OSGi and the Enterprise?

OSGi™ is the dynamic module system for Java™; but what is its relevance to the modern enterprise?

To answer this question, reflect for a moment upon the challenges most likely faced by your own organization:

- Do your customers demand highly reliable, agile, secure and low-cost business systems?
- Is your environment able to efficiently scale to meet unpredictable volumes caused by increasingly volatile markets?
- Are you able to quickly adapt business systems to meet new market opportunities?
- Is on-going fragmentation of your business value chain a challenge for your legacy stove-piped business processes?
- Is the operational environment volatile? Short term volatility caused by infrastructure failures, software faults and operational errors. Long term volatility driven by mergers and acquisitions, frequent strategy changes, waves of off-shoring, on-shoring, outsourcing and in-sourcing initiatives?
- Despite numerous initiatives, are operational maintenance costs continuing to increase?

As will be shown, application modularity is the keystone required to successfully address these issues. If your business systems are Java based, then OSGi technology will be central to this strategy.
The Cost of Complexity – The Harsh Realities

Large organizations have IT systems that comprise of thousands of applications: these highly complex environments comprising of a large number of tightly coupled network components running 10s or even 100s of millions of lines of code.

Such environments are heterogeneous and volatile. New green field applications run side-by-side with applications that have been in operational use for 30-40 years. Applications become obsolete and need to be replaced (obsolete technology, end-of-life applications). Meanwhile it is not unusual for several thousand application changes a week to be driven by a continuous flow of new business requirements.

The economic cost of such IT complexity is hard to quantify but probably exorbitant. According to the Standish Group (http://www.standishgroup.com/), 66% of all IT projects either fail outright or take much longer to install than expected because of their complexity. For IT projects costing over $10 million apiece, 98% fall short of meeting their business objectives.

Meanwhile, a 2002 study by the National Institute of Standards and Technology (NIST - see http://www.nist.gov/index.html) reports that developers spend nearly 80% of their time fixing bugs. Accounting for this debugging overhead, a typical programmer writes between 8 and 20 lines of new code a day. As a result, developer productivity has plummeted. The NIST study concludes that software errors cost the U.S. economy an estimated $59.5 billion annually.

The Nature of Complexity

We live in a complex world; a world comprised of diverse ecological, political, commercial and social networks; each node within each network itself a complex system of interrelated parts.

We make sense of this complexity through the processes of encapsulation and abstraction. Via encapsulation we partition reality into distinct entities; i.e. we modularize reality. Via abstraction, we mask the internal complexities of these entities; instead referring to these entities by simple descriptions.

Figure 1: On average, amount of software in existence doubles every 7 years - see http://techdistrict.kirkk.com/2010/02/26/osgi-devcon-slides/
Supporting similar conclusions of this earlier NIST study, recent work by Anne Thomas Manes (a Senior Gartner Research Analyst), estimates that ongoing maintenance accounts for 92% of the total lifetime cost of each application: see slides 9 & 10 – SOA Symposium: Berlin, October 2010.

The conclusions are inescapable: if left unchecked, environmental complexity and operational costs will continue to spiral; if left unchecked, an organization's business systems will continue to deteriorate.
Modularity and Assembly

Modularity is the cure for complexity. Modularity localizes the impact of change, which directly leads to increased maintainability.

- In a composite system, knowledge is only required for the particular module being worked upon, along with its relationship to other modules within the system. These other modules may be treated as “black boxes” that perform specific functions, without worrying about their internal details.
- As long as the module boundaries are invariant, the impact of change is localized to each module and prevented from leaking into the wider system.
- By allowing parallel engineering teams to concurrently work on different modules within the same system, modularity directly facilitates efficiency increases in the development process.
- The deliverables from each development team can be independently versioned.
- Finally, as these modules perform a few, or a single function, they are much easier to exhaustively unit test: the working modules then simple assemble into a full application for integration testing.

An Old Idea

The concept of assembling a product from a set of well-defined re-usable components is not new. Indeed its roots can be traced back to at least 250 BC with emperor Qin Shi Huang and his commissioning of the Terracotta Army (see http://en.wikipedia.org/wiki/Assembly_line). Whatever the product, the driver for modularity and subsequent assembly is to; increase and maintain quality, reduce cost and increase output. The modern archetype for modularity and assembly is the automotive industry, where extensive use of standardization, modularity and assembly results in the mass production of affordable automobiles.
As complexity is an issue at all structural levels, so modularity must be apply at all structural levels (see Kirk Koernschild http://techdistrict.kirkk.com/2009/11/03/turtles-and-architecture/). This realization has, sometimes unknowingly, underpinned a number of recent technology trends including:

- Service Oriented Architecture (SOA) – The move from business systems rigidly coupled with proprietary protocols to “services” accessed via common protocols.
- Cloud Computing – The decoupling of applications from the underlying computer resources upon which they run, so allowing a more modular runtime environment.

Yet in each of the above transformations the underlying application portfolio remains untouched, each a rotting code-base that drives environmental complexity. To directly address environmental complexity one must also embrace application modularity.

Accidental Complexity?
The term ‘accidental complexity’ is frequently used but rarely adequately defined.

Accidental complexity can be simply defined as the difference between a given structure and the simplest possible alternative that interacts with the external world in exactly the same fashion.

Left unchecked, accidental complexity (unnecessary structure) accretes over time.
For Java™ based applications, OSGi™ is the industry standard for modularity and dynamic assembly. OSGi enables application modularity in two distinct ways:

- **‘SOA inside’** - OSGi Services interact with each other via local or distributed OSGi service registries in a dynamic manner. As with traditional coarse-grained Service Oriented Architectures, as long as the service interfaces are maintained, individual services may be rapidly evolve and refactored without affecting their environment; i.e. the local or remote services with which they interoperate. Hence monolithic applications may be broken into a number of Services, each of which may be independently maintained, enhanced and used outside its initial context.

- **Java Modularity** - OSGi also provides the software industries modularity standard for Java. Each OSGi Service may be comprised of one or more OSGi bundles. In this manner, common infrastructure or processing logic code may be encapsulated within a set of commonly used OSGi bundles and re-used across many OSGi Services.

By enabling industry standards based modularity, OSGi technology provides the necessary foundations for any business IT transformation program whose primary aim is to reduce environmental complexity, and so medium term application total cost of ownership.
OSGi – The Business Benefits

Having embarked upon an OSGi based application transformation strategy, which operational metrics indicate that the strategy is succeeding?

1. Developer Efficiency – Accelerated compile, test, run

For large monolithic applications it may not be possible to test changes in the developer’s local IDE, as compile times may be many hours. As a result, developers are forced to rely upon unit and integration tests that run during the nightly build cycle. This, in turn, causes bug detection and rectification cycles to take days, with an increased likelihood that some issues are not found and will leak into production.

In contrast, OSGi bundles tend to be small, cohesive and de-coupled, significantly reducing development and debug time.

2. Resource Utilization

Traditional monolithic applications lack information concerning required software libraries. In response, developers typically load every possible library into their IDE: driving development, User Acceptance Testing (UAT) and production memory requirements much higher than actually required.

However, once dependencies are understood and mechanisms are in place so that only required libraries are automatically loaded, the number of artifacts may be reduced by an order of magnitude, with corresponding reduction in required machine memory.

For an organization with several hundred developers, the cost savings are considerable: the potential to reduce memory footprint in production potentially resulting in even more substantial savings.

3. Developer Efficiency – Parallelism and agility

Developing a monolithic application requires multiple developers to concurrently work and test against the complete application code-base. In contrast, a modular system lends itself well to many hands being involved in its development and maintenance. It’s not necessary to
understand the whole system inside out, each individual can independently work on small well-defined and decoupled modules. This directly translates to increased project delivery success rates; this, as smaller well-contained projects have a higher success rate than larger projects. Hence OSGi is a natural technology partner to the agile development techniques adopted in recent years by many organizations.

With the appropriate organizational incentives in place to encourage re-use over code creation, further significant efficiencies are realized by cross development team re-use of OSGi Bundles and Services.

4. Product Delivery
By embracing modularity, by breaking large monolithic applications into composite applications comprised of a number of independent modules, an organization naturally moves away from high-risk waterfall product release processes towards lower risk, incremental feature releases.

5. Maintenance
From an ongoing maintenance perspective it is now possible to re-factor a composite application a module at a time, reversing design rot, increasing code re-use, and so systematically driving accidental complexity out of each application and, over time, the environment as a whole.

6. Enhanced IT Governance
Structural information is no longer locked away within key members of staff or out-of-date configurations; rather it is explicitly defined by OSGi metadata. Courtesy of this metadata, the structure of each composite application is known. It is therefore a simple task to answer questions like:

- Which software licenses are used?
- Which production applications are at risk from a third-party library with an identified security vulnerability?
The structure of a composite application may be rapidly mapped by any OSGi literate engineer, significantly reducing the amount of forensic work need to decipher and understand an applications code-base, so decreasing operation risk associated with loss of key development personal.

Finally, prior versions of an application may be rapidly re-constituted, offering great value to those organizations that are required to validate the past behavior of a business system to a regulatory authority.

7. Reducing Operational Risk While Increasing Business Agility
As the internal structure of monolithic applications is poorly understood, upgrades are complex and high risk. This coupled with the long developer diagnostic/fix cycles, results in production outages and instabilities that may span several working days or even weeks. Operations respond to this challenge by maintaining a number of isolated horizontal production silos, attempting to ensure service availability by releasing new software, one silo at a time: so increasing operational management, compute hardware and data-center real-estate costs.

Yet stability and agility are closely related concerns. The more agile the business service, the easier it is to change from one well-defined state to the next; whether this be to introduce new business functionality, applying a patch, or rolling back to a previously well-known functioning state.

With an advanced OSGi based runtime, each application is self-describing; meaning the dependencies between versioned modules and dependencies on runtime infrastructure are known. A business system and any associated runtime middleware services may be dynamically deployed in seconds. Subsequent enhancements may be applied and removed just as rapidly: so significantly increasing business agility, decreasing operational risk and operational expense.
Modular Systems are Maintainable Systems
As identified by James Governor in his article “The Rise of the Stackless Stack” (see http://www.redmonk.com/jgovernor/2008/02/05/osgi-and-the-rise-of-the-stackless-stack-just-in-time/), current industry trends are collectively shifting away from rigidly coupled, static, opaque environments towards adaptive, loosely coupled systems which are dynamically assembled from well-defined software components that run across a fluid set of compute resources. Excessive application maintenance costs will ensure that this trend continues into the foreseeable future.
Sooner (for those that seek significant competitive advantage) or later (for those that find that operational issues are no longer bearable), ‘necessity’ - the mother of all invention - will drive organizations towards modular, dynamic assembled, applications and so towards OSGi, the industry standard for Java modularity.
About the OSGi Alliance
The OSGi Alliance is a worldwide consortium of technology innovators that advances a proven and mature process to assure interoperability of applications and services based on its component integration platform. The Alliance provides specifications, reference implementations, test suites and certification to foster a valuable cross-industry ecosystem. OSGi technology is delivered in many Fortune Global 500 company products and services. Member companies collaborate within an egalitarian, equitable and transparent environment and promote adoption of OSGi technology through business benefits, user experiences and forums. For more information on the non-profit technology corporation, visit http://www.osgi.org.

How to Reach Us
OSGi Alliance
Bishop Ranch 6
2400 Camino Ramon, Suite 375
San Ramon, CA 94583 USA
Phone: +1 (925) 275-6690
Fax: +1 (925) 275-6691
Email: help@osgi.org
Web: www.osgi.org

OSGi is a trademark of the OSGi Alliance in the United States, other countries, or both. Oracle and Java are registered trademarks of Oracle and/or its affiliates. All other marks are trademarks of their respective companies.

COPYRIGHT © 2011 OSGi Alliance. All Rights Reserved