Developing with Java Components

Maximizing Productivity in Java Application Development

July, 2001

A Sitraka White Paper

Contents

Management Summary

Much of software development follows an anachronistic approach that costs organizations time, money, quality and competitive advantage. Yet these same organizations are under increasing pressure to generate higher quality results in shorter time period using fewer resources. When developers build monolithic, customized applications from the ground up, they must rewrite code that has already appeared in other applications; as their responsibilities are extended over a range of functional and interface programming requirements, their level of expertise is invariably diluted.

Fear of the unfamiliar and of losing control over what is widely viewed as a highly specialized, even personalized process has made IT groups reluctant to use externally generated code. But a steady, gradual shift to Java component-based development is taking place as the development process is increasingly influenced by business production models and by market principles of efficiency, specialization and flexibility.

The benefits of adopting a component-based approach to development—shorter time to market, enhanced productivity, improved quality—are compelling. By using Java components, developers and their organizations have a unique opportunity to reduce costs while improving the quality of their client-side and server-side Java applications. When selecting Java components, therefore, it is important for organizations to seek out vendors with established reputations for quality, key industry partnerships, a range of products and comprehensive technical support.

Overview of Java and Java Components

"The use of components takes a lot of the busy work out of developing applications. Component use cuts in half the time it takes for a program to be developed."

—Mike Marzo, Senior Technology Architect, Goldman Sachs, InternetWeek, February 16, 2001

While the battle to establish standards in software development continues, Java is emerging as the dominant environment for e-business applications because it enables disparate technologies and platforms to communicate with each other. The Java environment—including the Java Language specification, the Java Virtual Machine and the Java API Core Class Library—was created with standardization in mind.

Readily capable of fitting into existing code, Java components can significantly reduce costs and improve productivity when developing business applications. Java applications, in turn, can be designed to run on a variety of platforms, including Windows, Solaris, Linux, HP-UX and AIX.

Java components typically fit into one of two categories: they provide technical functionality in the presentation layer of an application (e.g., SMTP for e-mail or enhanced user interfaces) or they solve sophisticated business issues in a business logic layer (e.g., credit card authorization for e-business applications).

Java components are usually connected to applications within a variety of integrated development environments (IDEs), such as Sun's Forté for Java, Borland's JBuilder, Macromedia/Allaire's JRun Studio, IBM's VisualAge for Java and WebGain's VisualCafé. Components are developed specifically to support the various IDEs.

JavaBeans—Originally designed for user-interface and client-side tasks, JavaBeans is a component architecture for the Java programming language. The "bean," or Java component, has the ability to tell a larger system about itself: both the actions it can take (events) and its properties (attributes). As a result, these components present a common face to applications. Developers can use them without having to know about their inner workings.

Enterprise JavaBeans—To allow for server-side component-based development, Sun released the Enterprise JavaBean (EJB) specification that adds server-side features such as transactions, persistence and scalability. While Java server-side components generally support any J2EE application server, they are often customized to run on specific application servers, such as BEA WebLogic, Apache Tomcat or IBM WebSphere.

Swing Toolkit—Introduced by Sun Microsystems in 1998, the "Swing" set of Graphical User Interface (GUI) components is part of the Java Foundation Class (JFC) library. The Swing toolkit is a collection of components—buttons, menu bars, dialog boxes, split panes, scroll bars and tables—that act as interface building blocks to help developers build GUIs for Java applications. While Swing offers basic GUI functionality, several important higher-level features such as sophisticated charting and graphing, data-intensive tables and database connectivity are missing.

Drivers of Component-based Development

"The market for IT products and services, once predicated on the elegance of the technology, is now driven primarily by business requirements."

—David Sprott, Open Market Components, a CBDi Forum Report, January 2000

Although reluctance to fully embracing the component-based approach exists in many organizations, it is diminishing. True standardized component integration, impossible before the appearance of object-oriented programming and Java technology, is now possible. The Java environment was created with standardization as a principal objective. Java objects are self-contained and are created according to set rules; they can be extracted and reused on virtually any machine that runs Java.

The strongest resistance to component-based development still comes from development teams concerned about losing control over the development process. Since application development can be as individual and idiosyncratic as the people who write it, developers are sometimes reluctant to accept externally written code. However, Java is already emerging as the language of choice for business. The Gartner Group estimates that more than 60 per cent of enterprises around the world will adopt Java applications and technology by 2004. This marks a widespread acceptance of a technology that is ideally suited to transform the very nature of software development.

Limitations of Procedural Programming

"The Enterprise JavaBeans component model and associated Java 2 Enterprise Edition standards will dominate the application server market and drive a potential market growth rate of almost 180% year to year."

—Mike Gilpin, Vice President and Research Leader, Giga Information Group in Information Week, December 18, 2000

Traditional applications are those written using procedural programming, an approach that views computing as a linear process. When developing applications to support multiple changing business processes, procedural programming is not as well suited as object-oriented programming for the following reasons:

Limited reusability of existing code—One of the main challenges that development managers face today is the pressure to build high-quality applications in shorter time periods using fewer resources, largely through increased developer productivity. Traditional application development entails lengthy development cycles because most lines of code must constantly be rewritten. Code is not reusable, so developers must write more code than they would were they to follow a component-based approach.

Limited application scalability—Monolithic applications are not readily scalable. When an organization's needs or business conditions change—industry regulations, organizational structure, information systems, financial models and so on—its applications must either adapt or be replaced. Procedural development methods often distribute functionality throughout an entire application, which makes assimilating changes a tedious matter of modifying every instance where functionality is affected—a poor use of development time.

Too much time spent on infrastructure programming—The procedural approach to development forces developers to spend inordinate amounts of time on infrastructure programming—writing code for low-level repetitive tasks, such as opening files, loading, moving and reformatting data, and so on. Their attention—and their expertise—is diverted away from the application's more customized requirements, which imposes extra costs on the organization.

Lower overall application quality—As with the practice of any highly technical and sophisticated skill, specialization leads to greater innovation and higher quality. However, the procedural approach often forces developers to write code for tasks outside of their areas of expertise; this "dilution" of skills can diminish the overall quality of an application.

Increasing Need for Flexible Applications

"As the concept of interchangeable parts fueled the industrial revolution a century ago, creating economies of scale and dramatic growth in productivity, this new capability (of using Java components) could put software production on track to achieve comparable gains in productivity and reliability."

—National Institute of Standards and Technology

Component use is essentially a form of outsourcing and can be understood in similar terms. The benefits that organizations derive from the outsourcing of non-strategic business processes—reduced costs, maintaining of focus on core competencies, increased speed, improved opportunities for innovation—are precisely those that can be expected from component-based application development.

Businesses must change as market demands change—Organizations are evolving more quickly than before, breaking geographical, industrial and organizational boundaries through expansions, mergers and alliances. They are continuously reinventing themselves in order to remain competitive. And they are focusing on flexibility and the capacity to adapt to changing business processes.

Business applications must adapt to new business processes—Today, organizations are focusing on technology as a tool that can be tailored to meet their existing and future needs. That is, it is now business itself—not technology—that is driving the market. Business processes dictate an application's architecture. It is no longer acceptable for a business to model its processes around a specific application.

New applications must adapt while integrating legacy systems—Object-oriented languages like Java, and the object-oriented development model in general, more closely resemble the way businesses work. While Java was not the first object-oriented language to appear, it was the first to be easy enough to use to make it highly practical. Business technology requirements—leveraging existing IT investments, integrating legacy systems and applications and reducing IT costs while increasing return on investment—are driving the push towards widespread use of Java components.

Rising Demand for Programming Efficiency

With object-oriented programming, sets of activities are grouped as discrete tasks, allowing the 'flow' of an application to follow whatever path is required by the activity. This is an approach that more closely resembles how human and business activity works. Component-based applications are "encapsulated": data and implementation are localized within the component so that variables and methods can be added, deleted or changed without affecting the services provided by the object. Applications built using components can evolve and adapt quite easily,

which enables developers to be more productive. And, as the organization grows and its needs evolve, existing applications can grow with it, dramatically raising the overall return on IT investments.

Accelerated application development—For application developers, time to market pressures mean earlier deployment dates and ever more condensed development cycles. As every software development manager knows, simply demanding more lines of code from developers in the same amount of time is not a viable option. Staffing up to meet demand is a possibility, but the worldwide shortage of skilled developers makes this costly and difficult. Using components dramatically reduces the amount of code that programmers have to write, which can accelerate the speed of application development.

Enhanced developer productivity and innovation—Because the use of components frees developers from having to spend time on infrastructure programming and repetitive tasks, it enables them to focus on the more specialized aspects of an application. Several benefits to productivity result:

- Developers can specialize and focus on the customized aspects of the application.
- Development teams can be concentrated on specific tasks according to experience and areas of expertise.
- Experienced C++ developers in organizations can make the transition to Java relatively quickly, which enables them to be productive after a shorter learning period than inexperienced developers.

Improved application quality—The rise of e-commerce has increased expectations of software performance and reliability. As dependence on Internet applications grows, the financial implications of unreliable code are amplified. Component-based development improves the quality of applications in the following ways:

- Java components developed by reliable vendors generally undergo a rigorous testing and debugging process resulting in a level of quality that is difficult to achieve in-house.
- A certain level of quality is inherent in the very nature of Java components. Each component is self-contained, having its own attributes and rules that prescribe how it interacts with other objects.
- The use of vendor-developed components allows in-house development teams to concentrate their own quality control efforts on the more specialized portions of the application.

Evaluating Java Component Vendors

Typically, in-house components cannot be produced as efficiently as prepackaged ones. As components are reused, so too is the knowledge that developers gain from using them, which saves time and money for the organization. Although the decision to purchase a certain type of Java component is driven largely by the specific requirements of an application, certain pivotal factors should be considered when evaluating them:

Integration—Development teams must ensure that the components they purchase integrate with the various elements of their technology environment, including web servers, application servers, integrated development environments (IDEs) and platforms.

Reliability—Once proper integration is confirmed, the components must function reliably within the larger business application in order to achieve higher performance.

Quality—Organizations seeking to purchase components should ensure that each vendor being evaluated tests its Java components rigorously before bringing them to market.

Reputation

Although it takes a great deal of experience and expertise to build reliable components, variances in quality exist between vendors and between specific components. To avoid problems, it is important to seek out vendors with strong reputations for consistently producing reliable products.

- Obtain input from developers familiar with the vendor through prior use of its products.
- Determine how long the vendor has been creating and delivering components.
- Read software industry publications, analyst briefings, case studies and customer testimonials to see how the vendor is reviewed and represented.
- Look for important industry awards obtained by the vendor.
- Where possible, research the financial stability of the vendor organization, as well as its future technology vision.

Partnerships

Leading component vendors form partnerships with major industry players, relationships that go beyond simple product integration and support. These relationships can result in the vendor receiving early access to updates as well as technical input and briefings. They can also result in testing cooperation to ensure compatibility with new technologies and collaboration in development initiatives. A vendor's Web site should include information about its key partnerships.

Breadth of Offerings

A host of vendors in the market offer only a single component. These one-product vendors may fill a very specific need, but their highly localized expertise means that they are poorly equipped to help an organization move beyond a very narrow and focused solution.

While specialization in components may indicate a certain level of expertise, vendors that offer additional products aimed at improving the overall development process should also be considered closely. Elements of consistency run across products developed by the same Java component vendor—elements related to reliability, design, and integration. Developers using components from a single source will experience a shorter learning curve as they master the component design and the methods for installation and use.

Technical Support and Documentation

The needs of each design team and development environment are unique. Customized technical support is therefore an important consideration for an organization to ensure that it is purchasing the right components and will be able to use those components effectively. Support should be available both prior to purchase—during the evaluation stage—and after a purchase.

Prior to purchase, prospective buyers should be able to download free evaluation versions of components accompanied by some form of support including FAQs, evaluation guides, online discussion groups and opportunities to contact technical support directly. After purchase, ongoing support is essential for ensuring maximum benefit from the product, including its future iterations.

Documentation provides developers with resources to help them better understand how components are constructed and how to install and use them. Tutorials, demos, examples of code, lists of data sources and properties, descriptions of features, customization procedures—all help developers gain the most from components in the least amount of time. Availability of documentation should be a key consideration in evaluating component vendors.

Sitraka JClass Components

Fully scalable to mission-critical development environments, JClass is the most comprehensive collection of integrated Java components available. JClass components cover a range of high-value GUI functionality including 2D and 3D charting, tables and grids, data connectivity, data input and validation, JAR optimization, GUI enhancements and layout and reporting.

The JClass family, which consists of nine client-side components and one server-side component, includes everything the professional Java developer needs to build powerful and flexible application interfaces.

JClass Client-side Components

JClass client-side components are available either individually or as part of the JClass Enterprise Suite or JClass Standard Suite. The JClass collection of client-side components includes the following products:

JClass Component	Description
JClass Chart	A 2D charting component, with multiple business and scientific chart types including bar, pie, area, line, plot, stacking, candle, HiLo and more
JClass PageLayout	A powerful component that embeds professional printing capabilities (like PDFs) into Java applications and provides complete control over page layout
JClass Chart 3D	A 3D charting component for building stunning, interactive charts with drill-down capability
JClass LiveTable	A powerful grid/table component for building sophisticated data driven tables and forms

JClass Elements	A comprehensive collection of GUI enhancements and extensions
JClass JarMaster	A Java archiving utility that significantly reduces download times by optimizing JARs before deployment
JClass Field	A collection of data-aware JavaBeans for providing data input and validation for a range of popular data types
JClass HiGrid	A unique grid for managing and displaying dynamic, hierarchical data, letting developers build data-bound GUIs in minutes
JClass DataSource	A hierarchical data source that binds automatically with any Java database connectivity (JDBC) or in-memory data object; works with JClass HiGrid, JClass LiveTable, JClass Chart and JClass Field

JClass ServerChart

JClass ServerChart is a server-side Java component that brings the power of data visualization from your Web server or application server to your entire network of users. With multiple scientific and business chart types, and support for rapid updates, JClass ServerChart gives users real-time access to business-critical information in a highly intuitive format.

Designed from the ground up for server-side development, JClass ServerChart provides extensibility into the enterprise by leveraging servlet and JSP technologies, application server integration and the option to operate in a headless environment (without a display). Charts are easy to maintain, requiring minimal developer resources.

All JClass components integrate with the latest platforms (e.g., Microsoft Windows, Sun Solaris, IBM AIX, HP-UX and RedHat Linux) and IDEs (e.g., Borland JBuilder, IBM VisualAge and Sun Forte for Java). In addition, JClass ServerChart integrates with leading application servers such as BEA WebLogic, IBM WebSphere and Apache Tomcat.

For a complete list of supported technologies, visit the Sitraka Web site at http://www.sitraka.com/jclass.

Conclusion

Sitraka's JClass family of Java components represents one of the broadest component lines offered by any Java vendor. This is important to organizations purchasing components for their reliability and integration, a shorter developer learning curve and the advantages to be gained from having an established relationship with the component vendor. Standardizing on multiple components from a single vendor enables organizations to equip their development teams with an integrated, broad set of tools, and at the same time spare their developers from having to repeatedly learn new programming techniques required to make the various components work.

Reputation—Sitraka is one of the few vendors entirely dedicated to developing solutions for Java. We released the first Java component in 1996, developed the first 100% Pure Java component in 1998 and were the first to release Java 2 compatible components. JClass was the first set of components to provide optimized support for Java 2 as well as the first to offer 3D charting that takes advantage of advanced technologies such as Java 3D and Open GL.

Partnerships—Sitraka has forged powerful alliances with the world's leading hardware and software vendors to ensure that our products integrate seamlessly with existing and emerging enterprise Java technologies. Among our valued partners are industry leaders including Sun Microsystems, IBM, HP and BEA.

Breadth of offerings—In addition to offering both client- and server-side Java components to maximize productivity in Java application development, Sitraka offers JProbe performance tuning tools and DeployDirector, a robust and scalable Java application deployment and management solution.

Technical support and documentation—All JClass components come complete with comprehensive online documentation, including general and installation information, a full product overview, API documentation and product manuals. JClass components also include Pre-sales Support and Gold Support with Subscription, providing a variety of online technical support materials as well as access to Sitraka's experienced and knowledgeable Technical Support Engineers in our offices in Toronto, Canada and Amsterdam, the Netherlands.

About Sitraka

Sitraka is a leading provider of reliable, innovative software solutions that enable enterprise IT organizations to leverage the power of the Internet. A proven leader in the Java marketplace, Sitraka delivers products and solutions that accelerate the development, deployment and management of Java-based e-business applications. Sitraka products include DeployDirector, a robust and scalable Java application deployment and management solution, JProbe performance tuning tools and JClass Java components. Sitraka has forged powerful alliances with key industry leaders including Sun Microsystems, IBM (as an Advanced Business Partner), HP and BEA to ensure that our products integrate seamlessly with the latest development environments and platforms. Sitraka products are sold and supported directly through Sitraka's North American headquarters in Toronto and European headquarters in Amsterdam, and through a global network of resellers. Visit Sitraka on the Web at www.sitraka.com.