INTERNATIONAL
Systems Group, Inc.

From Chaos to Order —
Delivering e-Business Integration Solutions

Overcoming the Limitations of J2EE for Financial Services Applications

International Systems Group (ISG), Inc.
32 Broadway, Ste 414
New York, NY 10004
tel: (212)489-0400
fax: (212)489-1125
web: www.isg-inc.com
email: isg@isg-inc.com

Financial Applications and Technology

Financial services have always been a powerful catalyst for technological advances. Seeking out a competitive advantage motivates—to a large degree—a financial services firms' engagement in new computer technology. The financial services industry spends more on IT, as a percentage of revenues, than any other industry.

As commercial Middleware products continue to mature and provide a more comprehensive framework than their predecessors, IT organizations find themselves under increasing pressure to deliver solutions at lower and more predictable cost, thus almost entirely eliminating the need to develop custom Middleware solutions and buy into standards, and commercial products.

Today technologies like J2EE-compliant Application Servers, XML, and relational databases are largely in place and widely accepted by the financial services industry as mature and production-ready.

The Promise of Application Server Technology

The advent of mature, production-ready J2EE-based Application Servers has made a strong impact on the way financial services applications are architected, developed and deployed.

Application Servers incorporate component model management, transactional integrity, persistence, remoting mechanisms, security, and other essential services, along with unified configuration and diagnostics tools. Together, they assure that the business application will be more robust, more maintainable, more open, and cheaper to construct and own than a custom solution.

In addition to development of new applications, Application Server technologies extend the solution to application integration as well via the J2EE Connector Architecture (JCA) standard. Finally, Application Servers provide an excellent vehicle to support such extended functionality as Web Portals and Business Process Management (BPM).

International Systems Group (ISG), Inc.
32 Broadway, Ste 414, New York, NY 10004
tel: (212)489-0400 • fax: (212)489-1125
web: www.isg-inc.com • email: isg@isg-inc.com

Trading Systems and Application Servers

Several business drivers have caused the Financial Services industry to embrace Application Server technology. The first is the real-time processing required to support Straight-Through Processing (STP) mandates. Another important business factor driving migration of the trading-related applications to Application Servers is the need to integrate the enterprise at the systems level for better risk information, more consistent reference data, and a more complete view of the customer.

J2EE has found broad recognition in financial institutions. The majority of enterprise architectures now incorporate Application Servers, and the technology has become a platform of choice for delivery of mission-critical, line-of-business applications.

However, while alleviating many essential computing and development problems, J2EE is not a panacea. Even in recent projects, where the latest releases of the major Application Servers were utilized, International Systems Group, Inc. (ISG) had to develop and integrate additional value-added J2EE frameworks. These frameworks either provide additional services not offered by the commercial Application Servers, or they replace some services that come with Application Servers with services that offer much richer functionality.

Examples of typical J2EE frameworks include centralized logging facilities, permission frameworks, unified error handling, and better encapsulation of the data access layer.

One of the most important requirements, consistently identified by ISG's clients as a high priority, is a better implementation of the data access layer. This requirement derives from the weaknesses of the existing J2EE data access layer, EJB, which has often been identified as a performance bottleneck. Addressing these performance issues through ad hoc caching introduces data integrity risks, thus presenting vulnerability to the rest of the platform.

For trading applications, the integrity and performance limitations of the J2EE data access layer act as barriers to the adoption of Application Servers for trading applications. To provide the best solutions to its clients, ISG recommends that companies adopt an architecture, which adds a "best of breed" data access layer to their J2EE platform. One leading software vendor providing such a solution is Persistence Software.

Breaking the Data Access Bottleneck

Over the past decade, International Systems Group (ISG), Inc has performed many application development and integration projects in the financial services industry. This has given us an opportunity to witness both the evolution of Middleware technologies as well as the shifting requirements and IT policies of many organizations.

As we have observed, some commercial Middleware products have clearly matured and provide a more comprehensive framework than what previous technologies—e.g., CORBA—had to offer.

Persistence Software provides efficient data management for high-access, time sensitive applications such as trading and real-time logistics. For over 10 years, Persistence Software has been a source of innovative data management software and has received 7 core patents on its mapping and caching technology. The introduction of the EdgeXtend product prompted ISG to take a closer look at how this technology could provide a commercial solution for improving the J2EE data access layer.

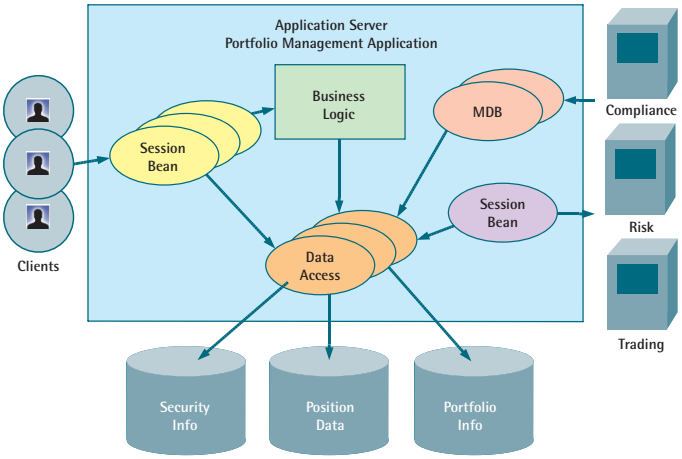
EdgeXtend is a Data Services product that remedies the inherent limitations of the J2EE data access layer. While remaining fully compliant with open standards and interoperable with the most popular Application Server implementations (including BEA WebLogic and IBM WebSphere), EdgeXtend provides a unique implementation of the data access layer. It achieves this by combining a custom distributed cache mechanism with standard Entity Enterprise Java Beans (EJB) interfaces.

International Systems Group (ISG), Inc.
32 Broadway, Ste 414, New York, NY 10004
tel: (212)489-0400 • fax: (212)489-1125
web: www.isg-inc.com • email: isg@isg-inc.com

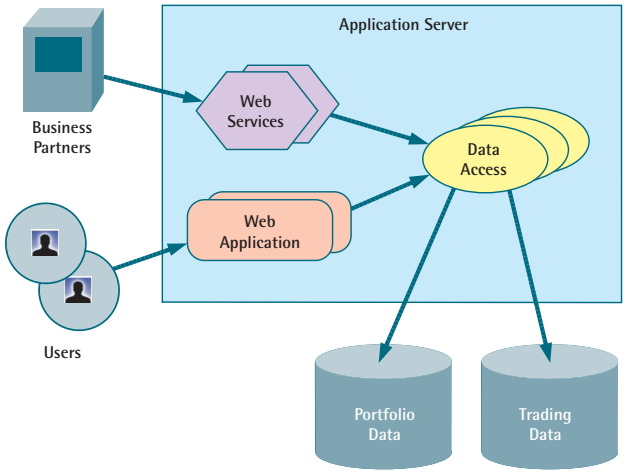
This white paper illustrates how the EdgeXtend product allows applications to benefit from cutting edge technology without getting locked into a proprietary vendor-specific solution. Without going into the lower-level details of the product (detailed product documentation is available from www.persistence.com/products/edgextend), this paper inspects EdgeXtend's approach to data management and demonstrates how it facilitates better performance and higher integrity for business applications in general, and financial applications in particular.

Examples of Application Server Deployments

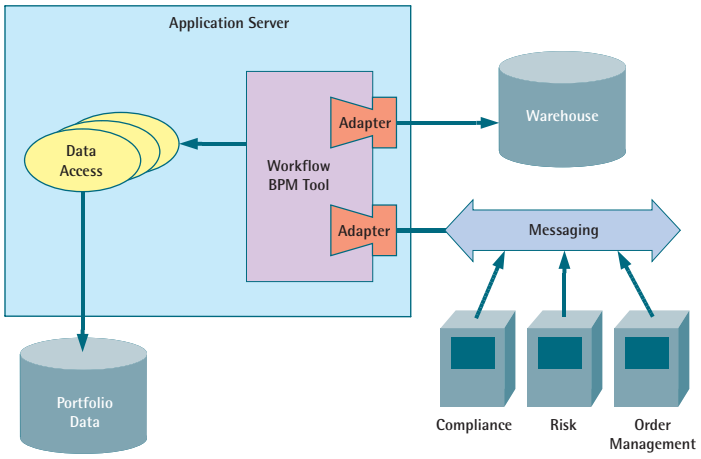
Utilization of an application server platform by a customer may vary from one instance for another. For example, the application server may be used to support a single business application, which in turn may consist of various front- and back-end components and integration points, such as, for example, a Portfolio Management Application that combines EJB-hosted business logic, servlets-based portfolio managers GUI, and integration with back-end data stores such as security warehouse through an adapter, or with external systems such as risk engine.



The Application Server may be used as a Web front-end to the existing Order Management System (OMS), supporting a business portal that facilitates a browser-based channel for retail end-user access, or integration via Web Services with business partners.



Another example of an Application Server-based architecture is an in-house monitoring system that aggregates comprehensive trade status from multiple sources (such as Order Management System, trading engine, position server etc.). The system then asserts validity of the business process through a workflow management tool, for example, by automatically dispatching compliance violation exceptions for a manual reconciliation.



Finally, an Application Server-based solution may comprise the elements of all of the above, serving as a hub in a hub-and-spoke enterprise architecture.

International Systems Group (ISG), Inc.
32 Broadway, Ste 414, New York, NY 10004
tel: (212)489-0400 • fax: (212)489-1125
web: www.isg-inc.com • email: isg@isg-inc.com

Typical requirements for High-Performance Trading Systems

Any Application Server-based solutions must provide for:

- High rates of business transactions, facilitating end-to-end throughput
- Response times demanded by the business processes
- Near real-time processing guarantees
- Ability to scale along with increased business loads
- Mission critical, highly available environment
- Guaranteed high levels of business data integrity
- Transactional consistency for distributed state

While the Application Server places high importance on these issues, architects of the Application Server-based applications still have to pay close attention to the throughput reliability "trade-off" for J2EE applications. This trade-off is driven by inherent weaknesses of the J2EE data access layer. Several factors contribute to J2EE's performance and integrity issues for real-time, high-volume operations:

- Every persistent data access operation is routed through a database call—a heavy weight operation that takes more time to complete than almost any other server-managed operation.
- The inability to optimize data access to take advantage of advanced database capabilities such as join queries or array operations.
- The database schema is frequently auto-generated by the bean compilers, and is not optimal (neither from a model nor SQL perspective).
- Inefficient thread pooling for data access operations creates congestion as application threads compete for disproportionately limited database resources.
- Finally, even the straightforward physical fetch of data from the disk has to go through the sub-optimal JDBC connection.

As a result of these issues, the data access layer in trading applications frequently becomes the performance bottleneck.

A Financial Services Case Example

To illustrate typical requirements for a data access layer to support real-time financial services applications, consider a portfolio management application that extends the following requirements:

- There are three kinds of business entities: portfolios, positions and security data, whose cardinality is in hundreds, thousands and tens of thousands, respectively.
- Any change to a model portfolio causes automatic changes to its child portfolios, each of which is subject to a set of business rules, for example portfolio holdings compliance rules.
- Benchmark rebalancing automatically generates actual buy/sell orders for all the associated portfolios.
- Position updates happen in real time based on trading activities.
- Security information is sporadically updated throughout the trading day (such as on an IPO issue).
- Finally, there are hundreds of portfolio managers who use the application to actively manage the portfolios and/or to model "what-if" scenarios.

International Systems Group (ISG), Inc.
32 Broadway, Ste 414, New York, NY 10004
tel: (212)489-0400 • fax: (212)489-1125
web: www.isg-inc.com • email: isg@isg-inc.com

This case example implies that the data access layer should accommodate hundreds, if not thousands of concurrent transactions, both of long- and short-running nature. In addition, it must accommodate complex entity relationships and the consequent business data consistency rules (e.g., it is not permissible to update price factors while playing out "what-if" scenarios).

Unless it is architected to support these requirements, the data access layer becomes strained and falls behind the real-time, highly concurrent application process. With increasing database congestion, individual business operations take more time to complete, decreasing the overall throughput and response time. This in turn compromises business processes that require real-time response and failure scenarios are given a wider window of opportunity, leading to higher failure risks. In the worst case, this congestion propagates to external, otherwise performant business systems.

The Standard Approach to Data Access Layer Architecture

Under the standard J2EE model, persistent business data (such as position or security information, portfolio data or order execution state) can be managed in one of two ways:

1. Direct JDBC access to the business data from servlets. This technique provides the architecture with all the benefits of direct control over the data model and the ability to optimize queries at the SQL level. However, this approach does not capitalize on container-provided transactional and failover guarantees. Nor does it benefit from decoupling of the data model from the business logic. In other words, this approach does not take advantage of the managed component model, and suffers from all the inefficiencies and dangers of custom architecting, coding and deployment.

2. Business data encapsulated EJB entity beans. Under the J2EE specification, the container manages the mapping between Java components and the underlying database schema. This approach suffers from inefficiencies, such as the inability to optimize queries from the database perspective, de-normalization of the schema, and redundant data proliferation.

A Solution for the Challenges of Real-Time Trading Systems

Persistence Software's EdgeXtend is a product designed specifically to address the inefficiencies of the data access layer in Application Server-based architectures. It achieves this through several means:

1. EdgeXtend seamlessly fits into a J2EE-compliant architecture, and assures high performance at the data access layer level without compromising robustness and integrity of the solution. It supplies several critical Data Services, such as specialized entity beans, in-memory cache and distributed cache synchronization. By following the J2EE standard, it does not impose proprietary, vendor-specific limitations on the application development or deployment.

2. EdgeXtend also provides a highly scalable solution that allows deployment configurations not commonly supported by the Application Server-centric architectures. An Application Server hosted on a distributed, dynamic cache rather than a database, creates a "virtual data server" that eliminates the need for a replicated database infrastructure, thus saving significant expense. As a result, the business attains reduced costs both through elimination of redundant hardware and software licenses.

In addition to runtime-level facilities, EdgeXtend also supplies a development-time toolset that further enhances the robustness and time/cost effectiveness of the application.

International Systems Group (ISG), Inc.
32 Broadway, Ste 414, New York, NY 10004
tel: (212)489-0400 • fax: (212)489-1125
web: www.isg-inc.com • email: isg@isg-inc.com

Using Database Caching to Address Performance Issues

Performance is typically one of the gravest concerns of application architecture, and is even more important in high volume, real time trading systems. However, a data access layer built on top of the native Entity EJB approach has inherent limitations: each data access operation results in one or several relatively expensive physical disk I/Os, and frequently, in a very expensive network call.

The traditional approach of the database vendors is to expedite data access by minimizing the number of physical I/O operations by inserting a cache between the calling application and the physical store. The cache holds a copy of the recently used data, which is served back to the application directly from memory; if the application updates the data, the data is written into the persistent store, and its cached version is invalidated.

However, relying on the vendor caching mechanism for increased performance has two notable caveats:

1. Significant semantics are lost in mapping business-level objects to the database-level relational schema. Consider, for example, an order instance that encapsulates such information as order id, CUSIP, side, order type etc. While an order exists as a singular, atomic object at the EJB level, it is not necessarily true at the database table level. Since a database vendor's caching mechanism operates at the record, rather than object level, discrepancies may exist in the cached state of the objects that map onto more than one record. Different portions of the business object may or may not be in memory at any given time, requiring assembly of the business object from cached and non-cached fragments, decreasing the effectiveness of the caching mechanism when compared to an object-level cache.

In contrast, the EdgeXtend caching mechanism operates at the level of object entities native to the business application. Object-to-relational mapping allows the EdgeXtend cache to hold business data aggregated exactly the way the business application expects them, and to serve them back very efficiently. In addition, the EdgeXtend caching mechanism is a supplement, rather than a substitution, to the database native cache, thus allowing the application to benefit from both.

2. Secondly, even when business data is fully cached by the database and served to the application without physically accessing a disk, the application still needs to issue a network (or at best, an inter-process communications, IPC) call to obtain it. Depending on the nature of the network, this call may take a comparatively long amount of time (to say nothing about retry attempts caused by unreliable or overloaded networks), resulting in considerable performance degradation. In contrast, EdgeXtend co-locates its cache in the same address space as the application itself, allowing the data to be served almost instantaneously and eliminating network bottlenecks.

Another issue affecting the performance is highly concurrent access to the database (for example when end users, trading engines and analytical tools attempt to simultaneously access position information). In this case, record-level locking on the frequently changing data will decrease both the throughput and the response time of the data layer. EdgeXtend enhances throughput by serving the data out of memory, which is extremely efficient for read/write access synchronization.

Even in the area of writing updates to the database, EdgeXtend can provide substantial advantages. EdgeXtend enhances update concurrency by enabling lazy writes (in other words, batching multiple object updates into a single database update) to the data store, and taking advantage of database bulk array operations whenever possible.

International Systems Group (ISG), Inc.
32 Broadway, Ste 414, New York, NY 10004
tel: (212)489-0400 • fax: (212)489-1125
web: www.isg-inc.com • email: isg@isg-inc.com

Thus, compared to standard J2EE implementation architectures, EdgeXtend provides the following benefits:

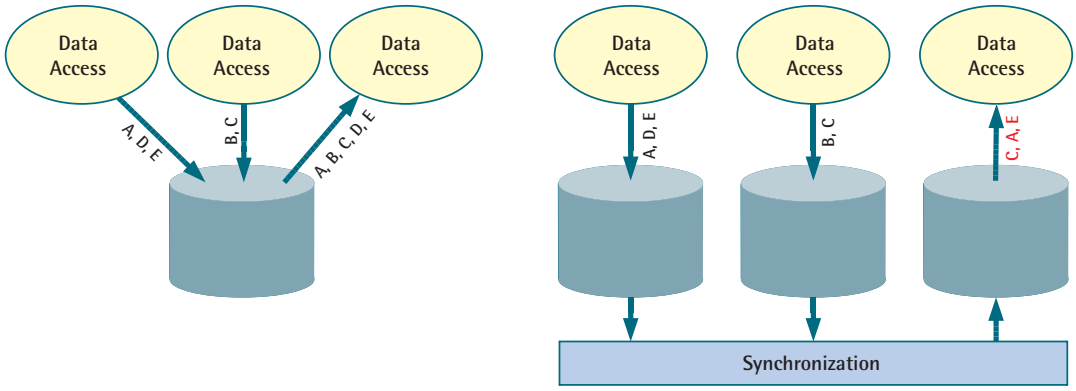
- Higher total throughput, in terms of business transactions performed (such as orders/day, position lookups/hour, executions/minute or market data updates/second).
- Better response time, in terms of time elapsed from initialization to completion of a business operation. Note, however, that the total response time may also be affected by lengthy computations (such as risk analysis), or unpredictable delays caused by external systems (such as compliance checks).
- Higher number of concurrent operations, in terms of business flows affecting the same data without loss of stability.
- Better real-time processing, as a measure of cause-effect delay (for example, processing quotes stream without sustained backlog).

Moving forward, performance and scalability requirements will continue to be driven by business needs. Consider, for example, the effects of increased trading volumes caused by web-enabling retail channels, and the increase in service requests resulting from application integration and workflow automation.

Using Standard J2EE Entity Beans to Improve Performance

Performance limitations imposed by the native J2EE data access model places significant limits on transaction volumes. These limits in turn increase deployment complexity by creating a need to deploy more servers to handle a given load.

While it is feasible to use database replication to deploy J2EE applications in a distributed environment, it is not realistic to expect such an application to scale well. Scaling by distributing the business data across multiple database instances has an immediate and unavoidable impact on the integrity of the distributed state.



The picture above demonstrates how background physical storage-level synchronization frequently employed by the database vendors does not account for business application-level processing context.

Two event producers perform data update operations A, B,C,D and E, in this order, which result in change to the stored business data. An event consumer relies on the data obtained through a read operation. In the case of a single data store (left), data update events seen by the event consumer through the transactional semantics of the database are guaranteed to be in order and without gaps. In the case of background replication, the event consumer may observe events out of order. This may lead to a severe integrity compromise (consider, for example, a trade order for a yet-unknown security in case of automatic trading of IPO

International Systems Group (ISG), Inc.
32 Broadway, Ste 414, New York, NY 10004
tel: (212)489-0400 • fax: (212)489-1125
web: www.isg-inc.com • email: isg@isg-inc.com

Some form of run-time synchronization between disjoint data replicas is necessary—for example, to ensure that order fill status is reflected consistently between two entity beans bound to different database instances. Additionally, keeping distributed databases synchronized requires complex engineering (e.g., database replication latencies may provide inconsistent execution price information to a price improvement algorithm).

The major dilemma facing data access-intensive applications is that performance and integrity are locked in a reverse relationship. Limitations in the J2EE specification force architects to enhance scalability by compromising the consistency of the business data. These same limitations dictate that ensuring sufficient data-level integrity constrains the scalability of the solution. Either case is an obvious jeopardy to the business needs.

This is not a new problem: Application Server-enabled architectures (at least those with complexity, performance and integrity requirements similar to trading applications) have long been exposed to this scalability vs. integrity trade-off, and there are two broad approaches to circumvent the limitations:

1. Scaling using the real-time replication facilities provided by all major database vendors.

These utilities establish the link between the database instances and propagate the changed data between them in the background in a master-slave manner. However, how, when and what business data is synchronized across database replicas is totally out of application control. As a result, the application cannot assure consistency of the distributed state from a business logic perspective, and even though the physical data may be in sync across different data store replicas, the business level integrity will still be compromised (consider, for example, the consequences of a benchmark data change during a lengthy risk analysis computation).

2. Custom propagation of the business state at the business object level via some form of messaging.

This approach provides sufficient control over when, what and how the business data is distributed, and is sufficiently integrated with the business application to assure that state propagation is consistent with the application's transactional context. However, this approach is not issue-free: implementation of custom state replication involves custom design, which can easily become very involved. The engineering complexities of the custom-build replication schemes revolve mostly around the following issues:

- A considerable amount of design work is required to generalize state concepts (such as types of business objects, transport independence, or transparency of operations from the API point of view). This activity has significant budget and time-to-market implications.
- Distributed state upkeep involves complex engineering issues, such as causality guarantees, object versioning, gap detection, and synchronization. Flawed implementations of custom-designed replication mechanisms create a potential for severely compromised application stability and integrity.

EdgeXtend addresses the performance versus integrity tradeoff for J2EE applications in a fundamental way. EdgeXtend proposes a data management model that incorporates the best features of both approaches:

- Integrates data across multiple instances into a shared data cache;
- Synchronizes data across multiple data caches, relieving the application from explicitly asserting distributed state integrity;
- Propagates data changes synchronously with the business process;
- Replicates data exclusively by business-level update events;
- Provides transactional control of data access in the environment native to the business logic.

International Systems Group (ISG), Inc.
32 Broadway, Ste 414, New York, NY 10004
tel: (212)489-0400 • fax: (212)489-1125
web: www.isg-inc.com • email: isg@isg-inc.com

EdgeXtend achieves this by replacing the standard, Application Server vendor-issued Entity EJB with a custom construct, an EdgeXtend Entity Bean.

EdgeXtend Entity Beans implement the standard J2EE entity bean interfaces. Thus, from the application point of view, they are indistinguishable from standard-issue beans. The container can manage lifecycle events for EdgeXtend Entity Beans—such as security, transactional demarcation and instance pooling—using standard interfaces in a transparent manner.

The internal implementation of the EdgeXtend Entity Beans, however, employs semantics different from the standard EJB approach. Instead of querying or updating a directly connected database, they perform read/write operations on an in-memory cache. In turn, the cache is directly connected to the database and manages the persistent data.

Furthermore, the cache broadcasts data changes to its peers using a guaranteed synchronization mechanism. These communications take place under control of a state propagation protocol, which assures that the shared state is delivered in a consistent, reliable fashion. When the remote cache receives a state update broadcast, it modifies its own object data correspondingly, while still honoring the transactional context and locking semantics of its own local Entity Beans.

Consider, for example, the portfolio management application described above. Enabled by EdgeXtend, the transactional volumes resultant from portfolio rebalancing, trade executions, market data and fundamentals updates, and user inquires can now be scaled across multiple servers without compromise to either performance or integrity of the distributed data.

The EdgeXtend data access layer will utilize a local copy of the business data, dedicated to the server, assuring maximum throughput rates, and it will cooperate in synchronizing data changes with other caches, ensuring that the local copy of the data is in sync across all server instances.

In other words, EdgeXtend empowers J2EE-based architectures with a data virtualization model, greatly reducing the complexity of a scalable solution, while simultaneously substantially enhancing its robustness and integrity.

Distributed Cache and Its Benefits

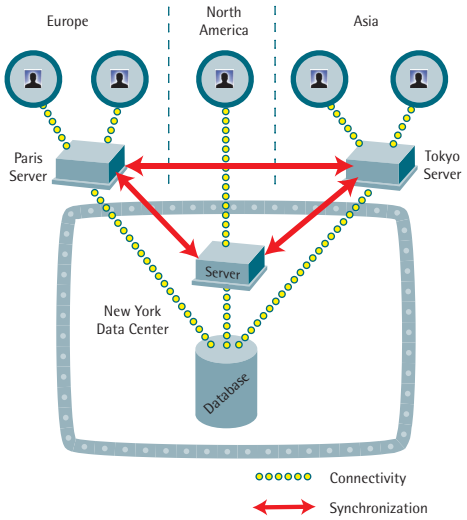
In many cases, applications need to be distributed between several geographically remote locations. For example, an order management application may be deployed at North American, European and Pacific locations, and provide all users with a Global Enterprise view, allowing them to place and manage orders on global markets.

However, Application Server-centric solutions are based on a hub-and-spoke architecture, concentrating all business rules and data within a resource-intensive data server. Often it is not feasible to base global operations on a single operations center. However, J2EE Application Server clusters do not lend themselves to geographical dispersion.

In such cases, the common deployment solution is to establish custom communications channels between application locations, and to route requests to the location that has the necessary resources (for example, a trade request originated in the Hong Kong branch, but placed on the New York Stock Exchange, would be processed by the New York branch). The cost implications of such a solution are obvious: increased development cost resulting from higher complexity, coupled with higher operations cost ensuing from the need to support server-to-server links, and potentially, elevated downtime risks due to diminished service availability.

Even if a custom propagation mechanism is available, it is likely to involve redundant data stores. Consider, for example, the need to support fixed income details locally at several geographical locations. This requires a server and a database at each of the locations—which may not be desirable from a cost perspective. Each database instance involves hardware and software license costs and operations support costs.

International Systems Group (ISG), Inc.
32 Broadway, Ste 414, New York, NY 10004
tel: (212)489-0400 • fax: (212)489-1125
web: www.isg-inc.com • email: isg@isg-inc.com



The EdgeXtend distributed cache mechanism helps to alleviate these concerns. Each geographically distributed, independent server instance (or an independent cluster) uses a local cache as the data layer. Only one (or a few) instances would actually back up the cache with a physical database, while others will totally rely on the in-cache data, or, at most, will use a common database for sporadic access induced by cache miss.

Since data is propagated between cache instances seamlessly and in real-time, applications running on the database-less Application Server instances will be supplied with a data access layer functioning effectively on top of a "virtual" database. In addition, the server instance may easily combine both a database-less copy of a distributed cache and a local database for its local, location-unique, non-shareable data.

Another, although less obvious cost benefit of the distributed cache is reduced losses from downtime. The locally cached version of the business data can ensure continued operations even when the application becomes disconnected from its remote data source due to a network failure.

In fact, a backup server passively caching business data can also provide a low cost, non-intrusive solution for failover and disaster recovery. Such a server is guaranteed to receive a consistent, integral view of the essential business data in real time with the actual business operations. Hence, operations switched over to the backup server will instantaneously and effortlessly be resumed from the point of failure.

EdgeXtend further mitigates the costs of the distributed environment by leveraging existing investments in the technical infrastructure. The data synchronization between different cache instances requires some form of a transport mechanism. EdgeXtend is compatible with existing corporate messaging infrastructures, such as Tibco Rendezvous, JMS or MQ series.

The Development Cost Benefits of EdgeXtend

The impact any given technology has on the development effort is frequently overlooked. Since the development effort translates into upfront costs directly, and into operational costs indirectly (such as induced by inadequate functionality or low stability), it is extremely important to understand the impact of any new technology.

International Systems Group (ISG), Inc.
32 Broadway, Ste 414, New York, NY 10004
tel: (212)489-0400 • fax: (212)489-1125
web: www.isg-inc.com • email: isg@isg-inc.com

EdgeXtend is an open, standards-based solution that extends the application server functionality. In this sense, it by no means is a proprietary solution. Because the functionality of EdgeXtend Entity Beans is expressed through the standard interfaces, it has the following key characteristics:

- A solution structured around EdgeXtend can be easily ported away from EdgeXtend to an Entity Bean EJB implementation native to the server (albeit less functional).
- Talent required is largely not different from the generic J2EE/EJB skill set. This applies both to development and support personnel.

The model-driven development capabilities of EdgeXtend also provide superior design and development efficiency. This includes:

- Automatic generation of data access code, greatly relieving developers from tedious, error-prone manual tasks, and increasing efficiency and quality of the produced code.
- Development cycle that protects custom processing logic, ensuring that designers and developers are not locked into the solution supplied by the tool.
- Model-driven data access layer design that enforces object model integrity and consistency rules throughout the analysis, design and development phases.
- Intuitive and robust object-to-relational mapping, an activity notoriously complex and problem-prone otherwise.

Finally, from the post-development perspective, EdgeXtend is not an invasive technology:

- From the deployment point of view, it is not substantially different than the deployment of a native J2EE application.
- From the configuration and runtime maintenance perspective, EdgeXtend does not require extensive specialization of knowledge—a few configuration parameters determine the run-time behavior of an EdgeXtend-enabled solution.
- From the perspective of the agile business, EdgeXtend enhances application flexibility through model-driven data access layer development. In addition, custom-coded constructs are preserved even when the underlying data model changes.

Summary & Conclusions

EdgeXtend benefits the enterprise by reducing the total cost of ownership in several important ways:

- EdgeXtend enables a more performant solution that, while benefiting from all the essential guarantees of an Application Server-based architecture, does not suffer from the inherent scalability limitations of a typical J2EE data access layer.
- EdgeXtend provides for higher integrity of business data. This translates both into reduced business risk and into increased quality of service.
- EdgeXtend enables more efficient deployment architectures that reduce both capital and operating costs to scale an application.
- EdgeXtend reduces design costs by providing an intuitive approach that eliminates most of the complexities of custom, home-grown architectures.
- EdgeXtend reduces development costs by providing a set of tools that increase development productivity and quality of code.
- EdgeXtend leverages investment into technology by using the existing corporate message bus.

At the same time, migration to EdgeXtend does not entail additional risk factors:

- It is based on well-established and widely accepted standards.
- It does not affect compatibility with other software packages.
- It does not require any additional investment into technology.
- It does not require any exotic skill sets.
- It does not build any vendor dependencies into the application: an application can be degraded to an EdgeXtend-unaware state in a relatively straightforward manner—but obviously with the significant implications on the quality and the operations costs.

Thus, EdgeXtend provides a very viable performance-enhancing, cost effective solution for J2EE Application Server-based architectures, and especially, for highly demanding financial services applications.

International Systems
Group (ISG), Inc.
32 Broadway, Ste 414
New York, NY 10004
tel: (212)489-0400
fax: (212)489-1125
web: www.isg-inc.com
email: isg@isg-inc.com