XAware™

Whitepaper >

XAware > Technology Overview

SERVICE ORIENTED INTEGRATION SOFTWARE

May 2007

Prepared by:
Kirstan Vandersluis
Chief Science Officer
XAware, Inc

Whitepaper >

Table of Contents

Whitepaper >

The Need For Integration >

Over the decades, companies have accumulated information processing systems to manage different parts of the business as efficiently as possible. Unfortunately, this evolution of information technology infrastructure has most often occurred without an overarching design, resulting in many stand-alone systems that perform a single function well, but fail to interact with each other. As companies accelerate electronic business initiatives, communication with customers, business partners, and between internal divisions demands well-integrated systems with fluid exchange of information.

Integration Evolution >

Traditional vendors of Enterprise Application Integration (EAI) software have evolved from message-based transaction processing into business process management platforms. While these platforms have done a good job managing transactions with their process-centric approach, they have not addressed the key issue facing most enterprises, that of complex data stored in a myriad of locations. In fact, analysts now claim that up to 70% of the application development effort is consumed by the task of accessing data. Clearly, there is a need to address this complexity with the same level of sophistication that EAI vendors have brought to bear on process-oriented solutions. It is this effort for which the XAware integration environment has been created. This environment has been solving complex, data-centric integration problems since 1999.

Integration Using XAware Technology >

The XAware integration software tools provide an elegant solution to many of the most complex data problems in the enterprise. By defining a data abstraction layer, and exposing information objects as rich business services, XAware provides what industry analyst ZapThink calls a Service Oriented Integration (SOI) toolset. SOI leads to reduced complexity in accessing information, reusability, cost savings, and rapid return on investment. XAware reduces or eliminates many of the issues caused by complex data environments, including:

- Up to 70% of application development resources are consumed by integration efforts to access the right information for an application.
- Business information is stored in many systems, applications, and databases.
- Data related to one business object, such as a customer, is contained in many locations.
- The dispersed nature of the data forces applications to connect to many different systems to build a single view of business information
- Efficiency suffers from lack of access to information critical to running the business.
- Changes in required data sets are difficult to manage due to "black box" transformations buried in code or old applications, hiding the flow of data in the enterprise.

The following sections provide an overview of the XAware tools.

Whitepaper >

Technology Overview >

As a Service Oriented Integration solution, XAware provides real-time access to multiple information sources within the enterprise. XAware makes information objects available through data "views". A view is a representation of an information object or other collection of data within the enterprise, such as a customer, policy, or invoice. XAware includes software programs needed for an organization to create and manage these views, then expose them as services in a manner that is optimal for a particular application.

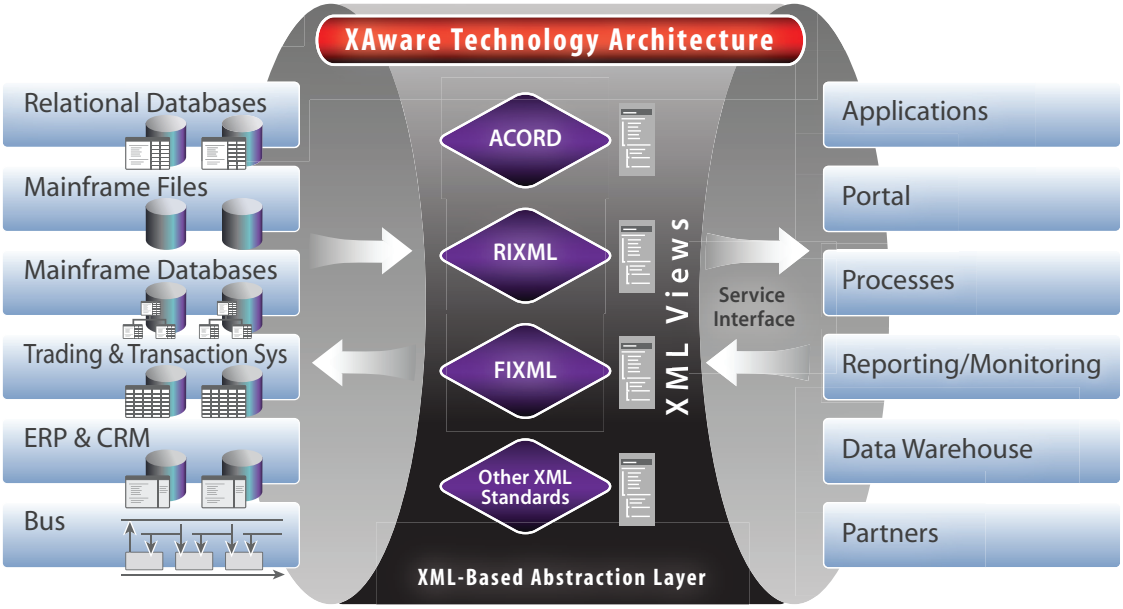

Figure 1. XAware Technology Architecture

Figure 1 shows the high level architecture of the XAware technology. As shown in the diagram, the XAware technology creates an XML-based abstraction layer to insulate the applications and users on the right from creating direct connections to enterprise systems on the left.

The interface into the abstraction layer is XML-based. Interaction with the data abstraction layer occurs using XML. Applications receive data in XML from the abstraction layer, and can write XML data to the abstraction layer, where it is processed into the appropriate back end data sources.

XAware Views

The abstraction layer provides access to data through views. These views represent information objects of varying complexity, from a simple, single database table, to a complete customer document containing profile, account information, and order history.

Since the interface into the abstraction layer is XML, the views are called "XML views". Each XML view defines an XML format, transformation rules, commands that interact with data sources, and processing instructions. Together, this information is called "metadata", and is itself represented in XML. XAware includes a design environment for creating XML views, and a server which makes the XML views available as services to applications needing real-time access to an organization's data sources. The format of the XAware metadata, and the software components included in the suite will be discussed later in this paper.

XAware Technology Benefits

There are several architectural benefits to using this XML-based abstraction layer. These benefits are discussed in the following paragraphs.

Data Abstraction

Applications interface to the abstraction layer, removing the concern about where or how many data sources are required to manage a logical business object. The data abstraction layer makes many physical data sources act as a single logical data source, greatly reducing the complexity of developing applications.

Service-oriented Architecture

The abstraction layer facilitates a service oriented architecture, where the manipulation of business information is handled through loosely coupled services with coarse-grained interfaces. Information exchanged with the service is defined by XML standards like ACORD or RIXML. Thus, the services are specified in terms of well understood business information like a Policy or Research Report. Using the nomenclature of the business analyst is a key aspect of SOA, which enables the rapid orchestration of new processes out of the rich business services layer.

XML-Enable Information

Because the abstraction layer is XML-based, the XAware technology transforms enterprise data to XML. This means that the growing number of XML-enabled applications can access the enterprise data. In addition, a growing number of industries are defining XML formats for common information within the industry. The XML abstraction layer "puts an XML face" on the enterprise data, enabling easy sharing and exchange of information conforming to a specified XML format.

Loose Coupling

Applications are no longer directly coupled to data sources. The physical layout of the source systems can change without affecting the applications. The abstraction layer is simply reconfigured to point to a new data source, and the application continues working without knowledge of the underlying physical change.

Repurpose Business Information

Because of the inertia within IT departments, there is great hesitancy to replace systems or rearrange the organization of data. The abstraction layer allows a system currently responsible for managing a particular information type, like a customer profile, to continue acting as the database of record, while enabling that information to be used for other purposes.

Centralized Security for Data Access

The XAware technology provides features to provide authentication, authorization, and encryption of information moving in and out of the abstraction layer. Organizations can manage data access to conform to existing security policies so that access to information sources is provided only to authorized applications or users.

XAware Technology Features

XAware has a set of features that combine to provide a flexible environment to create integration applications. These features are discussed in the following paragraphs.

Data Aggregation

The average organization has many sources of information, whether from multiple databases, applications, or other source. The ability to pull information from multiple sources, and package this information into a single logical unit is called data aggregation. XAware aggregates data from disparate enterprise system into logical XML views. A special case of supported aggregation is a "heterogeneous join" sometimes referred to as "XML join". This feature is similar to a database join between tables, except that it supports joins across data sets in any type of data source. For example, a customer data set from a mainframe can be joined with a data set retrieved from an enterprise application like Siebel or PeopleSoft.

Figure 2 shows a data aggregation operation for a bank. This particular bank has separate systems to manage accounts, consumer loans, and mortgages. It is desirable to view a consolidated list of all products for a particular customer. The XML abstraction layer retrieves information from each system, and packages the information into a single logical XML document. Presumably, a front end application would make a request to the XML abstraction layer, specifying a particular customer, then render the information in a form that is easily viewable, for example on a web page.

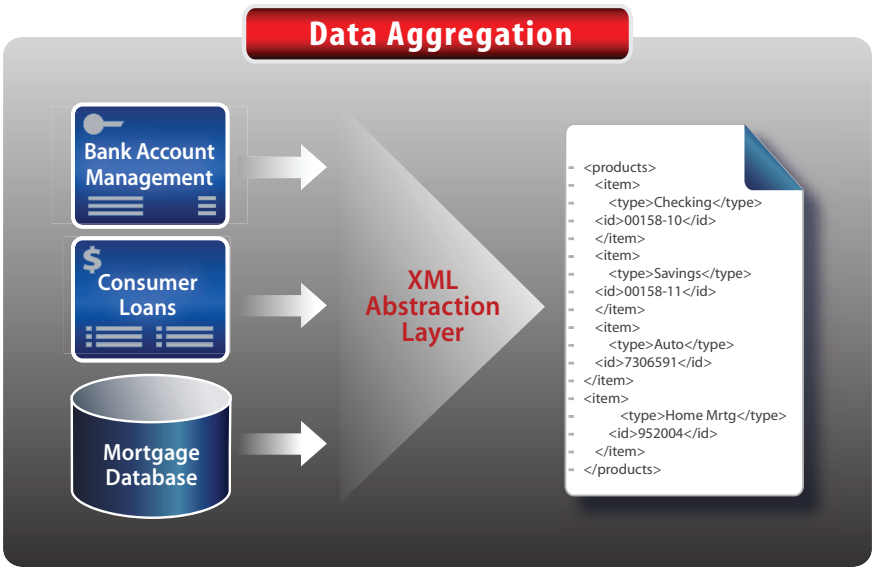

Figure 2. Data Aggregation

Data Chaining

Often, the various systems within an organization contain related data. Continuing with the previous bank example, a bank may require that a consumer establish a checking or savings account prior to obtaining a consumer loan. Thus, each loan in the Consumer Loans system has a link to a customer represented in the Account Management system. Data Chaining is a technique used to exploit such a link, to allow retrieving related information from multiple data sources. A data chaining operation is a sequence of requests into related system, where a second request depends on results from the first request. Said another way, a data chaining operation first retrieves a data set from one system, then uses information from that data set to send a request into a second system to retrieve related information. Data chaining is actual a specialized form of data aggregation.

Figure 3 shows a data chaining operation. The XML abstraction layer first queries the Account Management system for a customer ID. The input for the query differs based on the application, but as an example, the query could take a consumer's social security number or bank account number, or some other unique identifier. The result of the query is a set of data which includes the Customer ID. This Customer ID is then used to query into the Consumer Loans system to retrieve any related loans for this consumer. The results are combined and returned as XML to a requesting application.

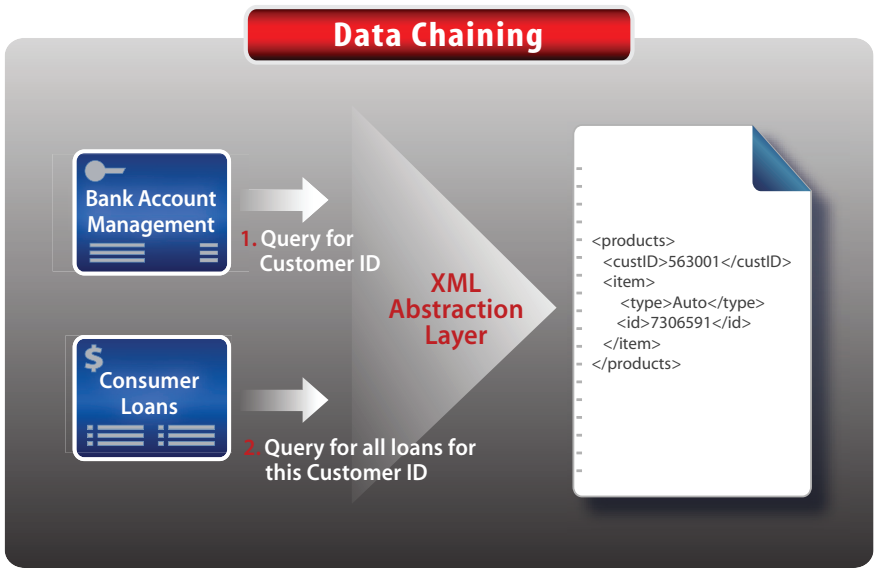

Figure 3. Data Chaining

The difference between simple data aggregation and data chaining is that with data chaining, there is a dependency on a first request before the second request can be sent. Usually, a parameter for the second request includes information that must be retrieved from the first request. In contrast, data aggregation creates requests that do not depend on one another. Often, these requests can be carried out in parallel to improve performance of building the view.

Inbound XML Processing

In addition to supporting views that return aggregated XML to an application, XAware's XML abstraction layer supports creating views that take XML as input, and send the data to appropriate processing resources. Very often, the inbound XML must be decomposed into multiple data sets to achieve the desired result.

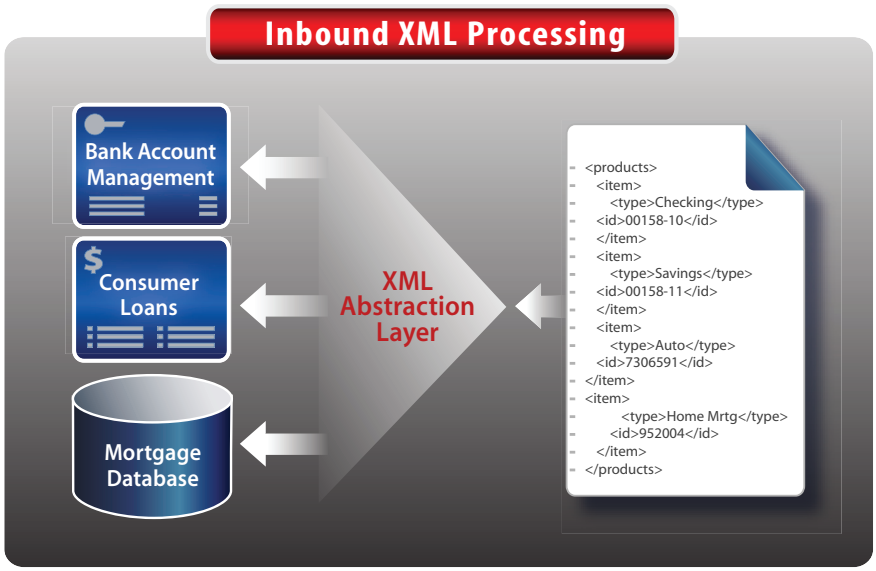

Figure 4. Inbound XML Processing

Let's assume in our banking example that our bank has acquired a smaller bank, and now needs to import business information into its systems. The acquired bank makes the information available in an XML format, which the parent bank needs to accept and decompose into its internal systems. Figure 4 shows the process. The inbound XML is received by the XML abstraction layer, decomposed into its constituent parts, and send to the proper system.

For inbound processing, the XML abstraction layer supports distributed transaction processing at the document level. This means that the processing of the entire inbound document into the multiple back end systems is performed within a transaction, so that either all the systems are updated, or none of the systems are updated. This prevents the highly undesirable situation where only a portion of the systems successfully process the new consumer information, resulting in an inconsistent set of data.

It is important to note that the distributed transaction processing requires that all systems participating in the inbound XML view must themselves support transactions, as the facilities of the individual systems are used to commit or roll back the overall transaction. If systems do not support transactions, significant programming effort is likely required to support what is known as "compensating transactions". This means that the state of the system needs to be managed manually, where the initial state before the transaction must be saved, then restored if a rollback occurs. At first glance, this may mean that the new information given to a system, such as a new savings account, simply needs to be deleted. While this may work for some cases, if the system has side effects, such as generating information downstream for other systems, then simply deleting the new savings account is insufficient. The dependent systems also need to be rolled back to their pre-transaction state. Compensating transactions is a complicated subject whose complete discussion is beyond the scope of this book.

Information Exchange

As its name implies, information exchange is the activity of moving information from one location to another location. This broadly stated activity can be applied both within an organization and outside an organization. Applications of information exchange include:

- Exchange Network. Many industries and government agencies have a need to exchange information with constituent organizations. Generally, common information formats are defined and agreed upon within the community, and all participants develop the ability to transform their data into the common format. XML is the most frequent technology chosen to define the common standard.
- Migration. Moving information from one set of systems to another. For example, a company changing its applications from older systems to more modern systems needs to move business information from the older systems to the newer systems. The application generally requires extraction and transformation from the older environment, and possibly additional transformation and loading into the target environment.
- Synchronization. Often, a company designates one or more "database of record" for business information. For example, the database of record for customer information may be a mainframe. For order information, the database of record may be a custom order management system built on an RDBMS. Often, it is desirable to maintain a copy of this information to make it readily available to other applications. While maintaining multiple copies of information generally complicates an enterprise's architecture, a replicated copy acts as a cache, reducing the need for real-time access to the primary source, and increasing retrieval performance for applications using the cache.

Figure 5 shows a typical information exchange architecture using the XML abstraction layer for both the sender and receiver of the information. While the diagram implies XAware technology is used for both, because of the non-proprietary nature of XML, any technology whatsoever can be used on either side. This is an important benefit of XML that has help speed its adoption in so many areas, especially when information exchange is required.

In figure 5, the information exchange relies on a common XML format agreed to by the participants. The sender transforms data in from its particular IT infrastructure and set of systems into the common format. The receiver transforms the common format and loads it into its systems.

The technique of using a common format for exchange is called a "common information model" (CIM) approach. CIM-based exchange can be successfully used for exchange networks, migration, and synchronization, and is ideal when the exchange participants have vastly different architectures. It abstracts the exchange process into two phases: extraction and transformation of source data to the CIM format, and transformation and loading from the CIM format into the target environment. This abstraction into two phases makes the overall process easier to implement, and also facilitates reuse. For example, a CIM-based migration which moves business information from a legacy system into a new application creates both the source to CIM transformation, and the CIM to target transformation. Each of these transformations can be used for other purposes. For example, a later migration from a different set of source systems requires only that the source to CIM transformation be developed. The previous CIM to target transformation and loading can be completely reused.

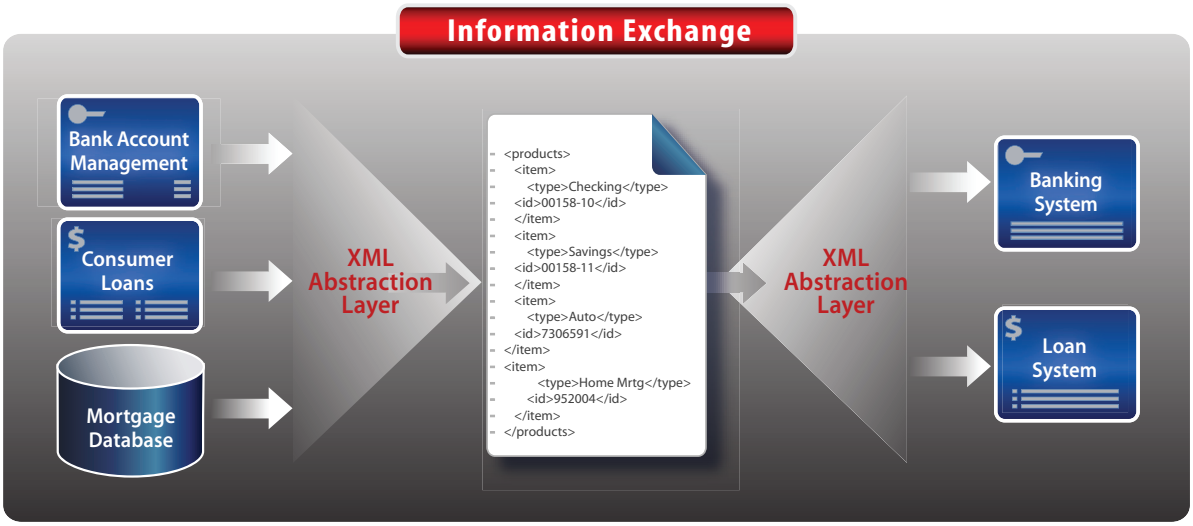

Figure 5. Information Exchange

Conditional Logic

XAware's XML abstraction layer allows the specification of processing instructions. The processing instructions are specified within an XML view, which controls the sequencing and flow of data within the view. The instruction set includes commands to read and write data sets, check results, alter the processing flow based on dynamic conditions, manipulate the XML to be returned to the application, control transaction and threading behavior, and handle error conditions within the view. In essence, the processing instructions constitute a scripting language defined by XAware that is executed in the XML abstraction layer.

The scripting language can be used to alter the course of processing for an XML view. This is useful both for inbound processing, and for generating XML views serving documents to applications. For inbound views, the conditional logic can direct the performance of multiple activities during processing. For example, if a company is processing inbound purchase orders, the view can first validate the customer, verify credit limits, send notifications to required systems, then finally decompose the XML and send each piece to the correct processing resource. For outbound views, those whose main purpose is to return aggregated XML to requesting applications, the conditional logic can be used to dynamically determine the proper data sources. For example, our initial banking example which provides an aggregated list of products for a customer queries three separate systems. Large companies may need to query dozens of systems to build such a view. Using conditional logic, an initial query can be made into a customer profile system to determine what products a customer has. Then, additional queries can be sent only to systems managing information pertinent to this customer. Using conditional logic, the processing of the view is more efficient because it only touches the systems needed, based on a run-time determination for the current customer.

Transformation

The XML abstraction layer provides the ability to transform data between almost any format. In general, transformation occurs from a system's native format into XML, from XML to a system's native format, or from one XML format to another.

Two categories of transformation are provided by the XML abstraction layer. The first is a series of tools that allow mapping a system's native data format to and from XML. These tools provide a graphical interface showing the source and target data sets, and allows visual mapping by dragging a source data element onto a target data element. The second category of transformation uses the standard XSL transformation tools, XML Stylesheet Language (XSL) Transformation (XSLT) and XSL Formatting (XSL-FO). XSL can be applied to either a subset of the view, or the entire view.

Whereas the two categories mentioned above are very good at performing structural transformation, and also support some field-level transformation, it is desirable to have greater flexibility in programming style and performance characteristics for some field-level transformations. For example, an insurance company may have a complex formula that creates a value for a field. To provide greater flexibility and performance for field-level transformations, the XML abstraction layer includes two additional transformation features specifically targeted at field-level manipulation. First, JavaScript (also known as ECMAScript) can be applied to any field to perform a field-level transformation. This provides a highly flexible field-level transformation feature, as the possible types of transformation are limited only by what is possible with the JavaScript programming language. Second, compiled Java functions can be custom-built and applied. These functions can take multiple parameters as input, and apply any type of processing to create output for the designated element.

Web Services

Web services enable organizations to create loosely coupled interfaces into system functionality. The XML abstraction layer provides a web service interface to XML views, exposing all the features of this layer to the growing number of applications that can work with web services.

Once an XML view is created, a web service interface can be generated by the software tools. This feature creates a WSDL file describing the interface, and enables a SOAP-based interface to initiate the view. Both inbound and outbound views are supported, so applications can both read and write enterprise application of any complexity using an industry standard interface.

Messaging

The XAware technology is provided with a standard interface to the Java Message Service. In addition to providing the interface to the messaging service, XAware resells a choice of J2EE-based application servers that include messaging. The result is a package that includes robust messaging capabilities, compliant with the JMS standard.

Two categories of messaging are included in JMS. Point-to-point messaging allows the asynchronous delivery of a message using a queue. Interactions can be loosely coupled, in that the sender does not need to know which application is actually receiving the message. Because of its asynchronous nature, the receiving application does not even need to be running at the time the sender places the message on the queue. The sender can simply place the message on the queue, then continue on with its work.

The second supported category is the publish/subscribe model. This model allows the sender to send a message to a queue, which can be read by many readers. In JMS parlance, the queue is called a "topic", and the act of sending a message to the queue is called "publishing". The applications that read from the queue must register this intention in a process called "subscribing". When a message is published, it is sent immediately to all applications that subscribed to it, and then deleted from the queue.

In addition to sending XML messages to a queue, XML views can be invoked using a message bean. In this way, an XML view can be asynchronously initiated based on the receipt of an XML message. For example, an order processing application can be configured to process inbound XML-based orders received on the queue. The corresponding XML view is executed, processing the order and updating back end systems as appropriate.

Security

The XML abstraction layer includes a role-based security scheme providing authentication and authorization capabilities that conform to J2EE standards. XAware's security model conforms to the JAAS (Java Authentication and Authorization Service) specification, to provide a consistent J2EE-based security framework. Using the JAAS model, security can be configured to integrate with LDAP and single-sign-on frameworks.

Within the security framework, each XML view can be enabled for a specific role or set of roles. Each role is managed via an access control list, which determines which users are part of the role. The roles are generally managed by the application server environment, and exposed to the XAware software via JAAS. In addition to controlling access to XML views, individual sets of data, of which the view is composed, can be controlled, so that data sets can only be aggregated and viewed by appropriate applications or individuals.

XAware Metadata Files >

As mentioned previously, XML views are defined by various characteristics such as an XML format, requests to data sources, transformation rules, and processing rules, which are stored as "metadata". XAware's metadata is stored as XML, in 3 types of files called metadata files. Figure 6 shows the relationship of the metadata files that make up XML views. Table 1 defines each type of metadata file.

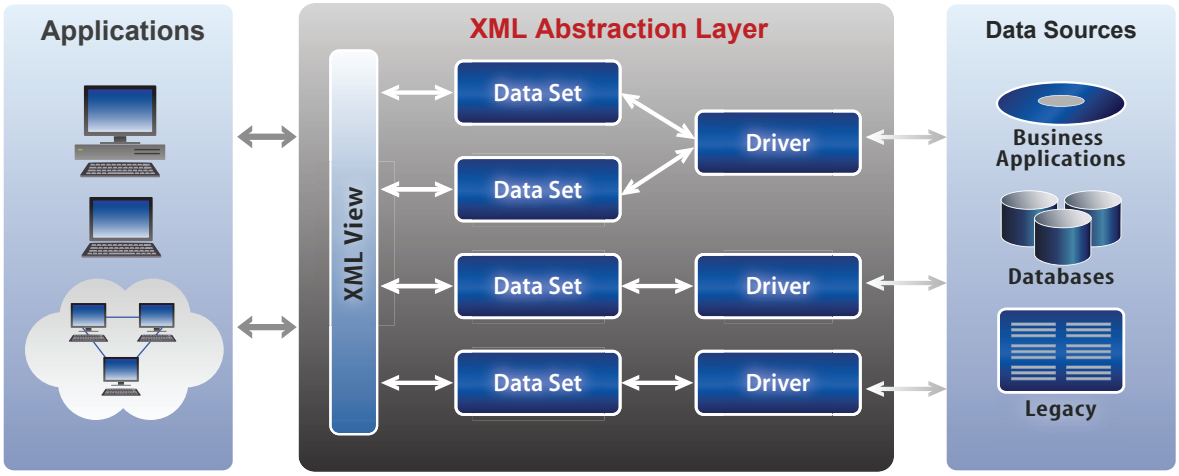

Figure 6. Metadata Files

Metadata File	Description
XML View	Primary metadata file representing the view. This file defines the XML format, and references data sets used to read or write to the data sources.
Data Set	Represents a single set of data from a single data source. Each data set file includes a request to the back end data source (for example, a select statement), plus formatting instructions on how to translate the raw data to or from XML.
Driver	A file containing connection information, specifying how to connect to the back end data source. For example, for relational sources, this file contains the connect string. Some data sets do not use a driver, and instead send requests directly to a data source. For example, HTTP-based data sets are retrieved directly by invoking an HTTP get, for example.

Table 1. Metadata Files

The previous discussion discusses the metadata files in generic terms. Within XAware, different terms are used for these items. Figure 7 shows the metadata file structure using XAware-specific terms. Table 2 defines the terms.

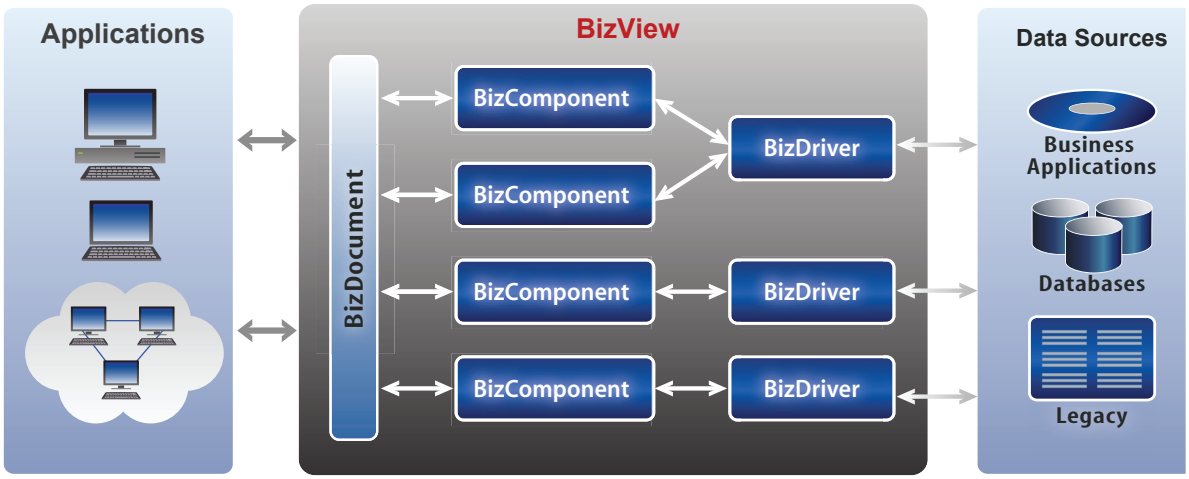

Figure 8. XAware Metadata Files

Metadata File	XAware Name for Metadata File
XML View	BizDocument
Data Set	BizComponent
Driver	BizDriver

Table 2. XAware Metadata Files

In addition to the XAware-specific names for each metadata file, you may notice in figure 7 that the combination of BizDocument, BizComponents, and BizDrivers for a specific XML view also has a name, "BizView". The BizView is referenced frequently in the XAware documentation and also the remainder of this book, and is logically the same concept of an XML view. It is the manifestation of an XML view by a set of metadata files.

The next sections discuss BizDocuments, BizComponents

BizDocument

An XAware BizDocument contains a hierarchical set of elements representing the XML view of business data. In addition to defining the XML format, a BizDocument may contain processing instructions, transformation instructions, and importantly, references to dynamically generated data sets called BizComponents. Each element in the BizDocument can be a static XML element, or a dynamic element. Static elements in the BizDocument are simply returned unmodified to the requesting client. In fact, the simplest BizDocument is one that is simply an XML file with no dynamic elements whatsoever. Dynamic elements contain processing instruction, such as conditionals, hierarchy manipulation commands, or references to BizComponents or other BizDocuments for execution. Because BizComponents present an XML-based interface, hiding the complexities of the data sources, all BizDocument processing instructions operate on XML structures. These commands are ambivalent to the original data sources, and operate equally well on static XML, or XML generated from a relational, non-relational, mainframe, or other source. For example, this layering enables an XML join between an RDBMS and a mainframe data source.

Elements in a BizDocument are processed in depth-first order. This means that the children of an element are processed before its siblings. In addition, many BizComponents create a data set of XML which actually contains additional dynamic elements. Once such a BizComponent is executed, its results are then processed, potentially accessing additional data sources and creating additional dynamic elements, all of which eventually get executed.

BizDocuments can also be defined with an interface accepting one or more parameters. Each parameter is a scalar value, such as a string or integer. In addition, a BizDocument may accept as input an XML document. Parameterization using either scalar or input XML allows customization of BizDocument processing for a particular purpose. For example, a BizDocument creating an XML view of a customer would likely be designed to accept an input parameter specifying a customer ID or perhaps a customer search string. The value of the parameter passed in at run time would drive the actual data retrieved from the back end data source. A common use of such a parameter is to reference it in an SQL 'where' clause to select just the required data. The parameter thus acts as a filter providing just the requested data to the application. We will see later that all the XAware metadata files have the capability to define an interface with parameters that drive processing within that component.

In addition to supporting parameters, BizDocuments allow the referencing of dynamic values from anywhere in the BizDocument. The references themselves follow a syntax similar to XPath, and allow manipulation of dynamically generated. A common example is generating a data set from one source, then referencing one of the generated elements in a parameter into another BizComponent. This is a typical process in data chaining, for example, where one data set contains a key into a second data set housed in a different system. The key is first retrieved using on BizComponent, then the dynamically generated element is referenced in the parameter to the second data set, allowing retrieval of related information.

BizComponent

The BizComponent is the dynamic data component of an XML view. A BizComponent can retrieve data, insert new data, update data in a back end data source, or send a data set to another processing resource. BizComponents can also provide processing capabilities not associated with data reads and writes, for example, to apply a processing algorithm to a data set. BizComponents are intended to be designed as reusable components representing the manipulation of a particular data set. For example, a BizComponent may be designed to retrieve a list of customers from an application's database. Once defined, this BizComponent may be called from many different BizDocuments.

BizComponents create an XML-based abstraction layer for BizDocuments. While BizComponents have widely varying interfaces to the many possible data sources in the enterprise, they all expose a common XML-based interface to the BizDocument. Thus, the BizComponent hides the complexity of communication to other data sources from the BizDocument. The BizDocument can manipulate data sources in a consistent way by using the common XML-based interface to BizComponents. Once data sets are converted to XML by the BizComponent, BizDocument commands can manipulate and process the XML without regard to the actual source. This enables common operations such as the XML join, hierarchy modification, and conditional operations.

When a BizDocument calls a BizComponent, that BizComponent is loaded and executed, generating XML results. The call to a BizComponent includes a reference to the appropriate metadata file, plus any parameters needed by the BizComponent. The BizComponent call acts just like a method or procedure call in a procedural programming language. The XML result set replaces the call to the BizComponent in the BizDocument. A BizComponent is called from a BizDocument or another BizComponent.

BizComponents can be nested so that one BizComponent calls one or more other BizComponents. For example, you can create nested BizComponents to query multiple databases, the first for customer information and the embedded BizComponent for order information, resulting in an aggregated view of the customer and order data.

BizDriver

A BizDriver enables a BizComponent to connect to a back-end data source. A BizDriver is generally required for each data source your BizComponent will access. Some BizComponent types, like HTTP and FTP, do not require a BizDriver for the BizComponent. These are generally stateless BizComponents that perform some action, such as copying a remote file using FTP. Multiple BizComponents can reuse any BizDriver as long as they are connecting to the same back-end data source.

As with BizDocuments and BizComponents, a BizDriver can have parameters. This feature is most often used to pass connection information to specify authentication and authorization information such as user name and password to the physical data source.

XAware Product Components >

XAware's XML abstraction layer includes several distinct layers of functionality and software components. Figure 9 shows these software components in relation to applications using the XML abstraction layer, and the data sources accessed. The software components are listed in table 3, and described in detail in the following paragraphs.

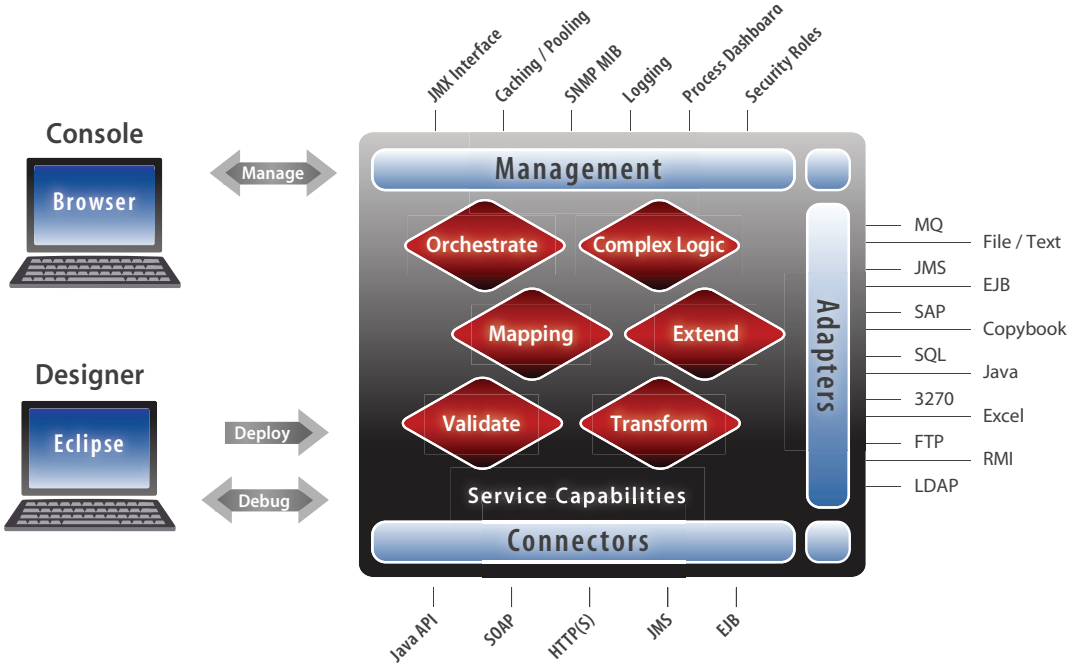

Figure 9. XAware software components

Software Component	Description
XAware Designer	Design environment used to create XML Views
Application Interfaces	Interface technologies used by applications to access the XML abstraction layer
XAware Engine	Run-time XML abstraction layer
Adapters	Software components providing access to data sources within an organization.

XAware Designer

XAware Designer is a design environment for creating and deploying XML views. XAware Designer guides a designer through he creation of the metadata files required by an XML view, which involves creating BizDocuments, BizComponents and BizDrivers. The tool is a visual, drag and drop design and development environment. Many of the features in XAware Designer are wizard based, which expedites development of XML-based applications.

Components developed utilizing XAware Designer are reusable. For example, once you create a BizComponent accessing a particular data set in Siebel, the same BizComponent can be used in any number of BizDocuments. XAware Designer also provides a cataloging feature, which presents a description of BizDocuments, BizComponents, and BizDrivers developed utilizing XAware Designer.

XAware Engine

XAware Engine is the core of XAware's tool set. It is an information integration server that serves XML views of disparate data, reducing the burden on applications to access the complex information within an enterprise. XAware Engine functions as a transformation server, transient workflow engine, business rule processing engine, and a virtual XML database. XAware Engine is multi-threaded, scalable, clusterable, robust, and offers high volume transaction processing capabilities.

XAware Engine is essentially a processing engine whose instructions are XAware metadata files. Client applications request a particular XML view. XAware Engine associates the request with a particular set of metadata files, and processes those files, returning the results to the requesting application.

A typical transaction inside the XAware Engine would proceed as follows:

1. A client application sends a request to the XAware Engine, identifying the XML view and appropriate input parameters and/or input XML.
2. The XAware Engine identifies the BizDocument corresponding to the XML view and creates a copy in memory.
3. The XAware Engine visits each element, one by one, in recursive fashion. For each, it performs the following activities:

> Perform variable substitution. This means that any reference to a parameter or other XPath reference to an element or attributes value, is replaced with the actual value.

- Process any commands. This could include checking a conditional, modifying the hierarchy, executing a BizComponent, and others. If the element references a BizComponent for execution, the referenced metadata is loaded to determine the BizComponent type. The corresponding BizComponent class is loaded and executed. The results of the BizComponent replace the current element.
- The next element for execution is determined. If a BizComponent was processed, the first element in the returned results is he next element to be executed. Otherwise, the first child of the current element is processed. If there are no children, then the next sibling becomes the next element.

4. Once the BizDocument is completely processed, its results are returned to the calling application.

Adapters

Adapters are used to provide connectivity between the XAware Engine and backend data systems. Using XAware Designer, users can quickly configure Adapters for use in web services or integration applications. Through a wizard process, users can configure Adapters to semantically map data between source and target structures and transform data by directly applying complex logical expressions. XAware includes a large set of adapters and provides you with the tools to build your own, when necessary. XAware can also build custom adapters to meet any data connectivity needs.

Application Interfaces

Application interfaces enable applications to communicate with XAware Engine via various protocols. XML views and their associated metadata files (BizDocuments, BizComponents, and BizDrivers) are designed and deployed to XAware Engine from XAware Designer. The application interfaces include both synchronous and asynchronous interfaces, providing flexibility in accessing a particular view to meet the needs of an application. Application interfaces include EJB, Servlet, Java API, SOAP, COM, ISAPI, HTTP, and JMS.

XML View Development Cycle >

Using XAware, a designer creates an XML view in a standard development cycle that includes XML view design, testing, and deployment to the XAware Engine. The following paragraphs describe these activities.

Design

XAware Designer is used to design XML views of data by creating the necessary XAware metadata files. The design process includes the following major steps:

1. Define the XML format for the XML view. An XML Schema or example instance can be imported for this purpose, or can be created. It is this XML format that represents the XML view. For an outbound view, the BizDocument will generate XML of this format. For an inbound XML view, the BizDocument will expect XML of this format as input, and will process the XML into one or more data sources.
2. Define the BizDocument Interface. Each view can be parameterized, so as a designer, you need to decide what information is passed into the BizDocument to help determine what information is returned or processed. For example, in creating a customer view, a customer identifier might be an appropriate parameter to determine which customer data to return to the calling application.
3. Convert XML structures to BizComponents. When the XML Schema or example instance is imported into XAware Designer, it displays the static hierarchy of the document structure. The next step, and the most important, is to convert the static structures to dynamically generated or processed XML. XAware Designer is optimized to perform this task, by letting the designer select a section of XML, then create a BizComponent to read or write that data to/from a back end data source. Converting to BizComponents is an iterative process, where one section of the hierarchy is treated at a time, potentially mapping each section to a different back end data source.
4. Provide control flow as necessary. Some XML views require control flow to process correctly. For example, the results of one BizComponent may need to be evaluated to determine the next BizComponent to process. While shown here as the last step in the process, injecting control logic into the flow can occur at any time during the design process.

Test and Debug

XAware Designer includes the features necessary to test and debug an XML view prior to deployment. At any time during the design process, a designer can execute the BizDocument to view and verify the accuracy of intermediate results. When a BizDocument is executed, detailed diagnostics are displayed in the log window and log file. Component calling structure in the Execution Profile shows each BizDocument and BizComponent that is called, and the execution time for each. A debugger is also provided, allowing you set breakpoints, single step through execution, and evaluate state at every step of the execution process. Items such as variable substitution, formatting of requests to back-end systems, and command results are available for viewing. In addition, when a data source provides native error diagnostics, that information is made available to the designer.

Deploy

Deployment of the XAware application involves packaging all related files into an archive file, called an XAware Archive (XAR) file. Security roles can be set during the packaging process to restrict execution privileges of BizDocuments and BizComponents. The archive file is then deployed to the server, where the XAware views become immediately available. In add, a web service interface can be created, in which case a WSDL file is generated for web service-enabled applications to access.

Manage >

Once deployed, XAware applications can be monitored for proper operation and performance in the production environment. A number of key management features are provided specifically to ease operational support concerns in the typical mission-critical, enterprise IT environment.

Statistics

The web-based management console and JMX interface provide detailed statistics about the execution of BizDocuments, BizComponents, and BizDrivers. For each component, execution counts are provided for successful and unsuccessful runs, plus minimum, maximum, and average execution time.

System Management Interface

The XAware engine includes an SNMP interface to enable common system management applications to monitor execution of BizDocuments, BizComponents, and BizDrivers. The interface lets operations personnel see at a glance the execution profile of components, what components are currently running, and success and error counts. Notifications can be configured to alert personnel of errors and of slow running components.

Remote Log Viewer

The Remote Log Viewer helps support personnel pinpoint problems by providing a query capability into the history of execution activity. Multiple levels of granularity are provided, showing component executions, BizDocument and BizComponent hierarchy flow, and detailed log events. A user can query by time range and search for errors, specific BizDocument executions, and keywords within the logs. For example, a support personnel can search for a specific end user or other parameter in a request or resoponse to narrow the problem search, then view specific information regarding the issue. Support personnel can individually set logging levels, to save highly detailed information for potential problem areas or components of interest.

Together, the management features enable enterprise-class capabilities to support production implementations. When problems do arise, support personnel can quickly locate and analyze available diagnostics to identify and correct problems.

XAware Environment Requirements >

XAware is a Java application with components that can be configured to run in multiple environments. XAware Designer is a Java application used to design and deploy XML views to the XAware Engine. As a Java application, XAware Designer can run on almost any computer with a modern graphical, mouse-based interface with a Java Virtual Machine, version 1.4 or above. XAware directly supports Microsoft Windows-based environments and Sun's Solaris-based workstations.

XAware can be configured to run in 3 different configurations, application servers, web servers, and stand-alone.

Application Server

The standard deployment option from XAware includes a J2EE application server from one of the leading application server vendors, including IBM WebSphere BEA Systems WebLogic Server, JBoss, and Sun. For organizations that already have an application server, the XAware Engine can be deployed to application servers from BEA Systems, IBM, Sun, and Oracle. A development version is also available which includes the popular JBOSS application server. Within the application server environment, the XAware Engine can be configured to run as a servlet, EJB, or message bean.

Web Server

XAware Engine can be configured to run in conjunction with a web server, without a full J2EE application server. In this configuration, XML views can be invoked using web service SOAP invocations, or using HTTP GET or POST commands. Other application interface technologies, like the EJB interface or message bean interface, are not available.

Stand-alone

XAware Engine features can be embedded in other applications by using the stand-alone configuration. Here, the software representing the server features are packaged as a Java archive (JAR) file, and can be directly accessed by another Java program through an application programming interface.

Extendible Architecture >

The XAware environment is extendible both in the server and in the design environment. Processing within XAware Engine is carried out by various processing components, and the set of components can be extended to add functionality to the environment. Each type of metadata file has a code component associated with it. When the metadata file gets invoked, the appropriate processing component is loaded and run. For example, when a BizDocument is invoked, the BizDocument processing class is loaded and run. This class knows how to interpret and execute the XML commands in a BizDocument metadata file. Likewise, when a BizComponent is called, an appropriate class is loaded that knows how to execute the BizComponent.

While there is only one BizDocument processor, there are many classes to process the various types of BizComponents and BizDrivers. There is exactly one class for each BizComponent type and each BizDriver type. For example, the BizComponent type that interacts with relational database systems is called the SQLBizComponent type. The corresponding class, SQLBizComponent.class, knows how to read and interpret the XML in a BizComponent with that type. Thus, the code module and XML format of the BizComponent are closely aligned, so that the code module understands the instructions in the corresponding metadata. Interfaces are standardized, so that the BizDocument processor relies on a standard interaction with all BizComponent classes. As you might guess, this means that BizComponent classes all implement the same base BizComponent interface. A similar relationship exists between BizComponents and BizDrivers, where BizDriver code modules implement a standard interface, which a BizComponent class relies on for standard interactions with BizDriver classes.

Extending XAware Engine, then, involves creating new BizComponent classes that conform to the standard BizComponent interface, then creating a metadata format that the new class understands how to process. The precise interactions between a BizComponent class and the BizDocument processor, though straightforward, are beyond the scope of this book. For more information, contact XAware at support@xaware.com.

The design environment, XAware Designer, can also be extended with new templates accessible from the environment. When a new BizComponent type is created, a new template can be installed in XAware Designer that helps in the construction of new BizComponents of that type. For more information, contact XAware at support@xaware.com.

Synergies With Other Integration Strategies >

We have stated that XAware is an SOI environment. However, in many IT environments, the complexity and diversity of integration problems requiring solutions means that no single integration strategy is a good fit for every problem. Often, companies take advantage of the focused solutions from different strategies such as EAI, SOI, and extraction, transformation, and loading (ETL)/Data Warehousing, so that the most efficient tools are used for particular integration problems. This section describes how XAware provides complimentary features to the other integration strategies, including Business Process Management, and ETL.

Business Process Management

Business Process Management (BPM) and EAI have the goal of automating business processes within the enterprise. BPM environment generally provide a flow graph-style programming environment, so that business analysts can graphically lay out a business process, and assign processing responsibilities to each activity in the flow. XAware provides a data abstraction layer that simplifies implementation of data-related activities within a process. As an example, figure 10 show a portion of a flow graph representing a business process. Two of the activities, Accept Order, and Verify Customer require interaction with enterprise systems. As shown, the data abstraction layer provided by XAware eases the complexity of accessing the data.

In addition to providing an information access layer for BPM/EAI, XAware provides a light-weight workflow control engine. The types of workflows that are implemented by the XAware Engine are "transient" workflows, those that run within a period of a transaction. This contrasts with general BPM/EAI platforms that can also handle long-term workflows, such as those required with multiple steps of human intervention. While XAware Engine can implement such a workflow with a series of coordinated transient workflows, BPM/EAI platforms focus on these types of long term processes, and so are better suited for that type of implementation.

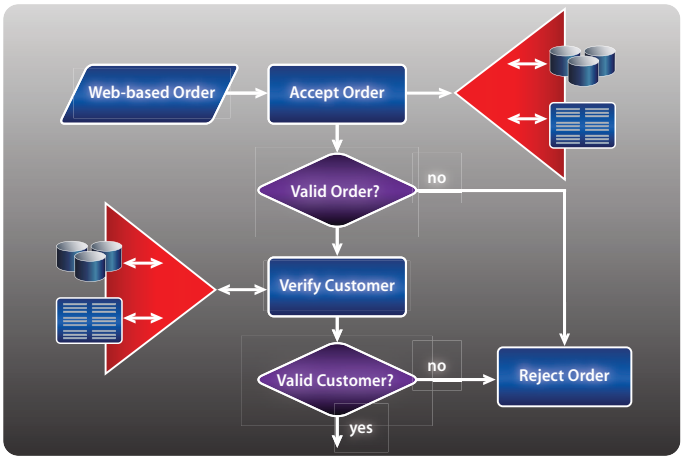

Figure 10. Abstraction layer reduces BPM complexity

ETL/Data Warehousing

Companies with data warehouses have the complex problem of defining business information that will be stored in the warehouse, then populating the common schema from disparate data sources across the enterprise. Moving data into the warehouse involves extracting, transforming, and loading the data (ETL). Additionally, these companies must make data available in a variety of formats for consumers of the information across the enterprise.

XAware assists organizations for both the ETL process, and for publishing data to consumers in customized formats. The ETL process generally requires that two views be built. The fist view is the input into a particular business object definition in the data warehouse. For example, a customer order may be represented in the warehouse, and an XML view is created reflected the data stored within it. The second view is created to extract and transform the data from one or more source data systems.
If more than one source system feeds the customer order object in the data warehouse, one view is established for each. When combined, the two views work together to extract and transform the data into a common XML format, then load the data into the data warehouse.

Consumers of information in the data warehouse also benefit from XAware's capabilities to expose views of information in the warehouse. XML views can be defined that extract data from the warehouse in a format appropriate for an application. Because of the aggregation features in these views, data from the warehouse can be combined with operational data where it makes sense for the applications.

Summary >

In this paper, we provide an overview of the XAware technology, a Service Oriented Integration product from XAware, Inc. It provides an XML-based data abstraction layer, exposing business information as services which greatly reducing the complexity of accessing information. As 70% of application development resources are spent accessing data, XAware's focus on reducing the complexity of data access leads to dramatic cost savings and shortened development times. XAware's SOI features provide a robust and flexible integration tool set leading to reusability of information objects, and rapid development of integration projects for data access and information sharing.