Creating an Agile Data Integration Platform using Data Virtualization

A Technical Whitepaper

Rick F. van der Lans
Independent Business Intelligence Analyst
R20/Consultancy

May 2013

Sponsored by

Table of Contents

1 Management Summary

Through the years, many organizations have built up massive amounts of enterprise data. Unfortunately, all this data is dispersed over many systems all using different technologies. This dispersion has several disadvantages. First, it has a negative impact on the quality of data. Second, it's hard for business users to get an integrated view of all the data. To the users, it feels as if enterprise data is deeply buried underneath a mountain of poorly integrated IT systems—as if it has been hidden in a *data labyrinth*.

More and more new IT systems need to retrieve and manipulate data stored in multiple systems. For many of these systems a dedicated integration solution has been developed. All these integration efforts have resulted in *data integration silos*. This is a highly undesirable situation, because eventually this approach leads to a *data integration labyrinth*. Data from a particular system is integrated with other data using different technologies and solutions.

Enterprise data is buried deeply in the data integration labyrinth and is therefore hard to exploit and diminishes its business value.

Due to this data integration labyrinth, data is buried deeply, and is therefore hard to exploit and diminishes the value of data to the business. What is urgently needed is a solution that centralizes and standardizes much of the data integration work: a *data integration platform*.

Data virtualization is a technology for integrating and transforming data coming from all kinds of data sources and presenting all that data as one unified view. When data virtualization is applied, an abstraction and encapsulation layer is provided that, for applications, hides most of the technical aspects of how and where data is stored. Because of that layer, applications don't need to know where all the data is physically stored, where it's coming from, how the data should be integrated, where the database servers run, how to insert and update the data, what the required APIs are, which database language to use, and so on. When data virtualization is deployed, to every application it feels as if one large database is accessed.

Data virtualization technology hides the dispersion of data.

This whitepaper describes this need for a data integration platform and explains how data virtualization can be used to develop a data integration platform. In addition, the data virtualization server of *Stone Bond Technologies* called *Enterprise Enabler Server* (EES) is described. The features of this product are listed and a short explanation of the product is given. Some of the distinguishing features of EES are:

- Use of a three-model approach for entering data integration specifications.
- Reuse of specifications to improve production and ease maintenance.
- Support for a large set of source and destination technologies.
- Extensive support for inserting, updating, and deleting data in multiple source systems.
- Active technology for "pushing" data to the applications using an event-based approach.
- Data workflow logic for developing composite applications.
- Integrated security and auditing features.
- One central repository for storing and managing all the metadata.

2 The Data Labyrinth

Data is the new oil.

Data is the New Oil – A popular saying in the IT industry is "Data is the new oil". Whether the analogy makes sense or not, data and oil have one thing in common: they're both well hidden. It is true that in selected places, oil comes to the surface, such as at the La Brea Tar Pits in Los Angeles, which makes it easy to discover and drill. But in most situations, oil is very hard to locate and also hard and expensive to drill. It can be hidden under a layer of rock miles thick, under dense layers of ice, or underneath the bottom of the ocean. Oil companies spend massive amounts of money on researching where they can find oil and, once they have located it, what the best way to drill is.

Like oil, data is hard to locate and hard to get to.

The same is true for an organization's data. Even though a lot of data is available, for some users it's hard to locate and access. They can't find the data they need, although they know the organization has it stored somewhere. Like oil has been hidden by nature, data has been hidden by the organizations.

To summarize, data is indeed the new oil, but primarily because it has inherited the same bad properties: hard to locate and hard to get to.

Dispersion of Enterprise Data – Usually, enterprise data has been dispersed over many systems. For example, data on a particular customer can be distributed across the sales system, the finance system, the complaints systems, the customer accounts system, the data warehouse, the master data management system, and so on. Usually, the underlying reasons for this situation are historical. Through the years, organizations have created and acquired new systems; they have merged with other organizations resulting in even more systems; and they have rented systems in the cloud. But rarely ever did new systems fully replace older systems. In most cases, the legacy systems still exist and are still operational. The consequence is a *dispersion of enterprise data*.

Data is dispersed across many IT systems all using different technologies.

Besides the fact that data is stored in many systems, an additional complexity is that such systems use a wide range of technologies. Some use SQL databases to store data, others use pre-SQL systems, such as IDMS, IDS, and Total, and there is more and more data available through API's such as SOAP and REST. The use of heterogeneous technologies for storing data raises the complexity to bring all that data together. In any case, it makes it hard for users to find the data they need. To them, the data has been hidden in a *data labyrinth*.

The Diminishing Quality of Data – Because of the dispersion of data, there is a considerable risk that data becomes incorrect. Types of incorrect data are missing data, misspelled data, and false data. All this incorrect data evidently diminishes the overall quality of data. The business value of data is inversely proportional to the data quality.

Besides the fact that data may be incorrect, data can also be inconsistent. The reason is that most systems manage their own data—their own version of the truth. One system may indicate that a particular customer is based in New York, whereas the other system shows he is based in Houston. Which value is correct? Which one signifies the truth? Especially if no synchronization

technology is used to help copy updates automatically from one system to another, inconsistent data can arise.

The introduction of new applications can also be the reason that data becomes inconsistent. Imagine a new app running on mobile devices that customers themselves can use to enter and change data. When the data is changed, the organization must enforce that all the versions of the data in all systems are synchronized accordingly. If this is not done properly, the data becomes inconsistent and its quality diminishes.

Improving the Quality of Data – Many technologies and methods are available to overcome problems with data inconsistency and incorrectness. For example, many data quality tools, data profiling tools, and master data management tools are available. However, most of them do not change the data in the production systems, they change data in data warehouses or master data databases. In other words, many of them are used in such a way that the data in the production systems remains uncorrected.

3 The Data Integration Labyrinth

Increasing Need for Data Integration – More and more new IT systems need to retrieve and manipulate data stored in multiple systems; see Figure 1. For example, a new portal for supporting customer questions may need access to a production database and a production application (an ERL system) to get all the required data to handle the requests. Another example is a website that customers can use to order products. To support the ordering process, data from different production applications may need to be queried and it may be necessary to add and update data in those systems. A third example is a report that shows what has happened with a particular business process. This report may also need to integrate data from multiple systems: a production database and a production application. Finally, the new generation of self-service reporting tools allows users to do their own data integration, because these tools have data integration technology built-in.

Data Integration Silos – For most of the new systems, data is currently integrated by developing a dedicated data integration solution for each one. If we take Figure 1 again, the website uses an enterprise service bus (ESB) to bring the data together and to insert data in the underlying production systems. The portal, on the other hand, uses a dedicated portal server, whereas the report uses a data warehouse with ETL-based solutions to integrate the data. And, as indicated, the reporting tool comes with their own light-weight data integration technology.

For each new integration project a dedicated data integration silo is developed.

If we consider all these new data integration efforts, one can easily speak of *data integration silos*, because for each application a dedicated integration solution is developed. This is a highly undesirable situation, because eventually this approach leads to a *data integration labyrinth*; see Figure 2. Data from a particular system is integrated with other data using different technologies and solutions.

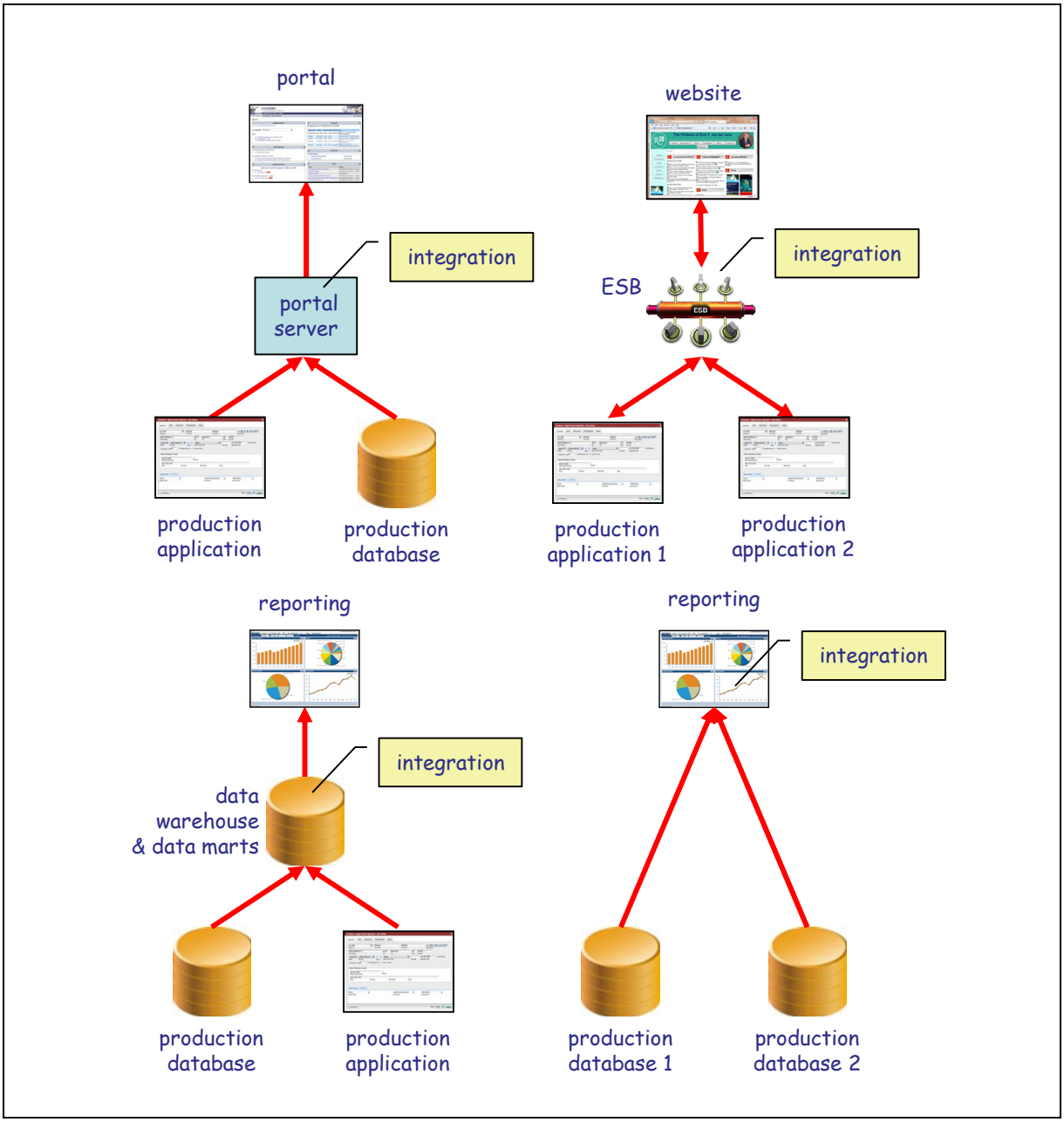

Figure 1 *For different systems, different data integration solutions are developed.*

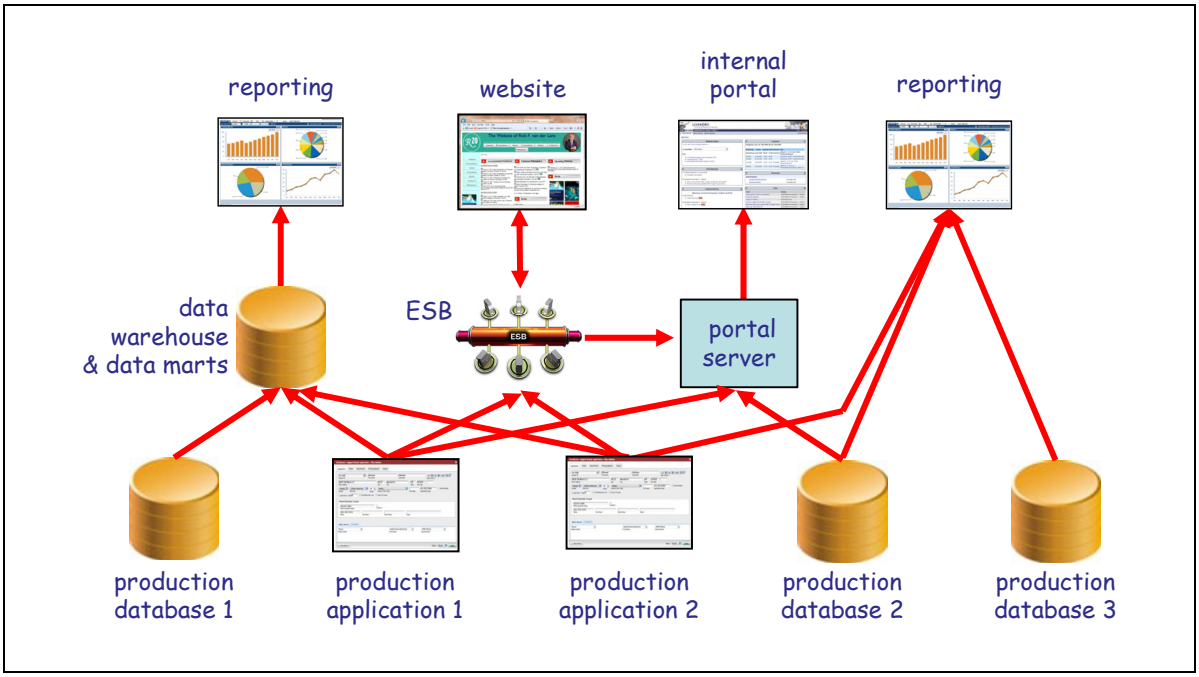

Figure 2 *Data integration silos result in a data integration labyrinth.*

Disadvantages of a Data Integration Labyrinth – A data integration labyrinth has several disadvantages:

- **Inconsistent views of data:** It's difficult to guarantee that the same data integration logic is applied in multiple solutions. For example, in Figure 2, both the reporting tool and the website integrate data coming from production applications 1 and 2. In this solution, it's hard to guarantee that the same integration specifications are applied by the data warehouse and the ESB solution.

- **Extensive Skillset Required:** Because different technologies are used, different skill sets are needed. For example, an ETL expert is not by definition an ESB expert or portal expert.

- **No Sharing of Data Integration Specifications:** The specifications used to transform and integrate data is replicated across all the data integration solutions. If, for example, gender codes must be standardized before they can be used, an implementation must be invented for each data integration solution separately.

- **Inflexible:** Because data integration specifications are not shared, changing them can be time-consuming and costly. This inflexibility also slows down organizations to become data-driven.

The Need for a Data Integration Platform – Due to the data integration labyrinth, data has become well buried and is therefore hard to exploit. One way of solving this problem is by rebuilding all the production systems from scratch. No explanation is needed when we say that such an approach is, for most organization, unaffordable and just plain impossible. An alternative solution to adopt

a technology that centralizes and standardizes much of the data integration work: a *data integration platform*.

In an ideal world, each organization has a data integration platform that can be used by any application to query and manipulate the data wherever that data resides. This should be a platform where data integration and transformation specifications are shared.

With a data integration platform applications can query and manipulate data wherever it resides.

The next chapter introduces one of the newer data integration technologies called *data virtualization* that can be used to develop a data integration platform.

4 Data Virtualization as Data Integration Platform

The Need for a Service Hatch – For historical reasons, the Netherlands has many Indonesian restaurants. What's amusing about most of them is the way orders are processed. When a customer has indicated what he likes to eat, the order is placed in front of a little service hatch that is normally located close to the bar. Next, the waiter pushes a bell. The little door of the service hatch opens quickly, an arm comes out, the order is picked up, and the door is closed. After a few minutes a soft bell rings to warn the waiter the ordered food has arrived. He opens the hatch, picks up the plates with food, closes the hatch, and brings the food to the customer's table. What's special is that the customers don't get to see the kitchen. Customers don't know whether the food was prepared fresh, whether it comes out of a refrigerator, whether the kitchen is clean and well organized or not, and so on. Everything is hidden. The only thing that customers see is the order that goes in and the food that comes out.

Our users need something comparable. They could also benefit from a service hatch where they can indicate what data they need without having to know how all the different technologies work or how the data should be processed. Whether the data has been dispersed or not, whether data is stored using different technologies, whether data is in the cloud or not, which technical API to use, all these aspects should be hidden by the service hatch. The machinery (i.e. the kitchen) should be completely invisible to the users.

Data virtualization is like that service hatch. When applications and users ask for data, the data virtualization server returns it, and when they request data to be inserted or changed, the data virtualization server executes the request. All the internal workings of the applications and databases are hidden by the data virtualization server.

All the internal workings of the applications and databases are hidden by data virtualization servers.

Data Virtualization in a Nutshell – Data virtualization is a technology for integrating, transforming, and manipulating data coming from all kinds of data sources and presenting all that data as one unified view. When data virtualization is applied, an abstraction and encapsulation layer is provided that, for applications, hides most of the technical aspects of how and where data is stored; see Figure 3. Because of that layer, applications don't need to know where all the data is physically stored, how the data should be integrated, where the database servers run, how to insert and update the data, what the required APIs are, which database language to use, and so

on. When data virtualization is deployed, to every application it feels as if one large database is accessed.

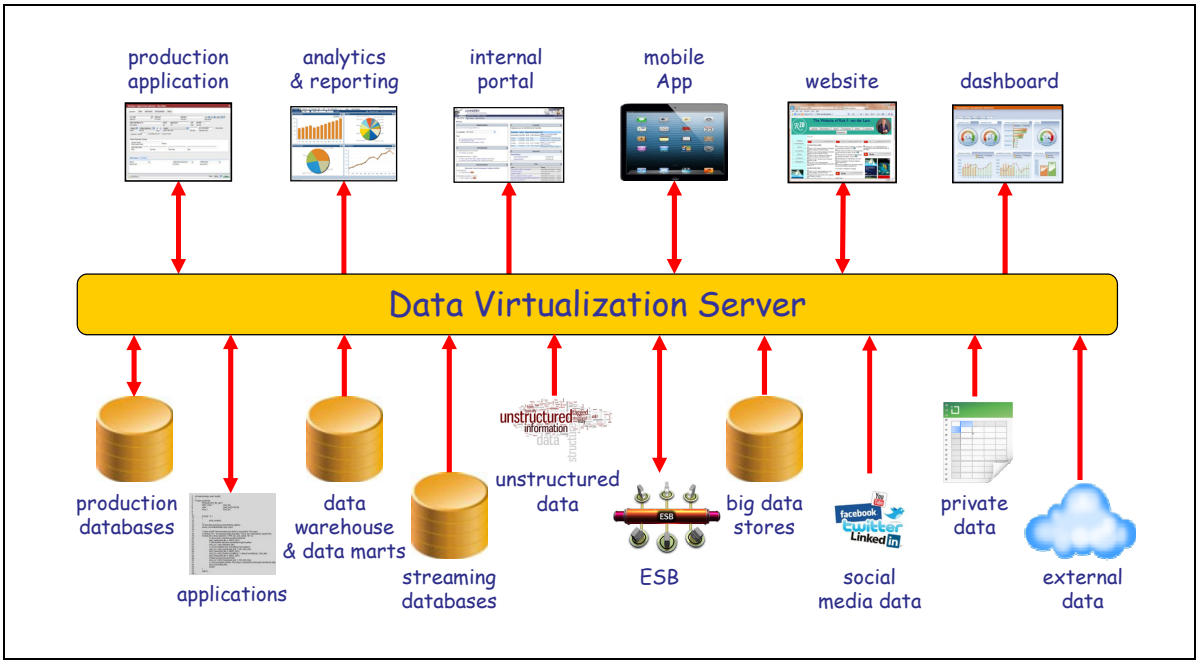

Figure 3 *Data virtualization servers make a heterogeneous set of data sources look like one logical database to the applications. The data virtualization server is responsible for data integration.*

The way modules are presented in Figure 3 suggests that some act as data consumers (the ones above the data virtualization server) and that the others act as data producers (the ones below the data virtualization server). In other words, the data consumers only request the data virtualization to do something, and the data producers are limited to delivering or processing data. This is not correct. Every module, meaning, every application, tool, database, message bus, can be a data consumer and/or data producer. So, a better way to illustrate a data virtualization server is through a circle; see Figure 4. Here, it's more clear that, for example, an application can request data from a database server, and vice versa is possible as well.

The definition of data virtualization[1]:

> *Data virtualization is the technology that offers data consumers a unified, abstracted, and encapsulated view for querying and manipulating data stored in a heterogeneous set of data stores.*

Technically, what this means is that when an application prefers to use the SOAP/XML interface to access data while the data source supports JMS, the data virtualization server transforms SOAP/XML into JMS. It also means that if the data in a source system has to be recorded before it can be used, the data virtualization server will perform this task. If data comes from multiple systems, the data virtualization server integrates the data and presents an integrated result to the application.

[1] Rick F. van der Lans, *Data Virtualization for Business Intelligence Systems*, Morgan Kaufmann, 2012.

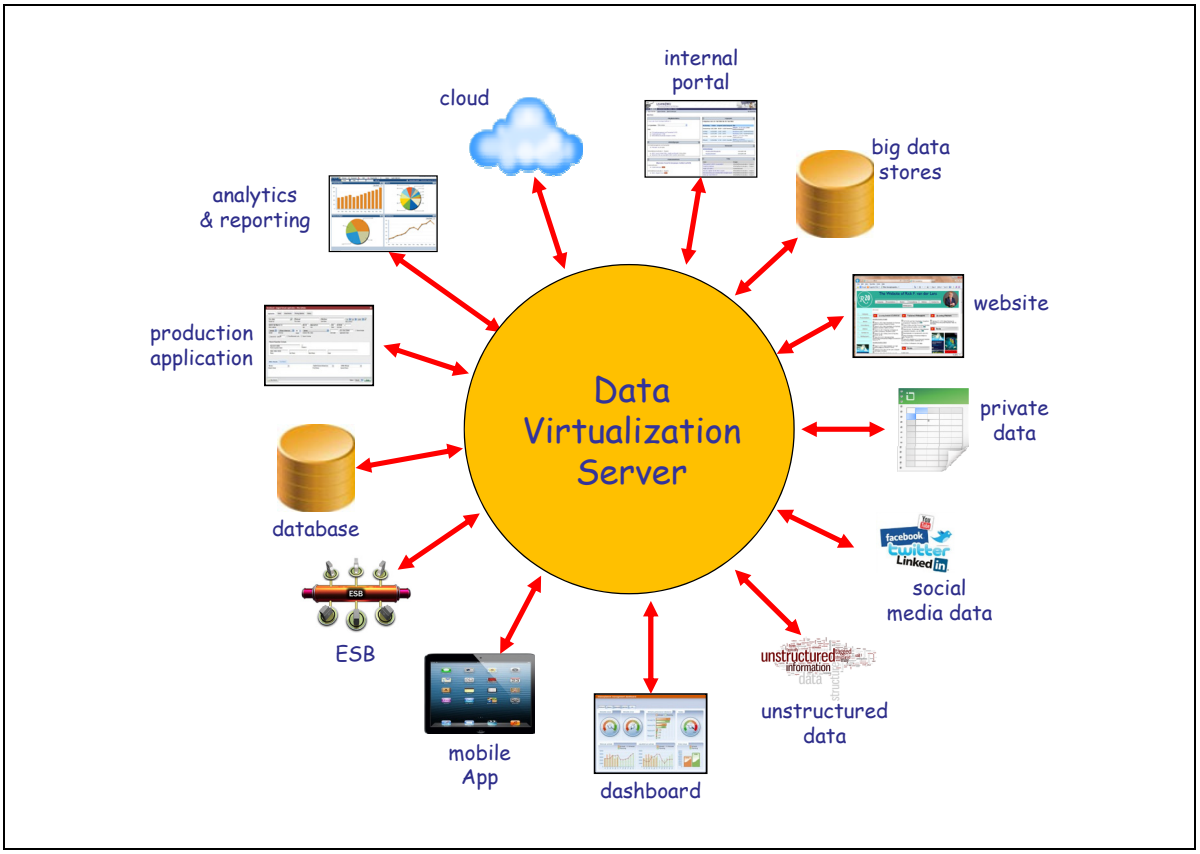

Figure 4 *Every application, database, or source system can act as a data consumer and/or a data producer.*

In other words, if the sources in Figure 3 use different storage formats and technologies, to all the applications, the data virtualization server presents them as one integrated set of data. In addition, different languages can be used to access the same data. Developers don't have to concern themselves with the technical aspects of the sources and the location of data, they can focus on what their applications have to do.

The Advantages of Data Virtualization – The key advantages of data virtualization when deployed for data integration are the following:

- **Increased Agility:** Because developers only have to focus on what data they need and what data must be manipulated, less code is needed and less code is easier to maintain. In addition, because the applications have been decoupled from the data sources, it's easier to make changes to the source systems without impacting the applications. All this dramatically improves the agility of an IT system.

- **Improved Time-to-market:** If data can be integrated faster, it simplifies the implementation of new systems for which existing systems must be integrated. In other words, it improves the time-to-market—organizations can react faster.

- **Improved Productivity and Maintenance:** As indicated, integrating data and systems with data virtualization is easier than with many other integration technologies.

- **Improved Data Quality:** When new data is entered using a data virtualization server, before the corresponding data is inserted and updated in the source systems, the data can be validated and corrected. In addition, when false data is retrieved from source systems, the data virtualization server can apply all types of data cleansing operations. Both forms of operations result in a higher level of data quality.

Data Virtualization as Basis for a Data Integration Platform – Because data virtualization can be used for many different styles of data integration, there is less need to deploy multiple data integration technologies. As shown in Figure 3, different application types can use a data virtualization server. Websites can use them for updating and inserting data in multiple systems, portals can use them to bring data together from multiple systems, reports can use them for integrating data, and so on.

This means that data virtualization can help to sanitize the current integration solutions. Instead of developing even more integration silos and expanding the integration labyrinth, an *integrated data integration platform* can be developed.

With a data virtualization server a real data integration platform can be developed.

Data Virtualization Versus Enterprise Service Bus – Besides data virtualization other integration technologies exist, such as ETL, data replication, enterprise services bus (ESB), and portal. Especially ESB technology, at a high conceptual level, resembles data virtualization. However, if we dive into the technical details, many differences become visible. The most important difference is that current ESBs offer low-level programming interfaces. Therefore, they require very technical skills of the developers. The consequence is that when ESB technology is used, the development time is higher and maintenance is more complex. Both negatively impact the time-to-market.

5 Key Features of Data Virtualization

This chapter describes a list of key features offered by data virtualization servers. To support the description of some of these features, Figure 5 shows a slightly more technical view of a data virtualization server than presented in Figure 3.

- **Data Source and Destination Encapsulation:** Data virtualization servers should support a wide range of APIs and languages, including SQL databases, Excel spreadsheet files, SOAP web services, message busses, and RFID data. This implies two things. First, applications can invoke the data virtualization server using their preferred API or language. This API or language doesn't have to be the language supported by the source system. Second, the data virtualization server is responsible for transforming the API used by the application to the API supported by the source system. In other words, data virtualization should encapsulate the technical details of the API and language of source and target systems.

- **Data Source Abstraction:** Data in source systems has a specific structure that has been designed to support its own applications. For new applications this may not be the most convenient structure and can possibly lead to unnecessarily complex application code. Data virtualization servers should be able to transform the structure of the data in the source system to a structure that fits the requirements of the new application. For

example, it must be possible to transform an hierarchical data structure into a flat relational structure (or vice versa), or to transform a structure showing data on the lowest level of detail into an aggregated data structure.

- **Data Transformation:** Data values from source systems don't always have the form that applications require. For example, values coming from different systems may have to be standardized to become matchable. Data virtualization servers must be able to transform values, therefore, they must be able to apply string and mathematical functions to values, concatenate columns, replace values by codes coming from a lookup table, and so on.

- **Data Integration Styles:** Different styles exist for integrating data, including on-demand integration, ETL, and data replication. A data virtualization server must be able to support all these styles, so that the developers can select the best solution for the job. Which style is selected, should be hidden for the applications.

- **Data Federation:** Data virtualization servers must be able to access multiple source systems for integration purposes. Accessing a heterogeneous set of source systems is usually called data federation. The fact that data is being federated should be hidden for the applications.

- **Transactions:** As indicated, data virtualization servers must be able to insert, update, and delete data in the source systems, even when the source systems are not databases, but, for example, ERP systems, or modern cloud-based systems. Data virtualization should not be restricted to query-only.

- **Distributed Transactions:** When a data virtualization server must access data from multiple source systems to answer a query coming from an application, it should give the application the feeling that only one source system is being accessed. Applications should not even be aware they're running *distributed queries*. The same applies for inserts and updates. For example, if an insert of data involves inserting data in a number of different source systems, the data virtualization server should be able to hide that fact. It should also handle all the transactional work to make it look like one atomic transaction. In other words, a data virtualization server should support *distributed transactions*.

- **Event-based:** Data virtualization servers must be able to "push" data to applications. When a data virtualization server processes a query coming from an application, it operates in a passive mode—it waits for the requests. Data virtualization servers should also be able to operate in a more active mode. When an event occurs, such a the creation of a file in a directory or the arrival of a particular message, the data virtualization server should be able to send data or a message to an application.

- **Centralized Repository:** To improve productivity and maintenance, a data virtualization server should support a centralized repository for storing all the meta data. Such a centralized repository minimizes replication of specifications and allows for *impact and lineage analysis*.

- **High-level Development Environment:** With some integration technologies it's necessary to use low-level development languages. Besides influencing productivity and

maintenance negatively, these highly technical skills are not always available in BI departments. Data virtualization servers should offer high-level development languages and graphical user interfaces.

- **Data Source Access Optimization:** Supporting an API is important, the efficiency with which the source systems are accessed may be as important. Data virtualization servers must be able to support optimization techniques when source systems are accessed.

- **Unified and Consistent Data View**: If all data integration work is handled by a data virtualization server, it can help provide a unified and consistent view of all the data. Whatever application, tool, or report is used, they all deploy the same data integration and transformation logic and therefore see the same data.

- **On-Demand Data Integration:** When an application requests data that has to be retrieved from multiple sources and has to be transformed, all that work is done when the request comes in: *on-demand data integration*. This is different from, for example, ETL solutions, where the result of data integration and transformation must be stored first, before it can be used. This means that there is a delay. The on-demand data integration capabilities of a data virtualization server should make it possible to present zero-latency data to applications.

- **Easy Migration:** When applications access database servers, a migration to another database server can be complex due to the differences in language. When the two have been decoupled by a data virtualization server, changing to another database server becomes easier, because a data virtualization server should be able to hide the differences. This makes it easier for organizations to exploit the advantages of new technologies in an existing environment.

- **Data Quality:** Data virtualization servers should support a rich set of functions that can be applied to help improve the data quality.

In the next chapter, Stone Bond Technologies and their data virtualization server called Enterprise Enabler Server are described.

6 Stone Bond Technologies

History of Stone Bond Technologies – Stone Bond Technologies was founded in 2002 and is headquartered in Houston, Texas. The company focuses on delivering software solutions for data integration. From day one, the goal of their core product, called Enterprise Enabler, was to develop a data integration product that could federate, align, and merge data from a wide range of data sources as it flows through sophisticated modeling, simulation, and operational systems but without the need for low-level programming. It would combine the features of other existing data integration technologies, including EAI, B2B, and ETL. The first version was released in 2005 and the current version is 8.025.

Products and Modules – Stone Bond's Enterprise Enabler consists of an integrated set of products and modules; see Figure 6.

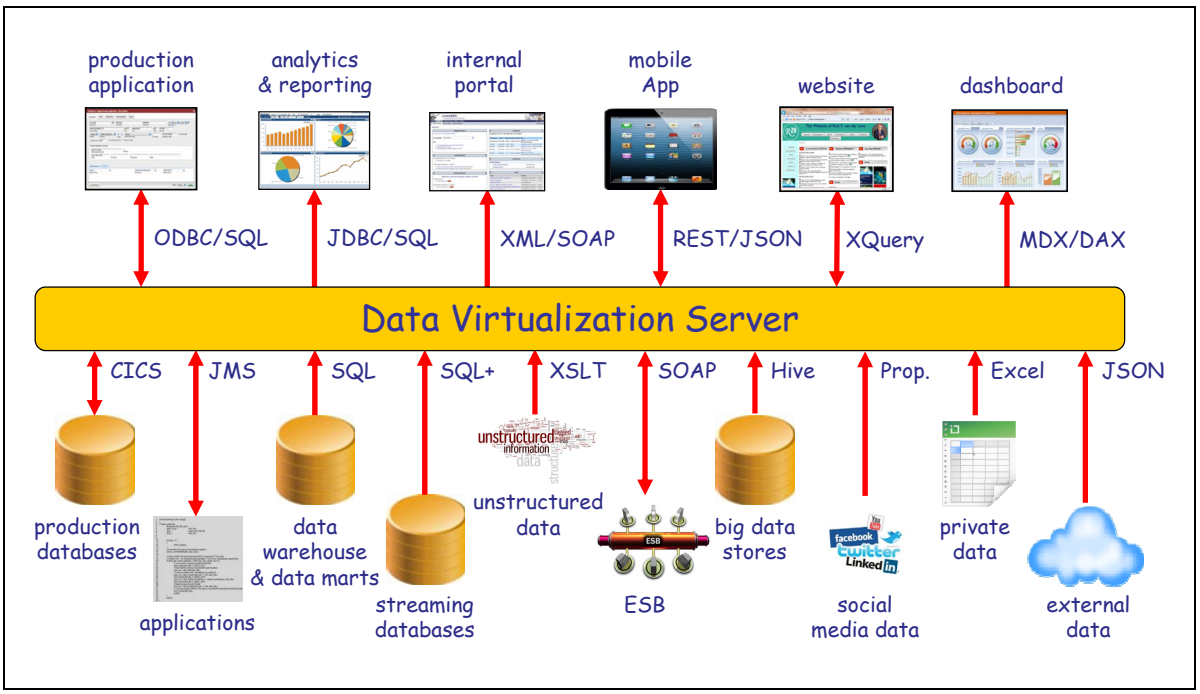

Figure 5 *When applications use a data virtualization server to access data, they can use the API and language they prefer. The data virtualization server translates that API and language to the equivalent supported by the source system.*

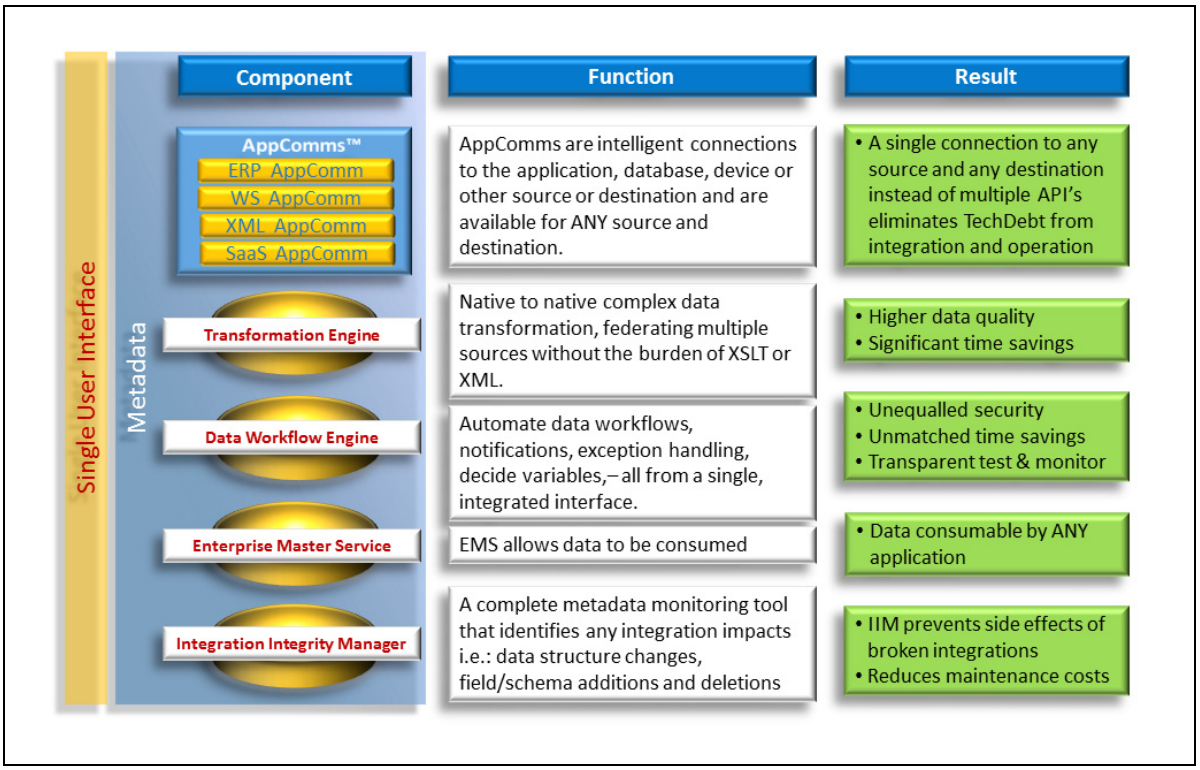

Figure 6 *This diagram shows all the products and modules that are part of Stone Bond's Enterprise Enabler. Copied with permission of Stone Bond Technologies.*

Descriptions of the various products and modules:

- **Enterprise Enabler Server:** This module is the heart of the product set. It's the data virtualization engine, the module that is doing all the heavy lifting. When developers configure parts of the system, they generate reusable metadata definitions. Everything within EES is defined as metadata and executed by the engines. At run time, the state of all the processing is visible, offering a high level of agility to respond to various situations. With respect to access security, permissions can be granted to generate, see, and act on metadata, and to deploy and access other activities. Support for auditing has been implemented by time-stamping every change to any object (metadata). This can be used to determine which users did what and when.

- **Enterprise Enabler Design Studio:** This is the graphical design environment in which all integration definition, testing, deployment, and monitoring occurs. This component is tied to a run time copy of the engine, so that when developers design, each component is validated against the actual data and systems. The Design Studio is a single, secure, and extensible integrated development environment including the Process Designer.

- **Appcomm Connectors:** For a large set of databases and applications, connectors are available to connect to applications, databases, devices/instruments, and standards like web services. These connectors are called AppComms. Technically, Appcomms sit between the sources and the Enterprise Enabler Server. They optimize access to the sources and they are configurable to accommodate the differences across a class of source systems. At design time, they provide the schema of the specific source or target system. At run time, AppComms react to instructions from the Transformation Engine.

- **Enterprise Enabler Transformation Engine:** This module coordinates all the reading and writing done by the AppComms to ensure proper timing from all sources involved. It performs data validation, alignment, and conversions across the data sources, and various forms of processing and performance activities.

- **Enterprise Enable Process Designer:** This module allows developers to create data flows. Data flows can be used to automate tasks. They can be seen as composite applications. The module offers a graphical, high-level development environment for creating and maintaining these data flows.

- **Data Workflow Engine:** The data flows built with the Process Designer are executed with the Data Workflow Engine. Execution of data flows include data integration via the transformation engine, as well as verification of data workflow conditions, processing, and sending out possible notifications.

- **Enterprise Master Service:** This module packages and handles on-demand data access for any of the integrations. It generates full CRUD processing and transaction rollback logic. For example, EMS automatically generates connection strings for ADO.Net and generates and hosts web services. In addition, EMS automatically generates and deploys end-user-aware bi-directional federation for SharePoint 2010 BCS External Lists.

- **Integration Integrity Manager:** IIM continuously monitors data source and target systems for changes in schemas. For example, should a field be changed or deleted in a database, IIM immediately performs an impact analysis and notifies all owners of

impacted integrations of all the changes that have been made and reports which objects in the integrated environment could be impacted. This pro-active approach can prevent potentially catastrophic situations.

7 Under the Hood of Enterprise Enabler Service

Each data virtualization server has its own architecture and approach. This chapter presents a high-level overview of Stone Bond's *Enterprise Enabler Server* (EES). In general, all the features one expects from a data virtualization server as described in Chapter 5, are supported by EES.

Transactional Support – One area in which the product excels and distinguishes itself from competing products is through its support for transactions. EES supports many features that make it possible to develop a data integration platform that supports a universal form of data manipulation (inserts, updates, and deletes). Two-phase commit is supported in case a transactions leads to changes in multiple, independent source systems. For example, in EES a transaction could mean an insert in Salesforce.com, an update in SAP, and a delete in an Oracle database.

The Three Model Approach – To access source systems and to make data in those systems available, EES uses a three model approach; see the screenshot in Figure 7. Each of the three large vertical areas corresponds with one of the models. The left-hand side contains the so-called *source templates* (model 1), the middle the *mappings* (model 2), and the right-hand side shows the *destination templates* (model 3).

.

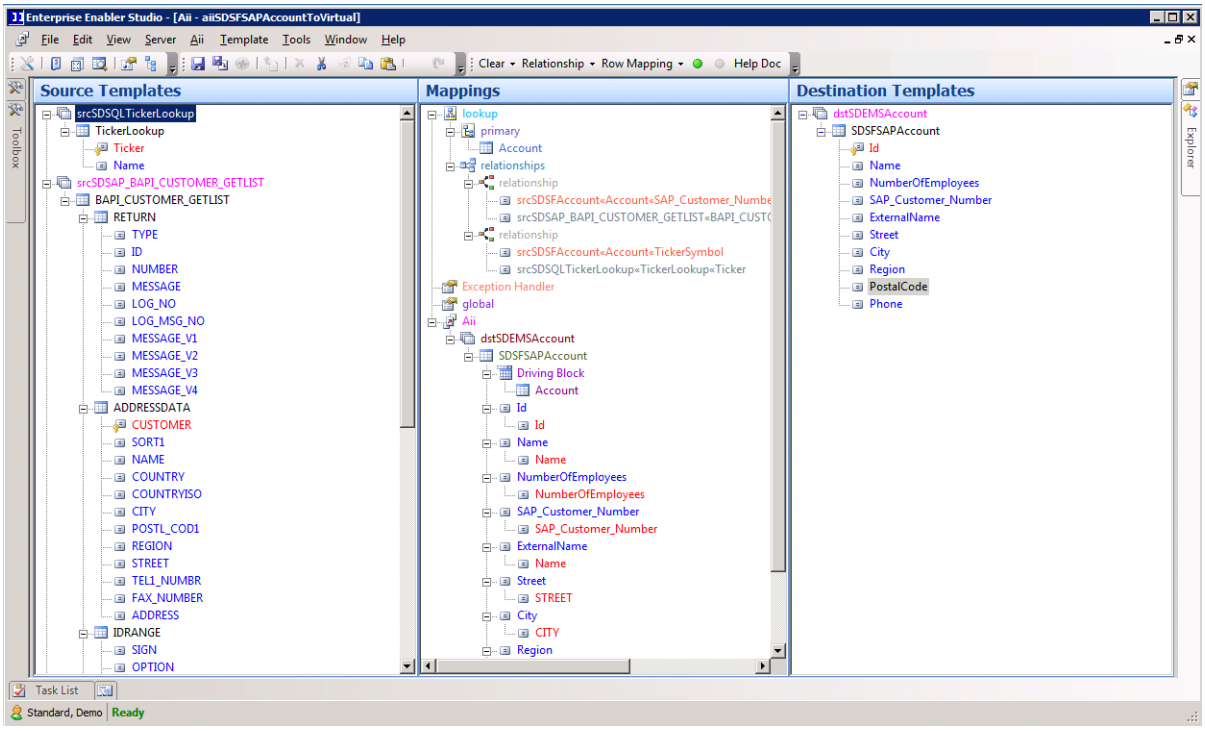

Figure 7 *In Enterprise Enabler Server, developers use a three-model approach for entering data integration and data transformation specifications.*

Source templates describe the technical connections with source systems. These source systems can be SQL databases, web services, message busses, and applications, such as SAP, and cloud-based systems, such as Salesforce.com. In a source template all the technical details needed to access the source system are defined. All the technical details are hidden behind the source template. To use alternative terminology, a source template *wraps* or *encapsulates* the source system and presents a conceptual view of the source system. This conceptual view only shows the structure of the data and the data itself.

Destination templates hide the technical details of the interface that the applications see. Here we define that applications use JMS, SOAP, or REST to access the source systems. Once the technical details are hidden, they are no longer a concern for developers responsible for the integration of systems. EES presents a more conceptual view of the way the applications want to access the data in the source systems. When EES is used for event-driven data flows, the destination templates encapsulate all of the destination application or data store's information just as it would the source.

Mappings indicate how the data and the data structures of source templates have to be transformed to destination templates. For example, a mapping can indicate that data from two different source systems must be joined together using a specific relation and only a limited number of columns should be presented. The mappings contain all the data integration, data transformation, and data cleansing specifications. Developers responsible for the mappings do not require knowledge of the technical peculiarities of the source systems or applications, because these are hidden by the source and destination templates. Developers focus on the conceptual views and the conceptual mappings.

For example, imagine that a portal must present data coming from three systems: a Salesforce.com system, data residing in a SQL database, and data available through a SOAP-based web service. In addition, that same data is also needed by a home-made Java application. The portal prefers to access the data via REST/JSON and the Java application via SOAP. First of all, for each source a source template is developed. Next, a mapping is defined that integrates the data from the three source templates. This is done by defining relationships between the templates and indicating projections, filters, and transformations. Finally, a destination template is defined that shows the data from the mapping via REST/JSON. For the Java application the only thing left to do, is to define the SOAP interface for the same mapping. It re-uses all the source templates and mapping specifications. Afterwards, if something changes in one of the source templates, that change has no effect on the portal or the Java application.

The above is an example of a data retrieval operation. The same story holds when new data must be inserted. Imagine that the data inserted by a customer through the website must be inserted in these same three systems: Salesforce.com, the SQL database, and the web service. The same development approach is used. The only difference is that the source templates now allow data to be changed. Again, the developers don't have to concern themselves with the fact that the data must be inserted in multiple systems, nor with the technical details of those systems. They don't even have to care about whether all the changes are processed as an atomic transaction. EES supports a distributed transaction mechanism to handle this correctly and safely.

The Data Workflow Logic Process – Besides passing and translating requests from the applications to the data sources, EES supports a visual programming environment in which *data flow logic* can be specified. Figure 8 shows an example of such a data flow.

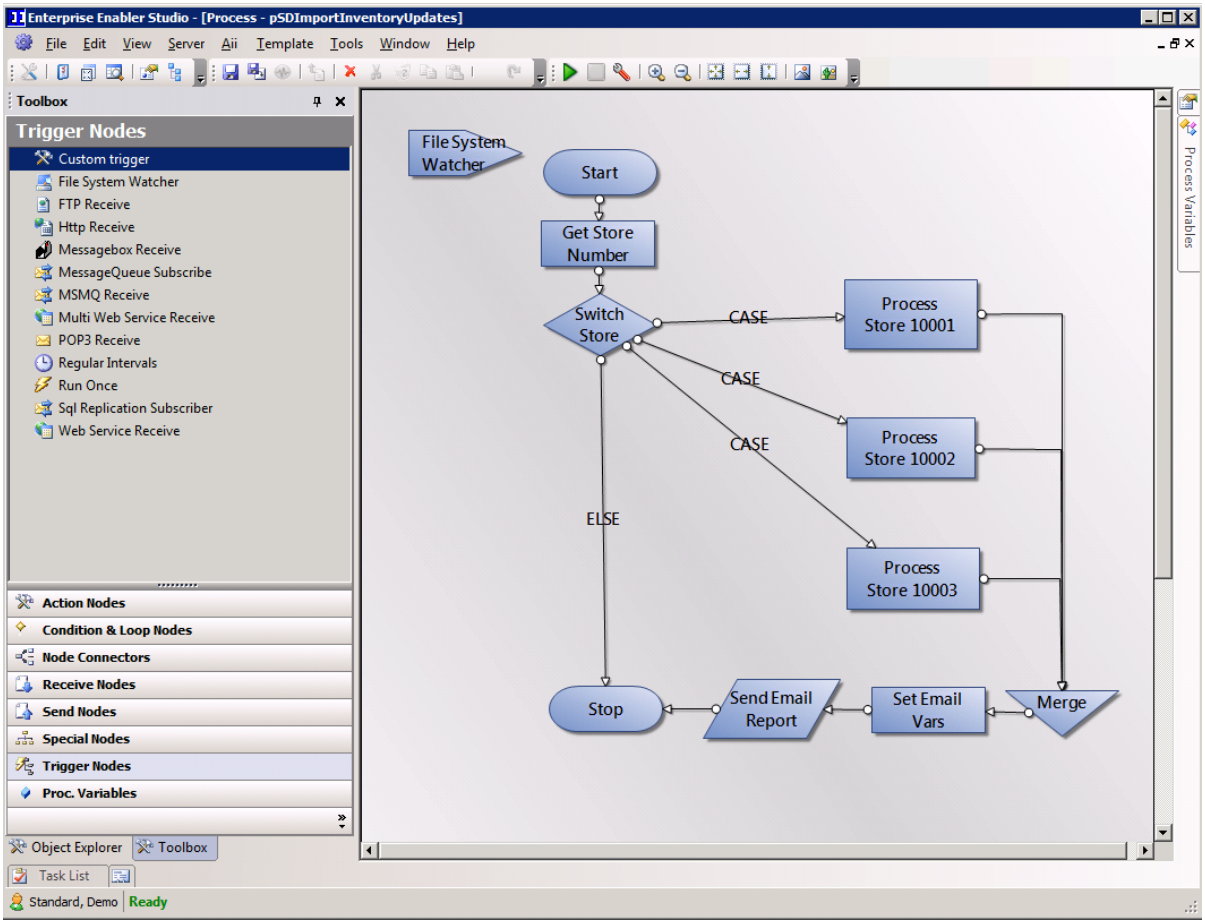

Figure 8 *With the data workflow logic, developers build logic for automating specific tasks.*

This data flow language is a distinguishing feature and enriches EES with procedural capabilities. Imagine that an airline has acquired a competitor. This airline now owns two reservation systems, two frequent flyer systems, and so on. Integrating all these systems into one can take many months, if not years. By placing EES on top of all those systems, one can simulate that the systems have been integrated. Imagine a new system has been built for inserting new frequent flyer members. However, the frequent flyer data is still stored in the two original systems. With the flow language, a data flow can be developed to determine in which frequent flyer system the data must really be inserted. When the new data is entered by a user, EES inserts the data in one of the two systems based on some properties of the data, thus hiding the fact that there are physically two systems.

The data flows also allow the EES to trigger activities. For example, when a new file is stored in a directory, when a certain type of message is detected on a message bus, or when a particular data element is changed, all these events can trigger the execution of a data flow. In Chapter 5 the feature event-based is described. EES supports this feature through the data flow language.

Most of the tasks done by a data virtualization server are data oriented, such as transforming data , integrating data, storing data, and retrieving data. Through the data flow language EES is also able to start up less data-oriented activities, such as sending emails, or placing messages

on a message bus.

To summarize, the data flow language transforms the EES data virtualization product from a rather passive environment that only reacts on what applications request it to do, into a more active and pro-active environment.

8 Two Case Studies

DermSurgery Associates – Houston based DermSurgery Associates specializes in state-of-the-art laser surgery procedures, dermatologic surgical procedures, Mohs surgeries (also known as chemosurgery), as well as various skincare and cosmetic services.

DermSurgery's IT's systems form a data labyrinth. Patient information is created and stored in numerous IT systems and databases. In addition, patient data is not limited to structured data stored in SQL databases, it also encompasses forms, images, and documents that capture the history of symptoms, diagnoses, and treatments, along with administrative information, such as insurances and referrals. There is not a single software system that covers all of the medical and business functions and information management that is required. One system is used for capturing initial information about new patients, one for scheduling, and one for billing; surgery details are captured in an EMR designed for that purpose; forms and documents are handled by Microsoft Word; and images are cataloged and stored in another system. Also, each surgeon maintains his own complete database with detailed information for each surgery performed.

Besides the fact that this has resulted in a situation where integrating data was very difficult and prohibitively expensive, this data labyrinth leads to several other business disadvantages. Valuable time was wasted re-entering data in multiple systems, locating paper charts, dealing with data entry errors, and transcribing details into the doctors' databases. All were challenges for DermSurgery to deliver on their mission statement. Paper-based patient charts were often needed by multiple people at the same time, making coordination of the data and maintaining the integrity of the data a major challenge. This data labyrinth was restricting the level of patient care DermSurgery could offer.

Therefore, it was decided to create an application with Microsoft Sharepoint that would use a data integration platform developed with Enterprise Enabler, the latter being responsible for all the data integration work. Currently, this solution synchronizes and consolidates information from all their systems into a single interface so that patient records, administrative information, and medical research data are all available in an integrated fashion and in a secure way. It offers access to all systems and processes all types of data, including all the forms, images, and documents.

Paradigm – Paradigm is a leading provider of enterprise software solutions to the global oil and natural gas exploration and production industry. They are headquartered in Houston, Texas.

Like so many organizations, Paradigm is using several ERP and several homemade systems. Sales performance data resides in Salesforce.com, financial reports are kept in Oracle E-Business Suite Financials, project management status data is stored in Journyx, and Solution Delivery Lifecycle status data resides in Microsoft Office Excel spreadsheets. All this dispersed data made it hard for management of their global operations to do more timely and efficient

reporting of business performance across departments. Creating a periodic snapshot of the state of the business operations worldwide could only be obtained after following rigorous sequential workflows among the company's leadership, which reduced collaboration efficiency and introduced bottlenecks. In other words, their data labyrinth was holding them back.

Paradigm wanted to be more responsive in its management of their global operations, through more timely and efficient reporting of business performance operations. Paradigm selected EES as data integration platform. EES was quickly deployed and configured as a composite application, providing the data integration and transformation between Salesforce.com, Journyx, Oracle E-Business Suite, the numerous Microsoft Excel spreadsheets, and other sources, without staging any of its data along the way. In other words, data integration is performed in an on-demand fashion.

About the Author Rick F. van der Lans

Rick F. van der Lans is an independent analyst, consultant, author, and lecturer specializing in data warehousing, business intelligence, data virtualization, and database technology. He works for R20/Consultancy (www.r20.nl), a consultancy company he founded in 1987.

Rick is chairman of the annual European Data Warehouse and Business Intelligence Conference (organized in London). He writes for the eminent B-eye-Network[2] and other websites. He introduced the business intelligence architecture called the *Data Delivery Platform* in 2009 in a number of articles[3] all published at BeyeNetwork.com.

He has written several books on SQL. His popular *Introduction to SQL*[4] was the first English book on the market in 1987 devoted entirely to SQL. After more than twenty years, this book is still being sold, and has been translated in several languages, including Chinese, German, and Italian. His latest book[5] "Data Virtualization for Business Intelligence Systems" was published in 2012.

For more information please visit www.r20.nl, or email to rick@r20.nl. You can also get in touch with him via LinkedIn and via Twitter @Rick_vanderlans.

About Stone Bond Technologies L.P.

Stone Bond Technologies L.P. is the provider of Enterprise Enabler, the leading business integration software for data virtualization, federation, and orchestration delivering the industry's fastest time-to-value. As the only integration solution to scale both up and down to meet business needs, Stone Bond provides software solutions that help businesses improve their bottom line by streamlining interaction among business systems.

To contact Stone Bond Technologies, call +1 713 622-8798, visit them at the Web at www.stonebond.com, or follow them on twitter @StoneBond. You can also read their blog published on http://agileintegrationsoftware.blogspot.com.

[2] See http://www.b-eye-network.com/channels/5087/articles/
[3] See http://www.b-eye-network.com/channels/5087/view/12495
[4] R.F. van der Lans, *Introduction to SQL; Mastering the Relational Database Language*, fourth edition, Addison-Wesley, 2007.
[5] R.F. van der Lans, *Data Virtualization for Business Intelligence Systems*, Morgan Kaufmann Publishers, 2012.