

Composite Data Virtualization

Composite Data Virtualization Platform Technical Overview

Composite Software

April 2010

TABLE OF CONTENTS

- INTRODUCTION 3**
- THE PROBLEM COMPOSITE DATA VIRTUALIZATION ADDRESSES 4**
- COMPOSITE DATA VIRTUALIZATION PLATFORM..... 5**
- COMPOSITE INFORMATION SERVER COMPONENTS..... 6**
 - DATA ACCESS COMPONENTS 6
 - QUERY PROCESSING COMPONENTS 8
 - METADATA MANAGEMENT COMPONENTS 10
 - CACHING COMPONENTS 11
 - SECURITY COMPONENTS..... 13
 - DATA DELIVERY COMPONENTS 15
 - TRANSACTION PROCESSING COMPONENTS 16
 - TRIGGERS 16
- COMPOSITE DEVELOPMENT..... 18**
 - COMPOSITE STUDIO..... 18
 - RELATIONAL VIEWS..... 18
 - COMPOSITE DESIGNER..... 19
 - DATA SERVICES..... 19
 - TEAM DEVELOPMENT 20
- COMPOSITE DATA VIRTUALIZATION PLATFORM OPTIONS 22**
 - COMPOSITE DISCOVERY..... 22
 - COMPOSITE ACTIVE CLUSTER OPTION 23
 - COMPOSITE APPLICATION DATA SERVICES..... 24
 - COMPOSITE MONITOR 24
- COMPOSITE SYSTEMS MANAGEMENT 26**
 - COMPOSITE DATA VIRTUALIZATION PLATFORM SYSTEMS MANAGEMENT TOOLS 26
- CONCLUSION..... 28**

INTRODUCTION

Information is power. And while more data than ever before is available from mainframes, relational databases, flat files, and other application sources, the task of making that data instantly accessible, and turning it into information, is daunting.

New approaches are needed. Following the path of storage, server, and application virtualization, enterprises and government agencies are applying data virtualization to meet their critical information needs.

Data virtualization is used to integrate data from multiple, disparate sources - anywhere across the extended enterprise - in a unified, logically virtualized manner for consumption by nearly any front-end business solution, including portals, reports, applications, search, and more.

As middleware technology, data virtualization has advanced beyond earlier high-performance query or enterprise information integration (EII) offerings. It complements traditional data integration techniques such as physical data consolidation, synchronization, and replication.

Composite Software is the only company that focuses solely on data virtualization. Scaling from project to enterprise, Composite's Data Virtualization Platform enables data federation, data warehouse extension, enterprise data sharing, real-time and cloud computing data integration.

This technical white paper provides a technical overview of the Composite's Data Virtualization Platform and shows how it is uniquely suited for today's enterprise scale data virtualization requirements.

THE PROBLEM COMPOSITE DATA VIRTUALIZATION ADDRESSES

To increase agility, reduce risk, and cut costs, Composite data virtualization addresses data integration's fundamental challenges:

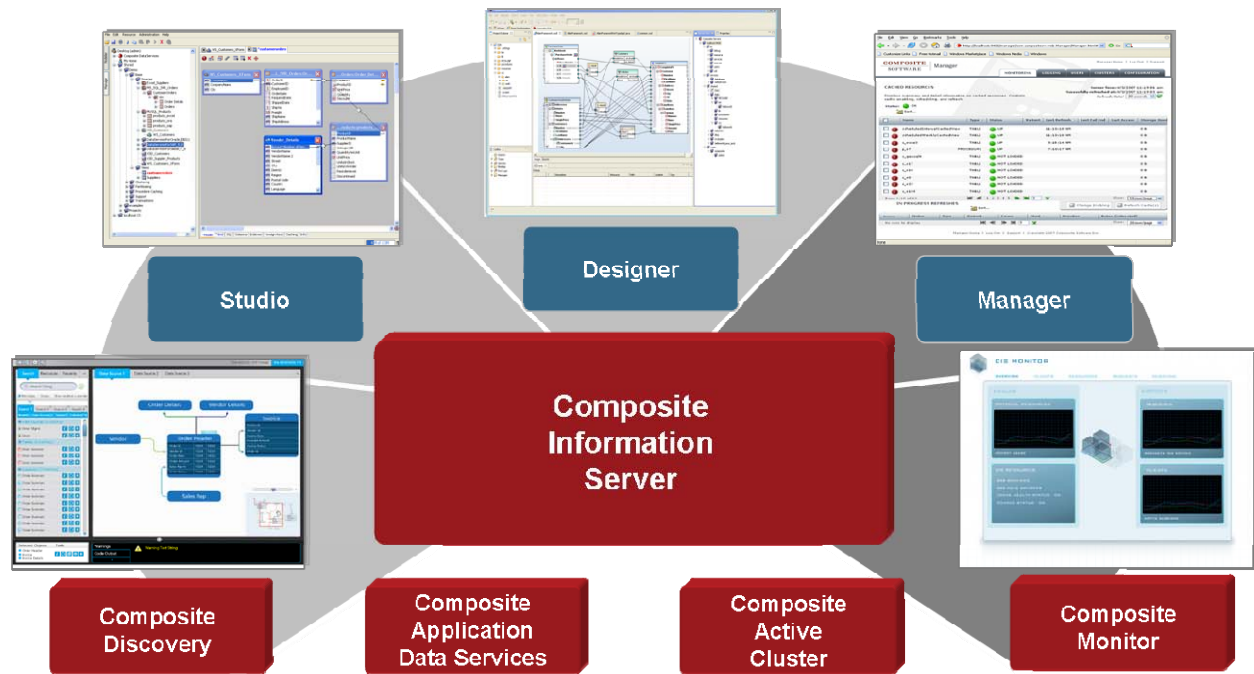
- **Data Complexity** – Data is difficult to identify and understand. Composite's data discovery simplifies the complexity challenge by automatically identifying relevant data and relationships and accelerating data model validation.
- **Data Structure** – Data isn't always in the required form. Composite's data abstraction overcomes data structure incompatibility by transforming data from its native structure and syntax into reusable views and data services that conform to standards and are easy for solutions developers to understand and solutions to consume.
- **Data Location** – Data resides in multiple locations and sources. Composite's data access overcomes the location challenge by providing data required by consuming solutions as if it were available from a single virtual location, rather than where it is actually stored.
- **Data Completeness** – Data frequently needs to be combined with other data to provide insight. Composite's data federation combines data to form more meaningful information. Data can be federated from both consolidated stores such as the enterprise data warehouse as well as original sources such as transaction systems.
- **Data Latency** – Up-to-the-minute data is often a key business requirement. Composite's data delivery provides the timely data required by consuming solutions whenever needed, without impacting source system performance.

Composite's data virtualization platform supports both a complete software development lifecycle and high reliability operation.

Typically deployed at two levels, the Composite data virtualization platform often complements other integration approaches such as consolidation or replication. At the project level, it can virtually integrate the data required in support of a specific application or use case. On an enterprise level, it can be implemented as common services or as a loosely-coupled data abstraction layer to share data across multiple solutions and use.

COMPOSITE DATA VIRTUALIZATION PLATFORM

The Composite Data Virtualization Platform consists of the Composite Information Server and a number of complementary options and products to support a complete software development lifecycle and high-reliability, scalable operation.



Composite Data Virtualization Platform

The Composite Information Server forms the core of the platform, enabling data access, federation, abstraction, and delivery using high performance query optimization, caching, security, and more. Within the Information Server, Designer is an Eclipse-based development environment, used primarily for data service development. Studio supports both relational and data service development. Administration of Composite users and management of installations is performed within the Studio and the Manager.

Composite Discovery finds enterprise data and relationships, and helps prepare the models needed by your downstream data virtualization development processes.

Composite Application Data Services provides access to leading applications such as SAP via standard APIs and pre-built objects such as customers, invoices, payments, etc.

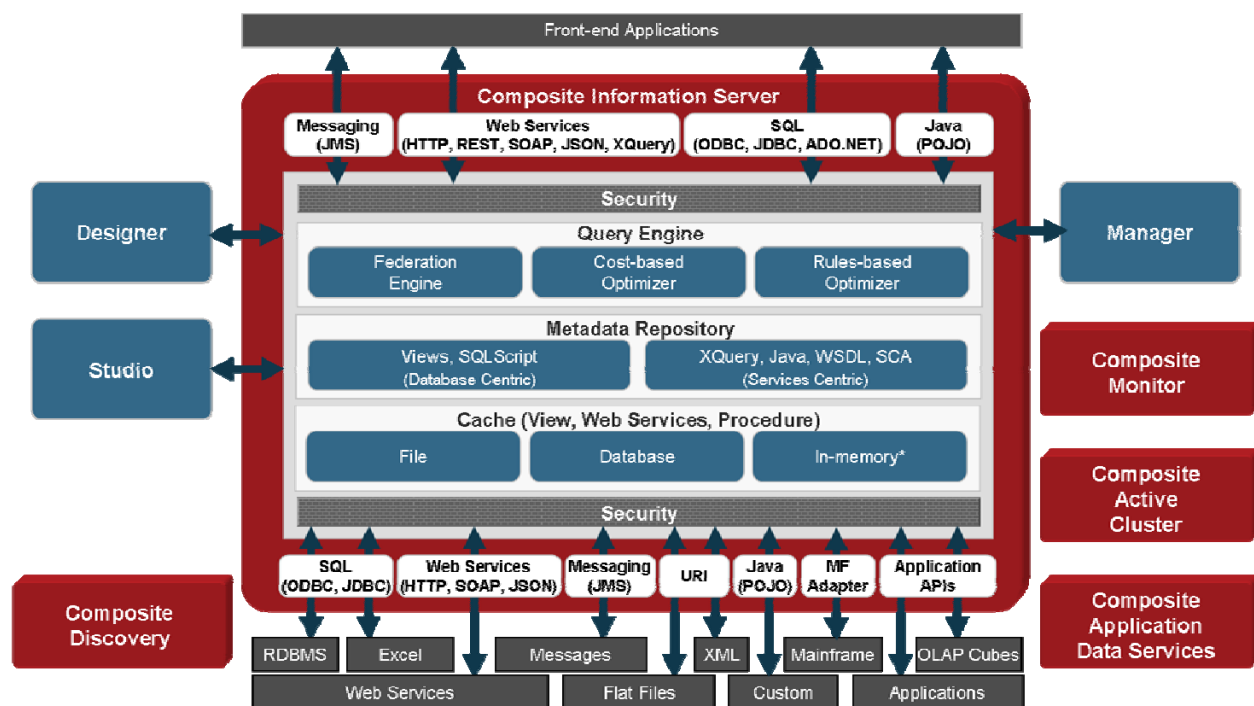
The Composite Active Cluster Option enables clustering of Composite Information Servers for added scalability and high availability.

The Composite Monitor monitors Composite activities and health within and across Composite Data Virtualization Platform environments.

COMPOSITE INFORMATION SERVER COMPONENTS

The Composite Information Server is a pure Java, multi-threaded application that forms the core of the Composite Data Virtualization Platform. The Composite Information Server is designed for “24x7x365,” highly available enterprise service and does not depend on a third-party application server. It connects to multiple, complex data sources, and streams arbitrarily large result sets from those sources to an arbitrarily large number of requesting clients. Moreover, the server software intelligently uses all available hardware resources and gracefully throttles back as resources become fully utilized.

The server comprises a number of major system components working in concert to provide secure, high performance data services. See the Composite Information Server Architecture below.



Composite Information Server Architecture

Data Access Components

Data source access is the lowest layer in the Composite Information Server stack, sitting closest to the data as it naturally exists in the enterprise. It provides protocol-specific and/or vendor-specific access to all of the disparate data sources required for your data virtualization implementation. At design-time, it examines (introspects) the data source to expose its structure and data types to the Composite Information Server. At run-time, it normalizes and streams data in forms that can be efficiently manipulated by the Composite Information Server.

Each data source type available in Composite Information Server uses an associated driver. When you provision a new data source for use in Composite Information Server, you set the driver’s parameters so that it can access a specific data source of that type (e.g., an Oracle database). Once connectivity is established, Composite Information Server introspects the data source structure and its data types, and the resulting metadata is stored in the Composite

Information Server's metadata repository. With the data source's metadata now available, you can then create views and data services using that data source.

One of the most important roles of the data source access layer is to normalize both the shape and types of the data. The Composite Information Server normalizes data sets into one of three shapes: tabular (i.e., rows and columns of values), hierarchical (e.g., like an XML document), or scalar (i.e., a single value). The Composite Information Server normalizes each datum within a data set into a standard SQL or XML data type (all of which are listed in the Composite Information Server reference manual). Data normalized into these standard shapes and types can then easily be manipulated, transformed, and combined with other data, without regard for the vendor-specific or protocol-specific details.

It's worth noting that even though the shape and type of the data get normalized by the data access layer, the values themselves remain unaltered as they pass through the data virtualization layer. This practice is important both for accuracy and efficiency, and the Composite Information Server has been carefully engineered to keep data values as close to their native form as possible.

The Composite Information Server can integrate data from a long list of data source types, all of which are listed in the Composite Information Server data sheet. Each of those data source types belongs to a shorter list of data source classes that are described below.

- **Relational Databases** – These are tabular data sources, and they include the familiar Oracle, DB2, and SQL Server databases. All relational databases are accessed using a vendor-specific JDBC driver, around which Composite wraps its own driver framework in the data access layer. Any database for which a standard JDBC driver exists can be accessed by Composite.
- **Web Services** – More and more data is becoming available through standardized web service interfaces, which may include SOAP, XML/HTTP, and REST. The Composite Information Server integrates the most prevalent web service standards to access the XML data from these sources.
- **Multi-dimensional Sources** – Composite accesses data from SAP BW and Oracle EssBase data sources by transforming SQL queries to MDX queries via Composite Application Data Services. In effect, this multi-dimensional data is transformed into slices of tabular data so the Composite Information Server can perform further federation, abstraction, and delivery functions.
- **Packaged Applications** – Most industry standard applications like SAP and Siebel provide their own vendor-specific APIs. Composite adds value to these APIs with collection of Application Data Services products that expose these APIs and key data entities as common objects.
- **Mainframes** – A significant amount of enterprise data still resides inside mainframes. Some mainframes can be accessed with modern database standards like JDBC, and Composite handles those as normal relational databases (e.g., DB2 on the mainframe). Other mainframes are unique legacy systems that require specific expertise and specialized software to access efficiently. Composite has a partnership with Data Direct to offer their native best-of-breed adapters to these important legacy data sources.

- **Files** – Composite can access data directly from certain kinds of structured files, including delimited tabular files, XML files, and Excel spreadsheets.
- **Custom** – Because the universe of enterprise data sources is extremely large and varied, and some data sources may not offer any standards-based API for accessing data, the Composite Information Server allows you to create custom data source drivers using the Java programming language. If there is any way to read data from a given data source, it is almost always possible to create a custom driver to allow the data to be used by the Composite Information Server.

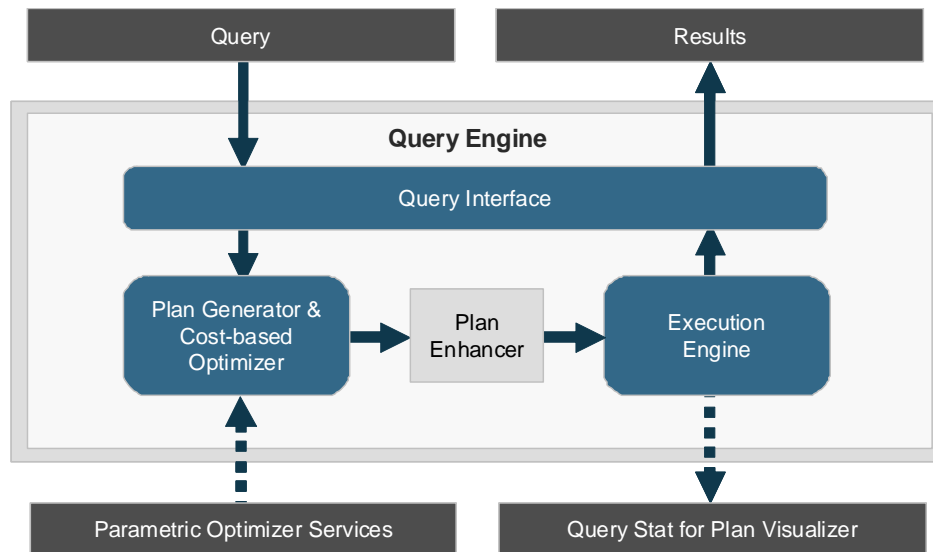
Query Processing Components

Key to the architecture is a very-high-performance query processing engine that uses distributed query-plan optimization and data-streaming technologies to return complex, federated results very quickly. The engine uses sophisticated rule- and cost-based query-optimization strategies that examine the plan of each query, and then creates a plan that optimizes processing and performance. The engine is designed to minimize any overhead and aims to leverage efficient join methods that outperform hand-coded queries written by a typical developer. In doing so, the engine employs a number of techniques, including:

- **SQL Pushdown** – The Query Processing Engine tries to offload as much query processing as possible by pushing down select query operations such as string searches, comparisons, local joins, sorting, aggregating, grouping into the underlying relational data sources. This allows the engine to take advantage of native data source capabilities and limit the amount of intermediate data returned from each data source.
- **Parallel Processing** – The engine tries to optimize query execution by employing parallel and asynchronous request processing. After building an optimized query plan, the engine executes data service calls asynchronously on separate threads, reducing idle time and data source response latency.
- **Distributed Joins** – The Query Processing Engine detects when a query being executed involves data consumed from different data sources and tries to employ distributed query optimization techniques to improve overall performance and minimize the amount of data moved over the network. The engine optimizes queries and takes advantage of sort-merge, semi, hash and nested-loop joins depending on the nature of the query and types of data sources.
- **Caching** – The Composite Information Server can be configured to cache results for query, procedure and web service calls on a per view/per query basis. When enabled, the caching engine stores result sets either in a local file-based cache or in a relational data base. The Query Engine will verify whether results of a query it is about to execute are already stored in the caching system and will use the cached data as appropriate. While caching can improve the overall execution time, its impact is most pronounced when used on frequently invoked queries that are executed against rarely changing data or high latency sources such as Web Service based data providers.
- **Advanced Query Optimization** – Additional techniques and algorithms include data source grouping, join algorithm selection, join ordering, union-join inversion, predicate pooling and propagation, and projection pruning.

The overall goal of these optimizations is to reduce the amount of data that must be moved across the network to satisfy the request. Network bandwidth is generally the scarcest resource in the processing pipeline, so reducing the amount of data that needs to be transferred has a significant impact on the latency and overall performance of a data virtualization implementation. Query optimization is aware of the capabilities of underlying data sources, and during optimization it can therefore determine how much work can be passed to the underlying data source.

Query processing occurs in several phases as seen below.



Query Processing Engine

- Plan Generation and Optimization Phase** –The plan generator and cost-based optimizer are responsible for producing one or more plans for each query. Based on rules and dynamic cost functions, the optimal plan is automatically selected. The cost function measures the time and computing effort required to execute each element of the query plan. Cost information comes from a service that pulls data from the repository about each data source in the query as well as other environmental information during plan-generation. For each data source, capabilities that are factored into plans include indexes, join support, and cardinality estimates.

The plan generator also compensates for the fact that each data source might have different capabilities that need to be normalized across the distributed and disparate data sources to successfully resolve the query. It's important to note that the cost-based optimizer can work with data sources with limited metadata (e.g., a Web service) as well as systems that have a significant volume of metadata (e.g., Oracle). Also, during plan generation, the server employs non-blocking join algorithms that use both memory-based and memory-disk algorithms to manage large result sets, and that have underlying data source awareness for pushing joins and predicates to underlying sources for execution.

- **Query Execution Phase** – Proprietary algorithms enable Composite to stream results from massively complex queries while the user query is still running. This lets applications begin consuming data from the server before the query completes execution, and results in substantial performance gains. Queries are decomposed and executed in the target data sources, and the streaming results are assembled and pipelined directly to the user’s client software. The parallel multi-threaded architecture of the server is truly exploited in this module where there are multiple queries with different run times across multiple data sources all required to resolve a single composite query. The execution engine also does dynamic plan re-writing based on actual results found at run time.
- **Query Plan Enhancement Phase** – An out-of-the-box installation of Composite Information Server will do a very good job of optimizing queries. However you can also enhance these query plans to improve its optimization results.

During plan generation and execution, extensive statistics are collected and logged for use in a powerful query plan visualization tool found within the Composite Studio. Composite developers use these insights to learn about the cardinality and distribution of data in the data source’s tables, which in turn allows query optimization to make better decisions about join algorithms and ordering.

Join options let you manually specify some of the run-time behaviors of the join instead of letting query optimization automatically determine the behavior. For example, you can select the actual algorithm with which the join will be processed.

Query hints can be included to help query optimization do its job. For example, you can provide the expected cardinality of either or both sides of a join. All available query hints are outlined and explained in the Composite Information Server Reference Manual.

Metadata Management Components

The Composite Information Server provides a number of features to manage the metadata required to create, maintain, and control views and data services.

The Metadata Repository stores and manages the links between Composite views or data services and underlying source systems. The repository handles explicit and derived metadata, such as metadata that may be generated for Web services.

Also, all data source metadata is “persistent,” so the repository can track and understand changes that have occurred to the structure of the underlying data sources. Any composition logic, packaged queries, and transformations are also stored here. The repository also captures all user and group information, event and log data, and system management and configuration information. Finally, the server ships with views, much like system tables, providing visibility into all metadata and underlying data, and the metadata repository can be exported to be shared, backed up, and to support migration.

Key metadata management subcomponents include:

- **Views and Data Services Metadata Repository** – Composite resources are the core building blocks within the Composite Information Server. This repository forms an enterprise ontology that is driven and created by the processes of building views and data services. The transformations, joins, entitlements, and data source and view dependencies captured as metadata in each Composite data service, are managed in this repository. Also captured — and searchable by users — is the business metadata associated with that view.
- **Metadata Import** – Composite can import both physical and logical metadata models designed in a wide variety of 3rd party modeling tools such as ER/Studio and ERwin.
- **Metadata Search** – Composite finds various resources such as columns, tables and annotations within logical views and data sources.
- **Version Control** – Composite supports improved collaboration capabilities, including resource externalization and resource locking to enable better resource sharing across projects and tighter design-time controls as new data services are designed and developed. (More about version control in the Team Development section below.)

Caching Components

Caching enables the Composite Information Server to hold onto a queried result-set for some period of time to allow it to potentially be re-used in satisfying another client request. The cached result-set might be an intermediate result that can be used to satisfy a variety of higher-level requests, or it might be a very specific final result that satisfies a single specific request. Either way, the Composite developer has complete control to specify when, where, and how often result-sets get cached.

Result-set caching is generally used for two reasons. First, because the cached result-set is immediately available to be used in satisfying a new request, overall latency of the request will be shorter, thereby increasing responsiveness to the client. This motivation for caching is sometimes characterized as a performance improvement, and the perception to the data consumer is definitely such. However, it is probably more accurate to characterize caching as a reduction in the amount of work that the system needs to do to satisfy a request. The system is more efficient, and this leads to improved responsiveness.

The second reason to use result-set caching is to reduce potential impact on underlying data sources. Because the Composite Information Server satisfies requests at the time they are made, each request usually results in corresponding data request to one or more underlying data sources. Depending upon the frequency of requests, it is possible to place significant load on underlying systems to satisfy the requests. In some situations this may be necessary and desirable, but in others cases it may be prudent to buffer requests to the underlying systems by using result-set caching. For example, a transactional system needs to be most responsive to the transaction creation process (e.g., order taking) during peak hours, and it should probably not be servicing reporting needs at the expense of transactional needs. By employing result-set caching, the Composite developer can protect an underlying system from too much activity while still providing recent (cached) data to reporting applications. In short, result-set caching

enables alignment of the currency needs of the consumer with the volatility of the underlying data, whatever they may be.

Result-set caching may be applied to any view or service created in the Composite Studio. By establishing a caching policy for a resource, the Composite developer instructs the Information Server where to cache the result-set and how often the cache is to be updated. The cached result-set may be stored in a local file, specified database, or in-memory. If the resource being cached is a procedure (including scripts, java components, and web services), multiple result-sets will be cached, each corresponding to a unique set of input parameters.

During query planning Composite's query optimizer recognizes that a particular resource has been cached, and it modifies the query plan to use the cached result-set instead of the original resource definition. This is interesting for two reasons. First, entire sub-trees of a query plan can be eliminated because of the availability of a cache for a resource. Second, because the cache itself is a data source (either a file or a database), the query optimizer can apply further optimization techniques on the cached result-set.

It is worth re-iterating that any view or data service resource at any level may be cached, and that resource may still be used to build other higher-level resources. This granular approach to result-set caching allows the Composite developer to truly take advantage of data encapsulation, and balance the volatility of the data with the currency needs of the consumer.

The caching policy for a resource is established at design time, and it allows the data designer to dictate storage location, update strategy, and update timing as follows:

- **Storage Location** – A result-set cache may be stored in a file, a database, or in-memory. Files work very well for small and relatively static result-sets, whereas a database works better for larger and more volatile result-sets. Data can be retrieved from memory an order of magnitude faster than it can be retrieved from disk storage, so it may be desirable to keep oft-used result-sets in memory instead of on disk. Composite supports in-memory caching in two ways. First, when result-set caching uses a database as the cache storage, that database can be directed to keep the result-set in memory. This mechanism is called “pinning”, and many traditional databases support this functionality. Second, it is possible to use an in-memory database product (e.g., Oracle's Times Ten database) as the result-set cache storage, which by definition will keep all cached result-sets in memory.

If a Composite Information Server instance is participating in a cluster, using a database for the cache storage allows a single result-set cache to be shared by all nodes in the cluster.

- **Update Strategy** – A result-set cache can be updated by performing a complete refresh or by incrementally modifying the cache with new changes. The complete refresh strategy is the most straightforward approach, and it can be automatically applied to any cached resource. An incremental refresh strategy requires specific business logic, which the Composite developer can provide through scripting or programming.
- **Update Timing** – A result-set cache may be updated periodically based on the clock, or it may be updated opportunistically when the cached data exceeds a certain age. It is

also possible to refresh a result-set cache manually only, which is very useful for caching reference tables that seldom change.

Procedural and Table Caching

In addition to the multiple result-set caching options described above Composite also supports Procedural Caching and Table Caching. For more information on these methods, please refer to the Composite Information Server User Guide.

Security Components

The Composite Information Server provides a complete set of capabilities to regulate who can perform operations on resources by authenticating users and enforcing user and group privileges. At run time, security regulates access to Composite resources and the data they return. At design time, security establishes ownership of resources, and enforces resource sharing and modification. Security is enforced for every request made to the Composite Information Server, and on all resources involved in a request.

Security Privileges

First, security is used to grant access to the Composite Information Server, which is called authentication. Second, security is used to allow and enforce privileges on resources in the Composite Information Server, which is called authorization. Resource privileges include:

- **Read, write:** At design time (i.e., when working in the Studio), these specify whether a user or group may see or modify a resource.
- **Select, insert, update, delete:** At run time, these specify whether a user or group may perform the given SQL command on a tabular resource (i.e., a view or column).
- **Execute:** At run time, this specifies whether a user or group may execute a procedure (e.g., a web service or a script), and also see the result of the procedure execution.
- **Grant:** At design time, this specifies whether a user or group may grant privileges for a given resource to users or groups. Of course, the owner of a resource always has grant privileges.

Privileges may be granted to users (individuals) or to groups, which are collections of related users. Users and groups may be created within the Composite Information Server, or the Information Server can leverage an external authentication system (e.g., Active Directory).

Security Authentication

When a user attempts to access Composite (that is, when they connect to the server) they must present credentials to identify themselves and allow Composite to authenticate them (usually in the form of a username and password). Users and groups may be defined in the Composite Information Server or in an external authentication system. When configured to use an external authentication system, the Composite

Information Server requests the external system to perform the authentication, which preserves the separation of roles.

Security Authorization

Whether at design time or run time, whenever a user attempts to perform any action on a resource (e.g., execute a procedure), the Composite Information Server checks the user's privileges, and the privileges of groups to which the user belongs, to determine whether the user should be allowed to perform the action. Authorization occurs on every resource involved in the operation, which means the user may need privileges on resources in addition to the resource they directly access. If a user does not have the required privilege to perform the action, an appropriate error is returned.

Executing Authentication and Authorization

When provisioning a data source, the administrator specifies whether the data source shall be accessed with a single shared set of authentication credentials, or with the specific credentials of the current user. If a single set of credentials are specified, all authorization performed in the underlying data source is done using those credentials. If user-specific credentials are specified (which is also called "pass-through authentication"), then authentication to and authorization within the underlying data source are performed using the current Composite Information Server user. To be clear, the security (authentication and authorization) in the underlying data source is enforced by the underlying data source using the given user (either shared or specific), so the behavior of the security in the underlying data source is unchanged by the introduction of the Composite Information Server as a client to that data source.

Row-level Security

The Composite Information Server supports row-level security using the same techniques that other systems use to support this concept. Specifically, the run-time system provides access to the current user, which can be used at design time to attach constraints to data being returned by a resource. The logic for performing row-level security can be arbitrarily complex, providing ultimate flexibility because the row restrictions are embedded in the data resource (e.g., a view).

For example, if you create a view that returns compensation information for employees (one row for each employee), it may be desirable to restrict access to these rows based on a management relationship. That is, only my manager and his managers (up to the company CEO) should be allowed to see my compensation information. You could place this constraint in the WHERE clause of the view's definition, using the employee ID of the current user, and only rows that satisfy the manager constraint would be included in the result set.

Token-based User Authentication

The Composite Information Server includes a pluggable authentication module that allows administrators to integrate authentication services from any external system, including those based on tokens, such as Kerberos. Built-in integration with Microsoft's

NTLM authentication paradigm will be included with Composite Information Server version 5.2 (which will leverage the open pluggable authentication module).

WS-Security Standards for Web services

Web service requests in the Composite Information Server are processed using a pipeline that may include any meta-operation on the request. The Composite Information Server includes WS-Security operations in this pipeline, and you can add your own operations as well. As WS-Security standards inevitably evolve, Composite will be able to modify the pipeline to accommodate the latest standards. The collection of operations that are performed in the pipeline is configurable by the administrator.

Data Encryption on the Wire

The Composite Information Server offers encryption for web services data transport. This encryption is often accomplished by using SSL as the connection mechanism, and it can also be accomplished by adding granular encryption to the web service processing pipeline. Wire encryption for JDBC/ODBC/ADO.NET connections is not currently available, but it is on the product roadmap to be included in a future version of the Composite Information Server.

Data Delivery Components

Data delivery allows data consumers to request and receive data from the Composite Information Server. There are a number of different protocols and request types, which allow a variety of data consumers to access the views and services offered by the Composite Information Server.

The Composite Information Server operates on a request-response paradigm: a client makes a request and receives data as the response to that request. This paradigm is available to clients as SQL-based queries and service-based calls. Support of both SQL-based queries and service-based calls provides greater flexibility and reuse across a wider range of consumers including Business Intelligence tools and Enterprise Service Buses.

- **SQL-based queries** – These work the same way they do with a traditional relational database. A client first establishes a connection to the Composite Information Server using a driver, issues SQL queries, and reads results. This form of the request-response paradigm establishes a persistent connection through which multiple request-response cycles may occur. When the client is finished, the connection is closed. The predominant data model is rows and columns of values (i.e., tabular data). There are no practical size limits on the result sets. Result sets can be as large as necessary to answer the query because Composite streams the results through the persistent connection as long as the client continues to read the result. All of the usual metadata that is expected from a relational database is available through this type of connection.

For SQL-based queries, there are three client drivers available: JDBC, ODBC, and ADO.NET. All three drivers adhere to the latest standards for the particular driver model, and all three drivers accept SQL queries based on the SQL-92 and core SQL-99 standards. If a data client has the ability to connect to a database using one of these

three mechanisms, they can successfully connect to and query the Composite Information Server.

- **Service-based calls** – Data services follow a web service model. Specific service operations are made available, and their interfaces are documented through a WSDL, which the client uses to form a request. There is no persistent connection. Instead a call is made, the result is returned, and the session is closed. The predominant data model is XML documents for both the request and the response, which places practical limits on the size of the result set.

For service-based calls, several web service standards can be used. These include SOAP, REST, and raw-XML. The request-response cycle can occur over HTTP or JMS (i.e., a message bus). There are many additional options within each of these standards, and also within the Composite Information Server processing pipeline, all of which are outlined in the documentation.

Transaction Processing Components

To support updates across multiple protocols and data sources, including flat files, relational databases, and web services, the Composite Information Server provides a transaction processing engine.

Transactional Insert, Update, and Delete are supported against underlying data sources, whether or not those sources offer native support for these functions. Composite supports compensating processes for data sources that are not inherently transactional like flat files and web services.

Applying this technology, transactional views or services can be created that function directly with the underlying data sources or act upon other transactional views or procedures. These data services can even be created that are semi-transactional. Users need only apply transactional overhead where it is needed.

As a result of this best-of-breed approach to transaction protocols, the Composite Information Server can thus provide true distributed transaction management across heterogeneous data sources, executing queries with maximum efficiency while protecting against core system failures.

Triggers

Triggers cause activity within Composite Information Server at a given time or as the result of a given event. They are a mechanism to make something happen inside of a Composite Information Server instance independent of the normal request-response cycle that clients use. For example, it may be desirable to have a particular data source swapped to a different source at night during a maintenance window. A developer could write a script that employs the Admin API to perform the switch, and then use a trigger to invoke it at the appropriate time. Triggers are also used internally by the Composite Information Server to implement some timed features in the Composite Information Server. For example, a timed cache refresh is implemented using a trigger.

A trigger resource has two parts. The first part defines what condition causes the trigger to “fire” or execute. For example, a trigger could be set up to fire whenever a particular pre-

defined system event occurs. There is an execution thread in the Composite Information Server that is charged with firing triggers based on their condition being met.

- **Timer** –The trigger will fire at the specified time, and it may also be set up to repeat periodically.
- **System Event** –The trigger will fire whenever the given system event occurs. There is a pre-defined list of system events in the Composite Information Server, all of which are documented.
- **User-Defined Event** – The trigger will fire whenever the given user-defined event occurs. User-defined events are posted by invoking a system procedure either from inside the Composite Information Server or through the public Admin API.

The second part of a trigger defines what action occurs as a result of the trigger firing. For example, a trigger could invoke a procedure. Once a trigger fires, its action is run in an independent execution thread. There are two main options for what a trigger does when it fires:

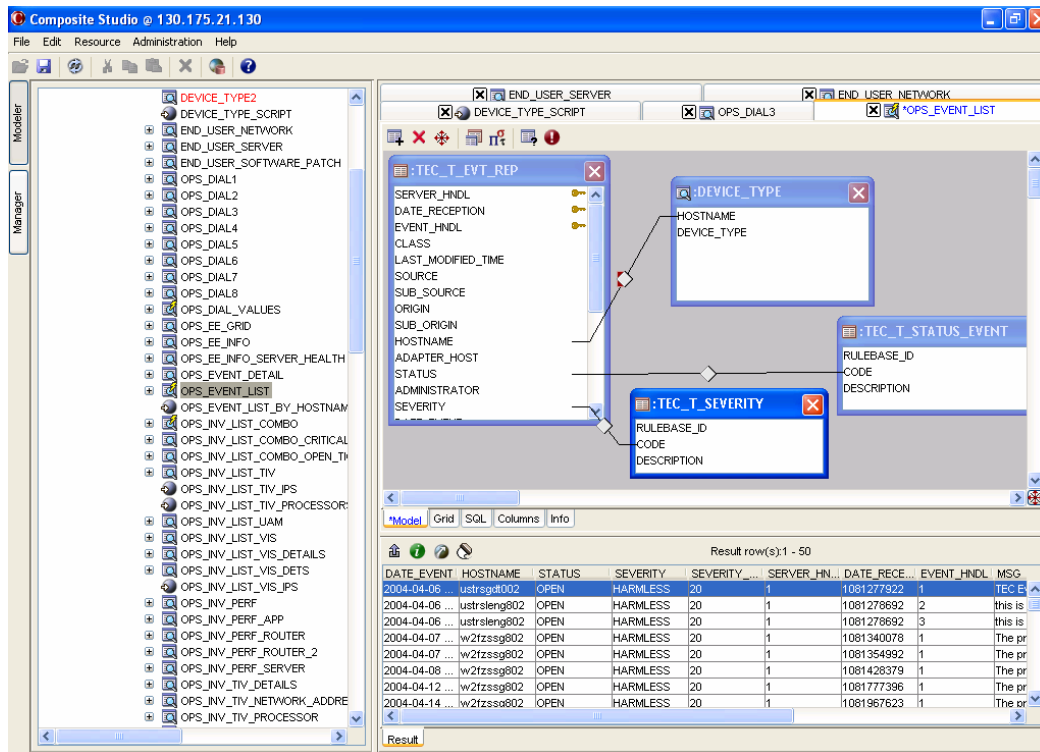
- **Execute Procedure** –This allows the trigger to invoke any procedure available in the Composite Information Server, which may include user-written procedures or system procedures.
- **Send an E-mail** – This is actually a superset of the previous action because it begins with a procedure execution, followed by sending an e-mail with the results of the procedure execution. You can also skip the procedure execution and simply send an e-mail as a notification.

Between these two actions, you can see how triggers can be employed to perform almost any task in the Composite Information Server: The trigger is simply the mechanism to make it happen, and the logic of what happens is implemented elsewhere.

COMPOSITE DEVELOPMENT

Composite Studio

The Studio is the primary modeling, view development, and resource management environment. The modeling environment presents a familiar resource-tree view of available physical data sources, a workspace area where queries are created and tested, and a data services area where Composite data services are published as views, data services, or caches. The Studio provides a data modeling environment that is similar in look and feel to other data modeling environments IT users have experienced, and that masks what is a very complex and innovative multi-data source type modeling process.



Composite Studio

Relational Views

Relational views, or just views, are resources in the Composite Information Server that integrate tabular data (rows and columns). A Composite view is conceptually equivalent to a traditional relational database view: it looks like a database table to the consumer. The view may be a final integration, which can be published for clients to query using SQL. Alternatively, a view may represent an intermediate result that can then be used with other data integrations (views or other resources). In the Composite Information Server you use the Studio to create and publish views.

Views in Composite are defined as a single SQL statement. The Studio development environment features a graphical view editor that allows you to create views using drag-and-drop techniques, and the resulting SQL is created automatically. You can also edit the SQL statement directly if desired (instead of using the graphical editor).

The key distinction between traditional views that you develop in a relational database, and those that you create in the Composite Information Server is that Composite views can read

the inside can take a variety of forms, and the Composite Information Server offers two different development environments to facilitate data service development.

- **Composite Studio** – For relatively straightforward data services, the Composite Studio can be used to create and publish data services that draw from a wide variety of data integrations. The developer has access to multiple languages for the implementation of the service logic, and the service can be published directly from the Studio.
- **Composite Designer** – For more complex services, especially services that need to conform to a pre-existing WSDL, Composite also includes the Composite Designer development environment. The Designer is a collection of Eclipse plug-ins that focus on XML manipulation. The Designer and Studio can share resources so that resources created in the Studio can be referenced by services being created in the Designer.

The data services artifacts created in Studio and Designer are hosted in the Composite Information Server. Data services can be provided in either SOAP or REST models. The transport can occur over HTTP or JMS (i.e., a message bus). There are many additional options within each of these standards, and also within the Composite Information Server service processing pipeline, all of which are outlined in the documentation for client access.

Team Development

Team development occurs when multiple people work on the same set of resources within a Composite Information Server, usually as part of a project development team. Team development carries certain challenges that are not present when one person is working on a set of resources. The Composite Information Server provides functionality to assist with those challenges.

There are two situations where Composite's team development features are important.

- **Shared Resources** – The first is when a team of people work on creating and modifying a shared set of resources over a period of time, usually during a project life cycle. This situation is similar to when a group of software developers works on a collection of source code files during the development of a piece of software. In both cases, a source code control system (SCCS) is helpful to keep track of the set of shared files, and to track changes and versions of the files. The Composite Information Server integrates with popular SCCSs to allow resources to be checked in and out of the SCCS directly from the Studio's user interface.
- **Versioning** – The second situation that occasionally occurs is when two people accidentally modify the same resource in a Composite Information Server at the same time. In this situation, it is important that neither person unknowingly overwrites the other person's work when they save their changes. To prevent this, the Composite Information Server always checks the version of a resource that is being edited against the version that has been saved, and the Composite Information Server will warn the person trying to save the resource that a newer version exists in the metadata repository. This safety feature is usually unnecessary, especially with well-managed projects, but it is certainly a welcome feature when this undesirable situation occurs.

You can think of each resource in a Composite Information Server as being analogous to a source code file in a software system. Whether it's a view, script, or any other resource, its implementation is the "source code" for that resource, and it is normally stored in the Composite Information Server's metadata repository.

In order to get a resource in or out of a source code control system (SCCS), the resource must exist in a file (because that's the unit that SCCS's work with). When you use Studio to "check in" a resource to an SCCS, Studio first externalizes the resource into a file using an XML-based representation of the resource, and then checks the XML file into the SCCS. Likewise, when you use Studio to "check out" a resource from an SCCS, Studio checks out the XML file from SCCS, and then imports the file into a Composite Information Server.

The Composite Information Server is natively integrated with Subversion (SVN) and Perforce source code control systems (SCCS). Composite also provides an open and supported API for integrating with any SCCS. Under normal circumstances, creating a new SCCS integration for Composite Information Server takes a couple days of effort, and the integration can then be deployed across the entire organization that has adopted a particular SCCS.

COMPOSITE DATA VIRTUALIZATION PLATFORM OPTIONS

Composite provides several optional products that extend the Composite Information Server and complete the Composite data virtualization platform architecture.

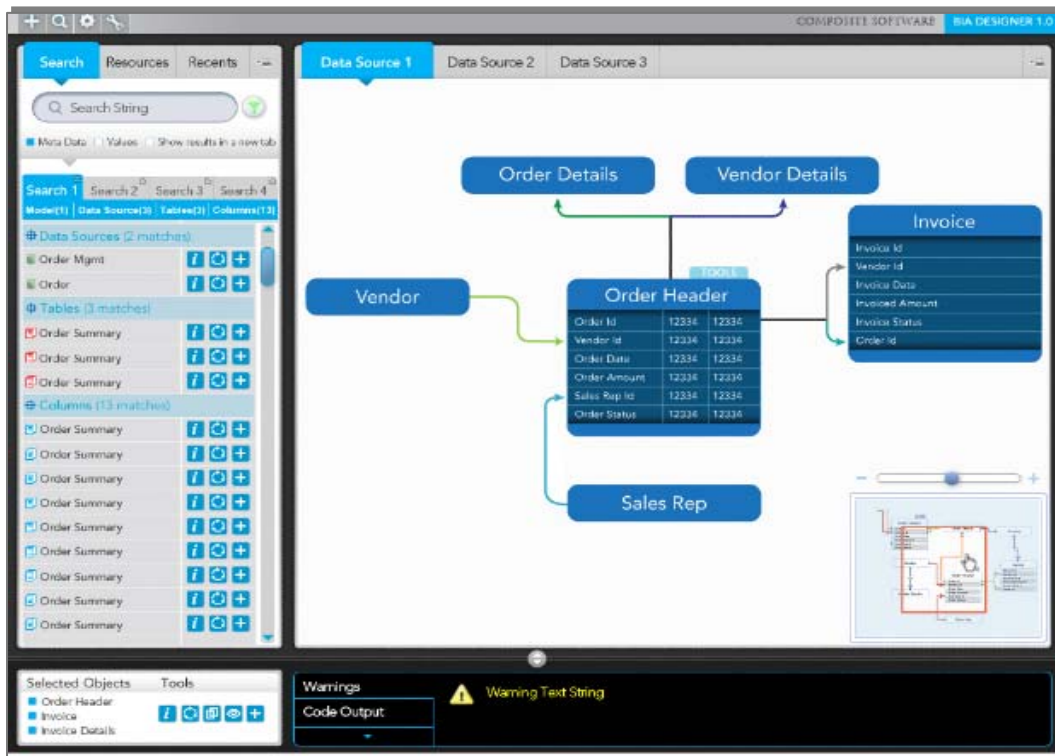
Composite Discovery

Composite Discovery is an option in the Composite Data Virtualization Platform. It uses a collection of strategies to uncover the data and reveal hidden relationships between entities in different enterprise data silos.

Composite developers use Discovery when they need to integrate and reconcile the data to meet a particular business need, but they don't know about the data they need or how data entities in different silos are related. Composite Discovery examines the metadata and data in the involved entities, and provides an entity-relationship diagram that can be used in the Composite Information Server to develop views and data services.

Discovery works by indexing the metadata and data in different entities, and then applying algorithms to determine whether a relationship might exist. It includes the notion of “confidence” in its suggestion of a relationship, and the data modeler can subsequently accept or reject a suggested relationship. Data modelers can also add relationships to their model that Discovery did not find, allowing for a complete federated entity-relationship diagram.

Once the relationships have been discovered and verified, the data integrator can create views that leverage the relationships. The collection of data sources, relationships, and views are referred to as a *model* in Discovery. A complete model or any of its components can be exported into a Composite Information Server directly from Discovery.



Composite Discovery

Composite Active Cluster Option

The Composite Active Cluster Option enables substantial scaling of Composite Information Server deployments while ensuring continuous availability to fulfill service level agreements. Clustering allows multiple instances of Composite Information Server servers to work together to handle user requests. Each Composite Information Server instance in the cluster is called a “node”. Clustering configurations are flexible based on reliability, availability and scalability requirements.

- **Scalability** – To achieve scalability (i.e., increased capacity to service requests), additional nodes running with their own computing resources (CPU cores, RAM memory, and hard disk storage) are added to the cluster as needed. For purposes of scalability it is not necessary for each node to run on a completely separate computer/server, as long as there are sufficient computing resources for multiple nodes running on the same server.
- **High Availability** – To achieve high availability, a cluster is created with nodes running on separate servers. Having the cluster span multiple servers provides protection from server hardware failures. As insurance, the capacity of the cluster must have enough processing headroom to accommodate the loss of one or more nodes so that all requests can continue to be serviced when a server failure occurs.

Composite Information Server’s clustering architecture is peer-to-peer such that no single node is more important than any other node in the cluster, and there can be up to sixteen nodes in a single cluster. With its active approach, all Composite Information Servers in the cluster are active at all times.

Distributing Queries across a Cluster

The usual practice for distributing client requests is to install a load balancer product in front of the Composite Information Server cluster (just as you would do for a cluster of app servers). The load balancer distributes requests among the individual nodes according to some pre-configured policy in the load balancer (e.g., round robin).

If a client request is a simple request-response (stateless) request (e.g., a web service invocation) then subsequent requests from the same client can be distributed to different nodes by the load balancer. Alternatively, if a client request creates a persistent (stateful) connection, as is the case with JDBC/ODBC, then the connection remains bound to a specific node in the cluster for the duration of the session. In this situation, if the node to which the client holds a persistent connection fails, the client would receive an error, and the client would take steps to re-establish a connection to the cluster (and the load balancer would distribute the request to another available node).

Metadata Synchronization

The cluster exists as a peer-to-peer network of Composite Information Server nodes. Each node holds its own copy of the metadata repository, but each node’s metadata repository is continuously synchronized with all other nodes in the cluster. If a metadata change is made to one node, that change is immediately propagated around the cluster to all other nodes.

Given this operational model, publishing new Composite Information Server resources to a cluster is a simple matter of publishing the resources to a single node. The cluster takes care of propagating the updates to all other nodes.

Caching Within a Cluster

A result-set cache in the Composite Information Server may be stored in a local file, a relational database, or in-memory. If a result-set cache is stored in a local file, then caching in a cluster works no differently than caching in a single Composite Information Server instance. In this case, each node creates and maintains its own copy of the result-set cache according to the caching policy of the resource. This may or may not be the desired behavior, so a second option exists as well.

If the result-set cache is stored in a relational database, the nodes in the cluster can share a single copy of the result-set cache. This arrangement provides better cache coherency, and makes more efficient use of resources. When the cluster is sharing a single cache, the nodes must coordinate the cache update procedure so that only a single node performs the update. This coordination is done through a bidding and assignment algorithm that is common in peer-to-peer architectures, and it ensures that only a single node performs the job of updating a shared cache.

Composite Application Data Services

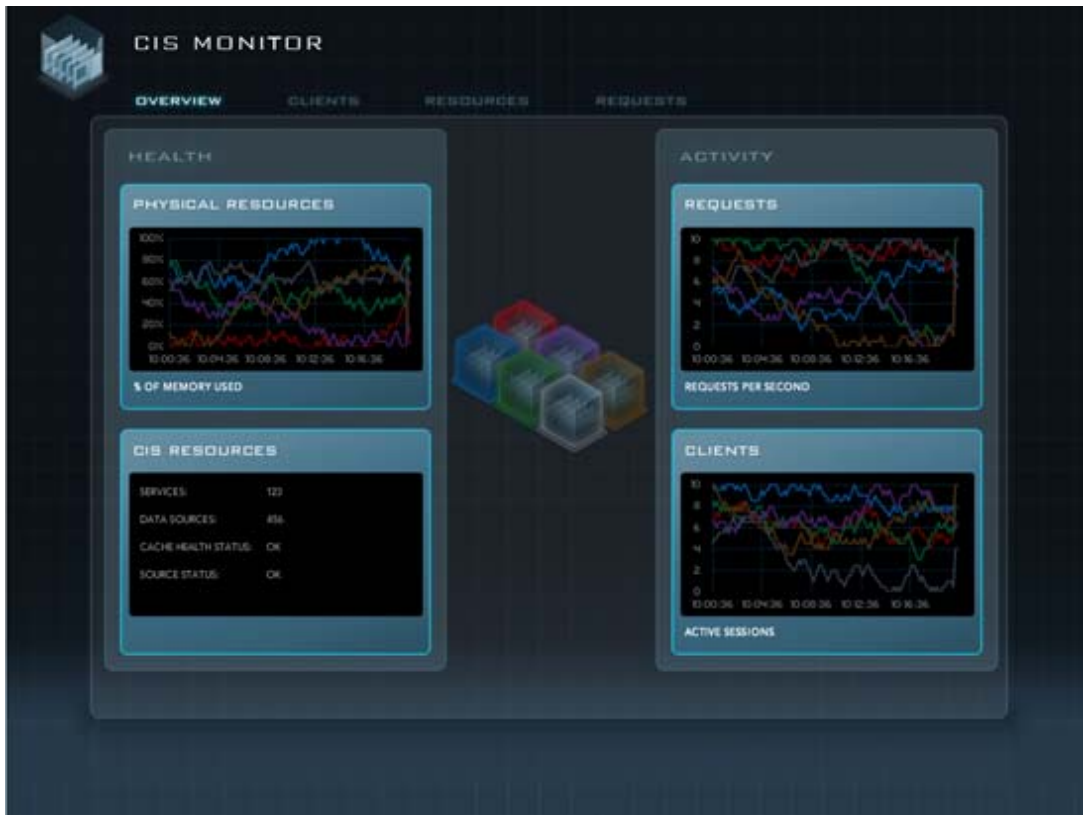
Composite Application Data Services™ enable access to packaged applications such as SAP, Siebel, and Oracle E-Business Suite, and deliver their content as pre-built, reusable data services.

Extracting data from packaged applications is not a trivial task. Specialized consultants are often needed to decipher these systems' proprietary interfaces and data structures. Combining packaged application data with other data sources is even more challenging.

Composite Application Data Services unlock data from packaged applications and deliver it as standards-based Web services or SQL views. Composite Application Data Services abstract the most common business objects, such as Orders, Invoices, Customers, and more. This enables SOA-compliant applications, as well as traditional front-end applications such as a reporting tool, to consume the content, providing users with the visibility they need to effectively run their businesses.

Composite Monitor

Composite Monitor provides comprehensive, real-time view of your Composite Data Virtualization Platform environment. Whether the environment is a single Composite Information Server or cluster of instances, Monitor displays all the system health indicators necessary to assess current conditions. If processes slow down or operations fail, IT operations staff uses these insights to guide the actions required to maintain agreed service levels.



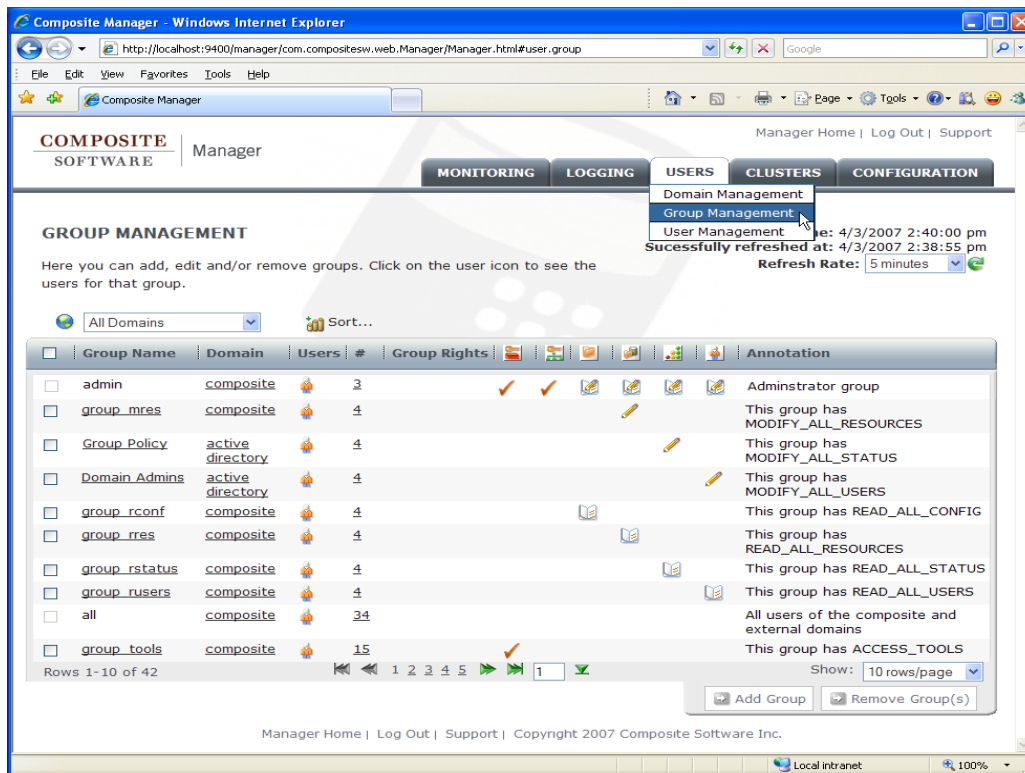
Composite Monitor

Monitor is designed specifically for Composite, so no additional systems integration or custom set up required. Features include:

- **Intuitive Displays** – Graphical displays are easy to read and interpret. You can quickly assess the state of a data virtualization environment by clicking through a few intuitive displays.
- **Configurable Alert** – Turn on and off various alerts and set tolerance gradients to meet desired levels of services.
- **Logging and Audit** – Twenty four hours of history is kept for review, investigation, and reporting.
- **Request Search** – Easily find requests to quickly diagnose problems and trace the sources of trouble.

COMPOSITE SYSTEMS MANAGEMENT

The objective of system management is to keep the Composite Data Virtualization Platform running well so the views and resources are continuously available to the consumer, and that they meet the service level agreements expected by the consumer. System management is used whenever you have a Composite instance in production with data consumers depending upon it. The focus of system management is the configuration, monitoring, and maintenance of the running Composite Information Server, rather than design and development activities.



Composite Manager through a Web Browser

Composite Data Virtualization Platform Systems Management Tools

Composite provides five tools that combine to support effective management of the Composite systems:

- **Manager** – This browser-based set of consoles facilitates configuration of, and visibility into, a running Composite Information Server instance. Among other things, the user can view and modify server settings, see active client sessions, and investigate running queries/requests. The Manager is the primary tool for managing a single Composite Information Server instance.
- **Monitor** – Composite Monitor is the primary tool for monitoring Composite Information Server instances. This browser-based client presents a dynamic view of a running Composite Information Server environment including all nodes and clusters. You can see collections of clients and their level of activity, monitor memory and CPU usage of the host machines, and display dashboards that include overall health indicators.

- **SNMP** – The simple network management protocol is a standard used in many network operations centers (NOC) to monitor the health and activity of components on a network. Although it is generally more focused on hardware assets than software assets, it is still important for software servers to participate in an SNMP monitoring strategy. Composite publishes a standard SNMP MIB that defines numerous available “traps” that can be monitored by the NOC dashboards and personnel.
- **Admin API** – Composite provides a public and documented API for programmatically manipulating all metadata inside a Composite Information Server instance. Although the Admin API serves a broader scope than system management, it includes many APIs that are useful for system management. For example, all Composite Information Server configuration variables can be read or changed using the Admin API. Those customers who wish to create their own custom system management strategies for a Composite Information Server can integrate their scripts with the Admin API.
- **Manager Pane in the Studio** – Although the Studio is primarily a development tool for data integration resources, it also has manager functionality built into it that includes much of the same functionality as the browser-based Composite Information Server Manager. Most of the functionality found in the Studio manager pane has been ported over to the Manager, and this trend continues. Eventually, the manager pane of the Studio may be deprecated and then removed to make the Studio product simpler.

CONCLUSION

The Composite Data Virtualization Platform as described in this paper, is a leading edge, yet proven approach for delivering critical information via data virtualization.

Purpose-built as a lower cost, more agile approach that overcomes data complexity and silos, Composite data virtualization provides business with the timely data it needs to meet today's business challenges.

Combining the Composite Information Server with multiple options, Composite's Data Virtualization Platform provides a complete data virtualization development and deployment solution that fits well with any IT strategy.

ABOUT COMPOSITE SOFTWARE

Composite Software, Inc. ® is the only company that focuses solely on data virtualization.

Global organizations faced with disparate, complex data environments, including ten of the top 20 banks, six of the top ten pharmaceutical companies, four of the top five energy firms, major media and technology organizations as well as government agencies, have chosen Composite's proven data virtualization platform to fulfill critical information needs, faster with fewer resources.

Scaling from project to enterprise, Composite's middleware enables data federation, data warehouse extension, enterprise data sharing, real-time and cloud computing data integration.

Founded in 2002, Composite Software is a privately held, venture-funded corporation based in Silicon Valley. For more information, please visit www.compositesw.com.