The data federation pattern virtualizes data from multiple disparate information sources. The pattern creates an integrated view into distributed information without creating data redundancy while federating both structured and unstructured information. This article describes the federation of structured information (data) with a focus on the SOA context. This pattern specification helps data and application architects make informed decisions on data architecture and document decision guidelines.

View more content in this series

Introduction

Many organizations struggle with the disparity and distribution of information. In many cases, users spend a large amount of time searching for and manually aggregating, correlating and correcting relevant information instead of acting on the insight that they gain from the information.

This widely recognized challenge also occurs when implementing a Service-Oriented Architecture (SOA). Often, core services require aggregated, quality information from multiple diverse sources.

Several concepts and technologies address those integration needs. Data federation is one of them. Data federation aims to efficiently join data from multiple heterogeneous sources, leaving the data in place -- without creating data redundancy. The data federation pattern supports data operations against an integrated and transient (virtual) view where the real data is stored in multiple diverse sources. The source data remains under the control of the source systems and is pulled on demand for federated access.

This article highlights the value of the data federation approach. After describing the context in which we apply data federation, we discuss the problem that this pattern addresses, as well as the solution. We characterize the applicability of this pattern based on non-functional requirements (see the Considerations section). Some known usages of this pattern illustrate our experience in
applying this pattern. We conclude by summarizing the focus areas, risk areas and constraints of this pattern.

**Value proposition of the data federation approach**

**Transparency of underlying heterogeneity**

With data federation, the consumer will see a single uniform interface. Location transparency means the consuming application of the pattern does not need to be aware of where the data is stored. Nor does it need to know what language or programming interface is supported by the source database, thanks to invocation transparency. For example, if SQL is used, it does not matter to the application what dialect of SQL the source supports. The application also does not need to know how the data is physically stored due to physical data independence, fragmentation and replication transparency -- or what networking protocols are used, known as network transparency.

**Time-to-market advantage**

An application that is a consumer of the data federation server can interface with a single virtual data source. Without using the federation pattern, the application must interact with multiple sources individually through different interfaces and different protocols. Studies have shown that using the data federation pattern helps to reduce development time significantly when multiple sources have to be integrated. See Related topics section for more information.

**Reduced development and maintenance costs**

Many consumers may potentially need the same -- or very similar -- integrated information. In one approach, each consumer has its own implementation for aggregating information from diverse sources. Alternatively, the integrated view is developed once, and it is leveraged multiple times and maintained in a single place, thus creating a single point of change. This approach reduces development and maintenance costs.

**Performance advantage**

An implementation of the data federation pattern with a specific focus on advanced data processing technology has, in many cases, proven to have superior performance characteristics compared with a home-grown approach to aggregate information (see the Related topics section for more information). By leveraging advanced query processing capabilities, the federation server can optimally distribute the workload among the federation server itself and the various sources. It will determine which part of the workload is most effectively executed by which server in order to optimize response time.

**Reusability advantage**

After applying the data federation pattern to a particular integration scenario, the result of this specific federated access can be provided as a service to multiple service consumers. For example, an integration scenario may require retrieving structured and unstructured insurance claim data from a wide range of sources. In this example, the data federation pattern can provide
the solution to integrated claims data which is then surfaced through a portal to a claims agent. The same federated access can then be leveraged as a service to other consumers such as automated processes for standard claims applications, or client facing web applications, for example.

**Improved governance**

Governance is a key underpinning to the SOA lifecycle. The governance process is enhanced by the use of patterns by reinforcing best practices with predictable outcomes. Reuse of proven flexible patterns in the development and creation of systems can both ensure consistency and quality and reduce maintenance costs by having a single source to update with changes.

**Context**

Mergers and acquisitions among companies and organizations often require data and application architects to integrate disparate data sources into a unified view of the data. Consumers of this integrated information are traditional applications that interact directly with databases and require access to an extended set of data sources. The decision on how best to provide this unified view are often set against the availability of tooling, experience, expertise and culture of the organization. Using traditional legacy architectures, the time, effort and cost associated with the integration may exceed the business benefit. A pattern-based information services approach, when implemented within a services based environment, can enhance the reusability characteristics of the system over time.

Information services are part of the core backbone of a SOA. These information services provide Create-Read-Update-Delete (CRUD) access to domain information. They also surface information processing capabilities such as the results of analytical and scoring algorithms, data cleansing rules, etc. For the purposes of this article, we will focus on information integration services that provide a unifying view of the data, which often involves the integration of a bewildering array of disparate backend sources, and services.

When applying the data federation pattern, we need to distinguish between two contexts: the traditional, non-SOA context, addressed by many previous applications, and the SOA context which is the focus of this article. It is important to keep in mind that SOA is an architectural approach which results in reusable services that in many cases extend the capabilities of existing non-SOA implementations.

**Traditional context**

In what we refer to as the traditional context, a reporting application in a bank might need to analyze credit card transactions. Considering the volume of this data -- there are many million of transactions per day -- it is not efficient to store all this information in the analysis warehouse. Much older data is very infrequently accessed, as is certain context information, such as a flight itinerary. Storing all credit card transaction data -- current and outdated, core and related -- in the warehouse negatively impacts the performance. A better solution is to separate the two types of data: frequently used, more recent credit card transactions are stored in a warehouse while older
information is stored on tapes, for example. However, the reporting application should not need to be aware of this data distribution which can be provided through the federated approach.

**Figure 1. Traditional data federation pattern**

In this traditional context, applications typically use standard relational interfaces and protocols to interact with the federation server, SQL and JDBC/ODBC for example. The federation server in turn connects through various adaptors, or wrappers, to a variety of data sources such as relational databases, XML documents, packaged applications and content management and collaboration systems. The federation server is a virtual database with all of the capabilities of a relational database. The requesting application or user can perform any query requests within the scope of their access permissions. Upon completion of the query a result set is returned containing all of the records that met the selection criteria. This is illustrated in Figure 1. The figure is intended to illustrate that the traditional implementation may be based upon a relational application programming interface (API) using SQL (JDBC/ODBC) or XQuery.

**SOA context**

In an SOA context, a service `getCustomerCreditCardData` might need to retrieve comprehensive information about a customer and his recent credit card transactions. This information might not reside in a single system. Customer information might be stored in a customer master data management system, or multiple repositories, and credit card transactions might be stored in another data source. Data federation joins the information from multiple sources so that it can be surfaced as a service to the consumer.

In this SOA context, the federation server can act as a service provider and/or a service consumer which leverages SOA conforming interfaces. Note that this does not preclude the server from also providing support for the traditional, relational interfaces. The breadth of support is an implementation decision which is beyond the scope of this discussion. When the data federation server exposes integrated information as a service provider, a service consumer can access the integrated information through a service interface such as WSDL and HTTP/SOAP or other
agreed-to bindings. The data federation server can consume -- in order to integrate -- services provided by multiple information sources.

The thought behind using the data federation pattern in the SOA context is to leverage and reuse integrated information, that is, information integration services in an extensible manner for a variety of consumers. The modeling and definition of services is a key aspect of SOA. It is a commonly acknowledged best practice to design services so that they provide reuse and/or cross-enterprise interoperability and/or business process enablement of information or functionality. Many if not most successful SOA projects focus first on the most important, most widely used business functions that are exposed as services. Due to the key role that those services play, they often span multiple backend systems. Gathering information from multiple heterogeneous sources is therefore an important requirement and capability that SOA relies on. The service is not a query as in the traditional data access context, rather, it is a request for a business entity (or entities) which may be fulfilled by the federation service through a series of queries and other services.

**Figure 2. Data federation pattern in an SOA context**

Enabling information integration services within SOA requires additional functionality that encapsulates a federated access within a service-oriented interface. This is accomplished through Information Service Enablement. The purpose of this component is to surface certain federated queries in a service-oriented interface. For example, a federated query might be written in SQL and might specify access to product information. Through the Information Service Enablement component, this federated query can then be surfaced as a service, for example, defined by SCA or WSDL. The service that implements access to product data can then be shared across and beyond the enterprise.

Solutions that apply the data federation pattern in the traditional context leverage the advantage of the declarative and flexible nature of SQL. With appropriate security credentials, consumers can access any data in the source through an almost unlimited number of different SQL queries. Consumers have great flexibility in what to access and the format in which the result is returned.
Although this flexibility is a great advantage in many situations, it also increases the complexity for consumers. Consumers have to understand the source data model and how to construct the result from this underlying source model. The larger the source data model, the more complex this task can become.

An SOA approach focuses first on defining and sharing a relatively limited number of the most critical business functions as services within and across the enterprise. Therefore, service-oriented interfaces are much more focused on the limited number of specific information requests that need to be surfaced. Developers benefit from this clear and narrow focus since they need less time to design the information request. They can simply select the appropriate service out of a relatively limited number of options.

**Problem statement**

In today's information-driven environment it is very common for architects and developers to implement a data federation solution. The challenges they face are usually affected by a number of architectural decisions, which may be driven by constraints that are technical, business or contractual in nature. This scenario includes several of these common constraints. First, data necessary to support the information access requirements of the project resides in multiple sources and must be integrated and provided as a single result to the consumer. Next, the target data sources cannot be replicated or copied in order to fulfill the access requirement. Lastly, the solution must integrate within an existing SOA while still supporting the traditional non-SOA applications as depicted in Figure 3.

**Figure 3. Heterogeneous interface access**

As described in the problem statement, it is the goal of this approach to avoid data redundancy when providing an integrated view over heterogeneous sources. The data federation server...
that is, the component that implements the data federation pattern -- must provide standard query interfaces for the non-traditional SOA context. This ensures that a wide range of traditional database applications can consume the federated data. The federation server must also provide query optimization capabilities in order to respond to the request most efficiently. The distribution and heterogeneity of data in this context requires a strong emphasis on how to best translate access to the integrated view and how to decompose and distribute the workload. When supporting write access to this integrated view, the federation server must synchronize the manipulation of data in the various sources into a logical unit of work. This ensures that the atomicity, consistency, isolation, and durability (ACID) criteria for transactions are met and that referential integrity is enforced.

In addition to these goals that address this traditional context, the approach must fit within a SOA. This will allow a wide range of consumers throughout and beyond the enterprise to effectively reuse the integrated view(s). Potential consumers of a federated access in a SOA are applications, portals and activities within a business process that need access to distributed information. For example, a manufacturer might define a service that retrieves real-time inventory information from heterogeneous sources. Internal applications as well as external business partners then access the same service, leveraging a consistent and most efficient implementation of this federated access.

**Solution description**

In both the traditional as well as the SOA context, the data federation server provides a solution to effectively join and process information from heterogeneous sources. This pattern realizes a synchronous, real-time integration approach to distributed data. The data federation server is responsible for receiving a query directed at an integrated view of diverse sources. It transforms it using complex optimizing algorithms that result in breaking the query down into a series of sub operations referred to as query partitioning and rewrite, applying the sub operations against the appropriate sources, gathering the results from each source, assembling the integrated results and finally returning the integrated results to the origin of the query. This processing sequence is done synchronously and in real time.

**Design time characteristics**

The data federation pattern requires the mapping of data elements from various data sources that are within the scope of the integrated view. For example, customer information, such as name and address from a policy holder, as in the example mentioned above, might be stored in a single table in one database and in multiple tables in another database. In order to build an integrated view, those different types of representations need to be mapped to the common view. The mapping can be performed manually by human actors or assisted by state-of-the-art tools based on various mapping algorithms which also capture any necessary transformation requirements. This allows the data federation server to receive queries against the integrated view and to calculate the optimum number and types of sub operations to perform.

When applying the data federation pattern in an SOA context, a set of federated queries need to be enabled and registered as services within SOA. For example, the integrated view to retrieve
critical structured and unstructured information about a policy holder, for example name, address, status, claim documents, repair estimates, and risk rating can be enabled as a service and shared among multiple consumers. The result of mapping in design time are typically federated views, similar to relational database views, which then can be deployed or created on the federation server.

**Run time**

The data federation server receives a request to the integrated view. According to the mapping definition, the federation server breaks down the federated query into multiple sub operations. Multiple factors influence this step:

- Where does the data reside that is necessary to respond to the federated query?
- What operations are necessary to transform the heterogeneous representations of the sources such as different data types, normalized model vs. non-normalized, for example, into the common integrated view?

The federation server uses the mapping information to address those questions. There are a number of other factors that influence the federated query processing which require information beyond the mapping specification such as:

- What operations are supported by the systems managing data sources and which have to be compensated by the federation server?
- What are the performance implications when executing a set of operations in the sources vs. the federation server? Which operations should the federation server delegate to the sources in order to better exploit their capabilities, to reduce data transfer, and to optimize the overall performance?

The answer to these questions requires knowledge of the source system and its query processing capabilities. In order to address the latter question, the federation server must also utilize a range of information about the operational environment as well as statistics of the source databases.

Once the federation server has determined the best execution strategy of all sub operations, it connects to the data sources -- both structured and unstructured information -- in order to retrieve relevant data, potentially using source-specific interfaces. According to the overall query execution plan, the sub operations are then applied at the sources. The result is received and aggregated into the result of the integrated view. The result is then returned to the consumer.

In the SOA context, the consumer submits a request via a predefined request format to the federation server. The federation server transforms the request into the corresponding SQL queries, or view definitions, to support the service. From there on, the same query decomposition, optimization and execution steps are performed as described above. The only difference in the SOA context is in the final step. The federation server translates the result of the traditional data federation approach into a service response and then returns it to the service consumer through the predefined service interface.
The functionality of the data federation pattern can be implemented using either database-related technologies such as optimizer or compensation, or by home-grown applications. Due to the complexity of query optimization over heterogeneous sources, it is an industry best practice to use a data federation implementation that leverages query optimization technology as provided by most database management systems.

Considerations

When applying the data federation pattern, it is important to understand its characteristics and how it is affected by the non-functional requirements described below. It is important to note that the non-functional requirements we have outlined do not take cache and data replication patterns into consideration. It is our belief that when adopting patterns that one starts with the basic patterns -- Data Federation in this example -- which can then be extended with additional patterns that address the additional non-functional requirements and functionality needed for the service. Cache and data replication patterns can be used to supplement the data federation or in the creation of a composite pattern. These patterns, and any other pattern that might be used in the overall implementation should be used cautiously as they may hinder the fulfillment of some non-functional requirements for which data federation has been chosen in the first place. For instance, they may increase data latency and create data redundancy. One needs to understand the trade-off points based on non-functional requirements and architectural decisions.

All characteristics of the non-functional requirements apply to both the traditional non-SOA context as well as to the SOA context. They include:
**Data security**

Only users and applications which have the appropriate credentials in the integrated sources are allowed to access the integrated view. This may be further restricted. One of the main reasons to apply this pattern is to leverage existing source systems with their data and capabilities. As a consequence, architects often intend to also leverage existing security mechanisms such as authentication and authorization of the source systems. Due to the heterogeneous and distributed nature of this environment, some challenges regarding single sign on and global access control might arise which are outside of the scope of the data federation pattern. In order to address those challenges, architects will need to combine the data federation pattern with other security-related patterns.

**Data latency**

The data federation pattern allows for real-time, integrated access to sources with the highest level of data currency.

**Source data volatility**

Due to the real-time access to source data upon receiving a request to the integrated view, data federation will always return the most current source information. Since the data federation pattern does not create copies of source data, source changes do not have to be propagated or processed in this approach.

**Data consistency and quality**

With the increase in frequency that complex data cleansing, standardization and transformation operations need to be performed, the probability of a negative impact on the overall response time increases. This is due to the real-time, synchronous nature of responding to requests in the data federation pattern. Any additional transformation will mean additional workload when responding to an integrated query. It is a best practice to minimize the complexity and number of field transformations required.

**Data availability**

The availability of integrated data depends on the availability of the data federation server and the integrated source servers at the time of the request. If one of the servers or any connection between the federation and the source server fails, the integrated view is not available.

**Impact of model changes on integrated model**

A very significant benefit to the data federation pattern is the ability to mask off many model changes which may be implemented in the source systems. The ability to accommodate the changes within the federation server can reduce the probability of exposing these changes to the initiator or consumer of the service. Further, changes can be made in the integrated view without requiring any changes to be propagated to the models for the data sources.
Frequency of transaction execution

A request to a federated server is executed synchronously. As soon as the response is received, the requester can invoke a subsequent request. The federated server should support concurrent requests initiated by multiple requesters. Highly frequent subsequent requests should have the same performance characteristics as a single request. An exception may occur if a source -- or a connector between the federation server and the source -- has specific characteristics that cause response performance degradation when frequently accessed. The ability of the federation server to execute transactions at a high rate is determined by the rate at which the federation server can access the source systems and the ability of those source systems to respond.

Transaction concurrency

In many cases, the data federation server has very similar characteristics than a database or content server. The ability to efficiently manage concurrent access is determined by the performance characteristic of the data federation server as well as the integrated source servers.

Performance and transaction response time

The transaction response time is determined by many factors, including:

- Complexity of the federated query: how many sub-operations such as filtering, joining, sorting, and so on, does the federation server need to execute to perform the query
- Query optimization and processing capabilities of the data federation server: how sophisticated is the design of federation server to take a federated query, break it down in sub-operations and optimize it, for example, applying certain sub-operation first such as the filter to reduce the data set and then performing other sub-operations such as sorting
- Data volume: the higher the data volume, the longer each sub-operation and therefore also the complete query will take
- network bandwidth: the throughput of a network connection between the federation server and a source impacts how quickly the federation server can access the source and therefore also the overall response time of the federated query
- CPU utilization: differences in resource utilization of the machines that the federation server and the data sources run on need to influence which sub-operations of the overall federated query are performed at the federation server vs. at the sources, if possible
- Query processing capabilities at the source servers: some data source servers have specific characteristics and limitations on how they process and optimize queries that impacts the overall performance
- The ability of the federated server to identify the optimal query strategy for each data source: if the federation server is aware of query processing capabilities of the source servers, it can determine what type of sub-operations to delegate and what sub-operations to perform at the federation server layer

The response time of a query against a virtual database, implemented by the data federation pattern -- fetching data from distributed sources -- might be slower than the same query against a single physical database with the same capabilities. The difference in response time will vary depending on the factors listed above. As a consequence, alternative patterns that provide the
integrated data set in a single physical database can allow for improved response times. Some implementations of the data federation pattern are capable of sending some or all of the sub-operations (sub-queries) in parallel to the integrated source systems. The parallel processing of sub-operations can significantly improve the response time.

**Create-read-update-delete (CRUD) profile**
Most data federation implementations support a various degree of read and write access. Some implementations coordinate a logical unit of work for write operations, known as a two-phase commit. In most cases, the data federation pattern is used for read access because of the complexity of write access. Without two phase commit support, the requester is responsible to ensure consistency among the sources when updating data. Because two phase commit generally requires a transaction manager, the degree of support for write access may vary depending upon the implementation of the transaction manager in addition to the functional capabilities of the source server with respect to applying and committing changes.

**Data volume per transaction**
The response time is influenced by the volume of data that need to be moved from remote source to federated server per transaction: the higher the data volume, the slower the response time. It is critical for the federation server to optimize the federated query so that the minimal amount of data has to be transferred between the federated server and the sources, especially when federated data volume is large. It is also important to understand the capacities and bandwidth supported by the network infrastructure and the impact that may have on the volume and frequency of data transferred.

**Solution delivery time**
As described in the value statement, data federation can greatly improve delivery time when integrating various sources.

**Skill set and experience**
The data federation pattern focuses on the integration of data sources and provides a single system image through a data-oriented interface. When surfacing integrated information as services, developers will also need to understand SOA concepts, standards and technologies.

**Reusability**
Logic on defining data access and aggregation can be reusable across different projects.

**Cost of maintaining multiple data sources**
Data federation does not reduce the cost of maintaining multiple data sources but greater benefits can be achieved due to integration and reuse of existing data sources.

**Cost of development**
It is relatively cheap if utilizing the best-of-breed federation engines, assuming a federated server infrastructure is in place.
Type of target models

This article has focused on federation for structured data. Today, the most common model is the relational model with the SQL standard. XML and XQuery are emerging standards with an increasing adoption in information management. Implementations of the data federation pattern typically support at least one of those models, sometimes both. Most implementations of the data federation pattern have a relatively strong focus on one -- or a very limited number of -- target models in order to process requests most efficiently.

Assured delivery and logical unit of work

In the IBM SOA Reference Architecture an enterprise service bus (ESB) is a key component of the infrastructure. One of the responsibilities of the ESB is to provide assured delivery. Due to the complexity of coordinating a logical unit of work, such as through two phase commit protocol in a federated environment, not all implementations of the data federation pattern support this functionality. When using federation servers that support this functionality, they need to be carefully analyzed on their database locking strategies to avoid negative impact to the performance of source systems.

Resource utilization

The federation server only utilizes resources when it processes a request that it receives from the consumer. The level of utilization on the federation server is also determined by the complexity of the request: the more complex the request, the more complex the task of finding the optimal plan how to decompose this federated request into sub operations. Another factor to the resource utilization is the percentage of sub operations that need to be executed in the federation server, for example to compensate lacking functionality in the source systems, vs. sub operations that can be pushed down to the source systems. Also, the amount of data that is received from the source systems and needs to go through the federation server impacts the resource utilization.

Transformation capabilities

The focus of the federation pattern is to leave data in place and to provide a real-time, virtual, integrated view. The solution approach in this pattern does not have any limitations on what transformations can be applied. Basic transformations are used in many implementations in order to convert heterogeneous source formats into the common view at the federation layer. However, complex transformations have a negative impact on the performance of the federation pattern and make this pattern less applicable for those scenarios. Therefore, most implementations of the data federation pattern focus less on complex transformation capabilities and more on query optimization technologies.

Type of source model, interfaces, protocols

Data federation addresses the problem of integrating data from heterogeneous source models and includes concepts to map those different source models into the common model at the federated layer. Implementations of the data federation pattern vary in their capabilities which specific source models they can integrate.
Scope and size of source models

The size of source models, number and type of attributes, may negatively impact the mapping task during runtime when mapping the underlying sources to the integrated view. The broader the scope, for example, the larger the number of attributes to be accessed, the longer it may take to identify corresponding elements.

Impact of federation server workload (transaction volume) to sources

The federation server forward for each request that it receives sub-operations to source systems. This impacts the resource utilization at the source systems negatively in that they need to respond to the sub-operations from the federation server. The more requests the federation server receives, the more sub-operations will be sent to the integrated sources.

Conclusion

We described the data federation pattern as an approach to data operations against an integrated and transient (virtual) view where the real data is stored in multiple diverse sources. We focused primarily on the SOA context within this article. We will conclude by summarizing when to apply and when not to apply the data federation pattern and to list important constraints.

Focus areas to apply the data federation pattern

- When time-to-market is one of top development priorities, data federation offers access to information sources quickly without lengthy information management infrastructure changes.
- Data federation supports requirements with respect to the replicating and duplication of data by enabling access to the data as it resides in the source. These requirements can be in response to regulations or rules restrict the movement or replication of data, e.g. subscription data or the commingling of personal information from different countries.
- Real-time access to distributed information as if from a single source. Information can be both structured and unstructured data.
- Flexible and extensible information integration approach for a dynamically changing environment, in particular schema evolution: due to the lack of data redundancy, changes in the federated schema reduce the impact of changes to integrated systems.
- The advantage of data federation is best exploited when a modest number of requests are received against limited result sets sized from multiple consistent, complementary data sources.

Risk areas to apply the data federation pattern

- Integration scenarios that require complex transformations to build the integrated view will have a negative impact on the response time particularly in this approach.
- Source servers may be negatively impacted by increased workload when they have to return data that is requested in a federated query. In order to process a request to the integrated view, the federation server will send sub-operations to integrated sources. The more complex those sub-operations are and the more frequently they are sent to the sources, the more additional workload the source servers need to manage.
• Scenarios that result in large intermediate result sets being moved from the target data sources to the federation server may have significant performance implications.
• Situations in which applications require a relatively high degree of availability of the integrated data may not be good candidates to apply this pattern. The availability of the integrated data is wholly dependent upon the availability of all federated and source servers involved in the process as well as availability, capacity and responsiveness of the network.

**Constraints when applying data federation pattern**

• Many implementations of the data federation pattern have limited capabilities to manipulate data. Many use SQL as the programming language and can only support SQL transformations.
• The performance largely depends on the sophistication of vendor specific implementation in terms of caching capability, understanding heterogeneous data sources and formulating optimal federation quires and execution path.
• Read-write access to different information sources - in particular when coordinating a logical unit of work - is constrained by the vendor-specific support.

**Product mapping**

The following IBM products implement this pattern:

• **InfoSphere Federation Server** allows applications to federate data among sources on distributed platforms (LINUX / UNIX / WINDOWS). Consumers can access through a SQL interface federated information from a wide range of data sources such as Netezza, Teradata, Informix, Oracle, SQL Server, DB2 for all platforms (LUW, i, z/OS), Microsoft Excel, XML files, etc. This product can also be combined with the following two products to aggregate structured, unstructured data and the valuable assets from the mainframe platforms.
• **InfoSphere Classic Federation Server for z/OS** provides a SQL interface through a wide range or data sources on the mainframe, such as DB2®, IMS, VSAM, IDMS, Adabas, etc. One of the key capabilities of this product is to map mainframe data structures to the relation model without coding so that developers with very limited mainframe knowledge can flexibly and efficiently access mainframe data.
• **InfoSphere Information Services Director** surfaces information management capabilities as services. It packages information integration logic, cleansing rules, information access etc. as services. This insulates the developer from the underlying provider of this functionality. Most relevant to this article is its capability to surface a federated access through a service oriented interface such as EJB, JMS, or Web services. This product provides the foundation infrastructure including load balancing and fault tolerance for Information Services. It realizes the Information Service Enablement component that is illustrated in Figure 2.
• **IBM Content Integrator** provides a uniform interface over federated content management systems. Typical content access operations are provided over a wide variety of content sources; among these is IBM DB2™ Content Manager.

**Acknowledgements**

We would like to thank Jonathan Adams, Kyle Brown, Lou Thomason, and Fan Lu for their support in writing this article and in developing this pattern.
Related topics

- **IBM Federated Database Technology** (developerWorks, March 2002) describes the technology that enables clients of IBM's federated database engine to access and integrate the data and specialized computational capabilities of a wide range of relational and non-relational data sources.

- **Patterns: Information Aggregation and Data Integration with DB2 Information Integrator** is an IBM Redbook that documents and prototypes the role of DB2 Information Integrator technologies and architecture in IBM's Patterns for e-business using a typical customer insight e-business scenario.

- **Performance of DB2 Information Integrator in a J2EE Environment with Multiple Data Sources** (developerWorks, June 2003) compares performance results for queries run with federation vs. queries using direct access.

- **Using DB2 Information Integrator for J2EE Development: A Cost / Benefit Analysis**, (developerWorks, May 2003) describes using DB2 Information Integrator to access disparate back end data sources, including Oracle, DB2 UDB, and Microsoft Excel.

© Copyright IBM Corporation 2006, 2012
Trademarks
(www.ibm.com/developerworks/ibm/trademarks/)