



.Net Support Expands

Integration & Development Strategies, Application Delivery Strategies

Thomas Murphy

Microsoft's .Net has seen strong uptake in its first year on the market, and we expect that to continue as Version 1.1 of the .Net Framework and an expanding number of third-party offerings hit the market.

The majority of large enterprises will have both Java 2 Enterprise Edition (J2EE) and .Net development projects (see ADS Delta 1147). We are beginning to see applications in deployment that utilize both technologies, with Web services as a bridge. Although we expect most developers to rely on Microsoft for tools to target .Net, third-party tools provide various features that may benefit IT organization efforts to improve quality or support cross-platform efforts.

During 2003/04, third-party language extensions to Visual Studio and standalone integrated development environments (IDEs) will begin to gain traction in the market and represent 15%-20% of the .Net IDE marketplace (see Figure 1). This will primarily be gradual market growth, since the majority of organizations maintain Visual Studio subscriptions to access specific product integrations and language features (e.g., Visual Basic). During this time, standards for Web services will mature, enabling the technology to play a critical role in cross-enterprise integration efforts. In addition, the continuing integration of life-cycle management facilities to IDEs will provide dramatically improved developer team coordination (see ADS Delta 1129), thereby increasing productivity and software quality. By 2005, development suites will compete as either part of an application platform or an enabler of cross-platform support. Such competition will cause continued market consolidation and vendors will extend offerings through frameworks as they take on model-driven approaches to development. By 2006/07, the broad use of Web services will demand improved software quality, and organizations that fail to adjust processes prior to this time will have difficulties exploiting the technology's full benefits.

Supporting .Net can be accomplished by third parties in two ways. First, the mechanism used by most language vendors has been to utilize Visual Studio integration. Visual Studio provides the key development facilities and the third party supplies the language-specific editing behavior, compiler, and any needed tools or libraries. In this form, more than 10 additional languages are supported within .Net. Second, the alternative to integrating with Visual Studio is creating a complete standalone environment. Borland's recently announced C#Builder will be the first major effort to create such an environment. Although the extension products certainly broaden the capabilities of .Net, we believe the addition of Borland's complete environment will have further value by creating a viable competitor to Microsoft and causing each product to improve.

A key value of .Net is the ability to utilize different languages, each of which may bring constructs that simplify coding for specific problems. In addition, support for legacy languages provides options for companies to migrate legacy code to the .Net platform. Both Fujitsu and Micro Focus are offering COBOL compilers for .Net with the intent to enable reuse and migration of legacy code. There are several limitations to code migration, including the lack of support for CICS and the need to re-engineer code that has embedded screen sections. However, legacy access provides the best option for immediate reuse.

Currently, additional language support focuses on scripting languages (e.g., Perl, Python from ActiveState). Such languages are primarily valuable for use in ASP (Active Server Pages) .Net

META Trend: During 2003/04, .Net will gain market share and acceptance as a competing enterprise platform to J2EE. During 2004/05, open source application servers will gain share in the J2EE market, forcing further vendor consolidation. By 2006, Linux will become the preferred platform for J2EE execution. By 2006/07, focus will shift from basic infrastructure to business frameworks, reducing the distinction between large middleware infrastructure and packaged application vendors.

because of the ability to work with regular expressions and simplify the processing and formatting of dynamic data streams.

One of the earliest supporters of .Net and its common language runtime has been Bertrand Meyer and his Eiffel Software. Eiffel Envision is a good example of the value that Visual Studio extensions can bring to IT teams. Envision provides Eiffel language support to .Net, and this is the only language currently available that supports multiple inheritance and generics. Eiffel also directly supports Design by Contract (see Figure 2) through its integrated support for assertions. Because .Net allows applications to utilize mixed languages (i.e., applications can include portions written in Visual Basic and C#, and a class written in one language may be extended in another), users can utilize Eiffel's assertion system to build wrappers around existing .Net assemblies to support Design by Contract. As organizations seek to drive reuse and build reliable Web services, assertions and Design by Contract provide valuable ways to ensure they meet requirements.

Borland C#Builder. Borland will be the first vendor in the market to ship a full IDE competing with Visual Studio. This is a key element in Borland's strategy to ship products that support deeply integrated and automated life-cycle management, and to be a platform-neutral provider of tools. Because the majority of enterprises will have mixed portfolios utilizing both Java and .Net, the availability of tools that support both platforms will be valuable to driving overall reuse and productivity goals. Borland provides a number of significant features that should appeal to organizations with mixed platform strategies. In particular, the product supports interoperability with CORBA and Java components by adding IOP support to .Net with its Janeva technology. In addition, C#Builder adds data access to Oracle, DB2, SQL Server, and InterBase with native drivers instead of the generic ODBC bridge. Microsoft is focused wholly on supporting its platform, and while support for other databases and technologies may be provided through various third parties, having an integrated package like C#Builder will be valuable to most enterprises. Borland's product will also provide full life-cycle support, including UML modeling (through its Together products), version control and configuration management via StarTeam, requirements with CaliberRM, and profiling and performance tuning with Optimizeit. Currently, Borland has much stronger tools for the application life cycle than Microsoft, and better integration than either Microsoft or IBM. This new competition will be good for the market and will drive Microsoft's Visual Studio team. It will also put pressure on IBM to continue and expand .Net support through its Rational and WebSphere Application Developer products, and continue full life-cycle integration around its XDE product. C#Builder will support languages other than C#, with initial support for Visual Basic in the first release and the goal to allow .Net-compatible languages to plug in to the environment, as they do in Visual Studio. However, we believe the main audience for C#Builder will be organizations with Java developers using JBuilder that are looking for a similar environment for .Net development, and these developers will use C# because of its similarities to Java. By the end of the year, Borland will also be supporting .Net with its Delphi RAD environment.

A key element of Borland's product strategy is the support for an integrated development life cycle. By providing support for both J2EE and .Net, together with deep integration across all phases of development, Borland has created a compelling value. Its Together unit illustrated this value originally with Together/C++, providing continuous round-trip engineering. True long-term productivity is founded on the ability for software engineers to deliver quality software. Since it is impossible to ensure the quality or correctness of a piece of software without the specification, it is critical for developers to easily reference requirements, models, and other specifications during the development process. When life-cycle tools exist separately and are disjointed, they are often seen as a hindrance to productivity. This is why the leading tool vendors are expanding efforts to integrate across the development life cycle. Borland's ability to integrate best-in-class life-cycle tools with its C# and Java environments creates an excellent choice for IT organizations with mixed development. Microsoft's Visual Studio will retain its position for pure .Net shops and those focused on Visual Basic. We also expect IBM, through Rational, to add increased support for .Net, but we do not expect a full-featured IDE during the next year. However, a key IBM strength over both Microsoft and Borland is support for legacy integration through its WebSphere Host Integration solution set.

Bottom Line

IT organizations should investigate and use third-party tools for .Net to drive software process improvement and platform interoperability.

Business Impact: New technologies will boost developer productivity, but they must be matched with processes and metrics that ensure software quality.

Figure 1 — Third-Party Languages for .Net

A wide variety of languages is available or under development for .Net, including COBOL, FORTRAN, Smalltalk, and Pascal. Languages are being created in commercial and academic settings. In addition to languages, a broad number of components and libraries are also available. The following resources provide pointers to third-party languages and components available for .Net:

- www.gotdotnet.com/team/lang/
- www.componentsource.com/Marketplace/Default.asp?SC=VS

Source: META Group

Figure 2 — Design by Contract

Design by Contract is a concept introduced by Bertrand Meyer. The premise is that software correctness — and, therefore, quality — cannot be determined without understanding the relationships between a service and its clients unless a contract specifying each party’s rights and obligations is formally stated. Utilizing Design by Contract drives productivity by ensuring software correctness, and generates the following developer benefits:

- Improved understanding of object orientation and software construction
- A systematic approach to building bug-free software systems
- Like unit test-driven software, an effective framework for debugging, testing, and quality assurance
- A method for documenting software components
- Better understanding and control of the inheritance mechanism
- A technique for dealing with abnormal cases, leading to a safe and effective language construct for exception handling

Eiffel provides direct constructs in the language to support Design by Contract. The key elements involve support for pre/postconditions. Preconditions specify the client’s or calling object’s obligations. This simplifies the actual implementation by isolating testing code from business logic. In Eiffel, these are presented by the “require” clauses. Postconditions specify the state that must be satisfied after the state change has occurred, ensuring the caller that the service has performed the job and left the system in a valid state. These are specified by the “ensure” clauses. Assertions are tied together with exception handling, and at runtime, the level of checking may be varied. For instance, a system may be tested for “require” failures, showing defects in the client or the “ensures” to identify supplier errors. Such checking may be turned off to improve performance once a system is validated.

```
insert (v: like item; pos: INTEGER) is
    -- Add 'v' at 'pos', moving subsequent items
    -- to the right.
require
    index_small_enough: pos <= count
    index_large_enough: pos >= 1
do
    if count + 1 > capacity then
        auto_resize (lower, count + 1)
    end
```

Source: META Group