CE for C#, the newest member of our product family, which supports application development on the .NET framework. We look forward to providing our customers with the same benefits and successes that we've shared within Java/J2EE.



Real Time Division

[www.progress.com/realtime](http://www.progress.com/realtime)

### Worldwide and North American Headquarters

Progress Real Time Division, 14 Oak Park, Bedford, MA 01730 USA Tel: +1 781 280 4000

### UK and Northern Ireland

Progress Real Time Division, 210 Bath Road, Slough, Berkshire, SL1 3XE England Tel: +44 1753 216 300

### Central Europe

Progress Real Time Division, Konrad-Adenauer-Str. 13, 50996 Köln, Germany Tel: +49 6171 981 127

### France

Progress Real Time Division, 3 Place de Saverne, Les Renardières B, 92901 Paris la Défense Tel: +33 1 41 16 16 56

© 2005 Progress Software Corporation. All rights reserved. Progress, DataXtend CE and PowerTier are trademarks or registered trademarks of Progress Software Corporation in the U.S. and other countries. *Java and all Java-based marks are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. and other countries.* Any other trademarks or service marks contained herein are the property of their respective owners. Specifications subject to change without notice. Visit [www.progress.com/realtime](http://www.progress.com/realtime) for more information.