Data Integration in Reporting Systems using Enterprise Service Bus

A Thesis

Presented in Partial Fulfillment of the Requirements for the Degree of Master of Science in the Graduate School of The Ohio State University

By

Ketaki Koppal

Graduate Program in Computer Science and Engineering

The Ohio State University

2009

Thesis Committee:

Dr. Rajiv Ramnath, Advisor

Dr. Jayashree Ramanathan

Dr. Paul Sivilotti
Abstract

Data integration, the process of combining data from different sources into a unified view, has become a significant problem in the field of computer science. As the volume of data and rate of information exchange increases, data integration becomes more challenging. An ideal data integration system is the one that allows loose coupling among heterogeneous data sources, facilitates access management and maintains currency of the data. One of the reasons for data integration is the generation of reports. While the process of designing a reporting system inherits the challenges in designing a typical data integration system, it also has some characteristic issues of its own. One of the key issues is allowing mass customization. Existing reporting tools are limited in addressing one or the other of these challenges. This thesis is an attempt to come up with a customizable integration architecture that meets these requirements in the clinical domain. We will also discuss one of the web 2.0 technologies – data mashup - and how it can be leveraged to design an advanced solution for this problem.
Dedication

Dedicated to my family.
Acknowledgements

I would like to express my sincere gratitude to my advisors, Dr. Rajiv Ramnath for his constant guidance and encouragement and Dr. Jay Ramanathan for her insights and counsel that played an important role in shaping my research.

I would also like to thank Strategic Thinking Industries for their part in influencing and funding this work.

Lastly, I would like to thank my family, for it is because of their un faltering support that I am able to accomplish this milestone.
Vita

2005……………………………B.E. Computer Engineering, University of Pune, India

August 2005 – August 2007 …………… Software Engineer, Infosys Technologies Ltd.

Fields of Study

Major Field: Computer Science and Engineering
Table of Contents

Abstract ............................................................................................................................... ii
Dedication .......................................................................................................................... iii
Acknowledgements ......................................................................................................... iv
Vita ................................................................................................................................. v
Table of Contents ......................................................................................................... vi
List of Tables .................................................................................................................. ix
List of Figures ................................................................................................................ x

Chapter 1: Problem Statement ....................................................................................... 1
  1.1 Report Generation .................................................................................................... 2
    1.1.1 Challenges in Report Generation ...................................................................... 2
  1.2 Electronic Health Record (EHR) ........................................................................... 4
  1.3 STI Outbound Interfacing ...................................................................................... 8

Chapter 2: Related Work ............................................................................................... 12
  2.1 Extract, Transform, Load (ETL) ............................................................................ 13
    2.1.1 Discussion ....................................................................................................... 15
4.1.5 CBESB Component Flow Editor ................................................................. 50

4.1.6 CBESB Admin Console .................................................................. 52

4.2 Key Contributions ........................................................................... 55

Chapter 5: Conclusion and Future Work ............................................... 57

5.1 User Interface ............................................................................... 57

5.2 Mashup based solution .................................................................. 57

References .......................................................................................... 59
List of Tables

Table 1: Mashups – summarized. ................................................................. 29
List of Figures

Figure 1: Electronic Health Data – Pre-EHR................................................................. 6
Figure 2: Electronic Health Record ............................................................................... 7
Figure 3: STI Outbound Interfacing Framework......................................................... 8
Figure 4: STI Outbound Interfacing Schematic......................................................... 9
Figure 5: U of M Outbound Interface ......................................................................... 10
Figure 6: Simple schematic for a data warehouse - the ETL process ......................... 13
Figure 7: The Information Mashup fabric ................................................................. 26
Figure 8: Outbound Interface based on ESB .............................................................. 36
Figure 9: ChainBuilder ESB product schematic ....................................................... 39
Figure 10: ChainBuilder ESB format Editor ............................................................. 45
Figure 11: ChainBuilder ESB map editor ................................................................ 47
Figure 12: ChainBuilder ESB Driver File Setup ....................................................... 48
Figure 13: ChainBuilder ESB Connection File Editor ............................................. 49
Figure 14: ChainBuilder ESB component flow editor...................................................... 51

Figure 15: ChainBuilder ESB Admin Console – Components/Control .......................... 52

Figure 16: ChainBuilder ESB Admin Console – Assemblies/Control ............................ 54

Figure 17: ChainBuilder ESB Admin Console – Endpoints Status................................... 54
Chapter 1: Problem Statement

Data integration, the process of combining data from different sources into a unified view, has become a significant problem in the field of computer science. As the volume of data and rate of information exchange increases, data integration becomes more challenging. An ideal data integration system is the one that allows loose coupling between heterogeneous data sources, facilitates access management and enables currency of the data. Report generation is a specialization of data integration problem. A well-designed reporting system can have a significant impact on cost, performance and scalability of the entire enterprise architecture. While the process of designing a reporting system inherits the challenges in designing a typical data integration system, it also has some characteristic issues of its own. One of the key issues is allowing mass customization.
1.1 Report Generation

Every reporting application will typically pull data from different data sources, integrate the contents and format the report as per requirements. These reports would then be generated at scheduled time intervals and monitored for any failures. What varies between different reporting applications is the kind of data sources they interface with, granularity of content aggregation, reporting formats and the storage systems the reports would be pushed to.

1.1.1 Challenges in Report Generation

1.1.1.1 Mass Customization

As mentioned earlier, every reporting tool has certain standard functionalities and some variable features. Hundreds of man-hours are spent in implementing the standard functionalities in different ways depending on business requirements. If instead, users were provided with baseline architecture that has all the fundamental features implemented and then allowed to customize the variable part of the reporting system, we would be drastically improving the flexibility and efficiency of the system while lowering its cost. Some reporting solutions give application users total control over the content as
well as the look and feel of the report. But in the process, they lack the ability to integrate
with heterogeneous data sources thus failing to deliver a complete solution.

1.1.1.2 Heterogenous data sources

Reporting systems integrate with different sources to retrieve data. These sources could
be databases, or flat files such as XML, EDI X12, HL7 etc. Different databases have
different query languages. Reporting systems should be able to query the databases in the
language they understand. They should also be able to understand different file format in
order to be able to read from different data files.

1.1.1.3 Degree of Coupling

Degree of coupling in a software application indicates the extent to which its components
are dependent on each other. A tightly coupled system has a high level of dependency
among its components and this evidently has a lot of disadvantages. Firstly, any change
in one component requires making appropriate changes in the dependant components.
Code assembly and troubleshooting thereafter becomes a long and time-consuming
process. Thus it is desirable to have a loosely coupled architecture.
1.1.1.4 Access Management

Reporting systems integrate content, possibly pulled from different data sources, and present it to the end user. The report generated thereby should not expose confidential information or any data that the user is not authorized to see. For example, a report containing patient’s billing data should not disclose his social security number to an unauthorized user. This aspect is particularly emphasised in the healthcare domain by the Health Insurance Portability and Accountability Act (HIPAA).

Electronic Health Records (EHR) is a special kind of report. It is a digital record comprising of longitudinal collection of patient’s health information. This thesis mainly deals with the integration challenges in generating EHR.

1.2 Electronic Health Record (EHR)

Electronic Health Record is an evolving concept in the healthcare domain. The Health Information Management System Society\(^1\) defines EHR as:

“A longitudinal electronic record of patient health information generated by one or more encounters in any care delivery setting. Included in this information are patient
demographics, progress notes, problems, medications, vital signs, past medical history, immunizations, laboratory data, and radiology reports.”

Each time a patient receives service from an ancillary unit such as pharmacy, radiology or an event - such as filing a claim - is triggered, an electronic record may be created. Traditionally, these electronic records are not integrated. The data is captured and stays in silo systems, wherein each system has their own standards for vocabulary, user identification and patient identification. For accessing the patient’s complete record, one has to login and do a lookup across multiple applications. According to the statistics published by the Ash Institute for Democratic Governance and Innovation at Harvard University, in 2004, an estimate was made that 1 in 7 hospitalizations occurred when medical records were not available. Additionally, 1 in 5 lab tests were repeated because results were not available at the point of care.² Evans DC et al. claim that electronic medical records are estimated to improve efficiency by 6% per year, and that the monthly cost of an EMR is offset by the cost of only a few unnecessary tests or admissions.³ An EHR is designed with a purpose of combining data from the ancillary units and is capable of being shared within different healthcare settings to serve different needs.
An EHR is generated and maintained within an institution, such as hospital, clinic or physician office. The EHR in Figure 2 depicts the integration of healthcare data from participating set of systems for a single patient encounter. Each system in figure 2 stores its data locally. To share patient information, a system (or system user) must allow another system to access its files, or it must transmit a copy of the file to the other system. Once the file is identified for sharing, it can be integrated with other files, depending
upon the level of interoperability between the integrating systems. This sharing and integration is facilitated by providing interfaces that allow clinicians to access the silo systems through a portal or by importing data via a custom interface.

Figure 2: Electronic Health Record

Strategic Thinking Industries (STI) specializes in providing customized interfacing solutions for hospitals. STI Interface Command Center is an outbound interface designed
to automate and manage the flow of information from Labor & Delivery to Electronic Health Record Systems.

### 1.3 STI Outbound Interfacing

STI Interface Command Center is a pre-built solution suited to the needs of various hospital information systems.

Figure 3: STI Outbound Interfacing Framework\textsuperscript{5}
It interfaces with central hospital systems for medical record information, with billing systems to capture additional data for missed revenues and with Information Systems for additional back up and recovery coverage.

In the current approach, STI has a separate outbound interface implemented for each clinical system.

Figure 4: STI Outbound Interfacing Schematic

Depending on the requirements of the EHR system that it interfaces with, the clinical system has certain pre-defined report formats. The outbound interface designed by STI has those report structures coded in the implementation.
Figure 5: U of M Outbound Interface

The management console of this application allows the user to setup a schedule for kicking off the report generation process. The scheduled job will initiate the data retrieval activity. The backend system will process this data feed, generate reports and move them to appropriate directories to be pulled by the EHR systems. A listener polls for any error encounters and feeds in to the reporting server for error handling. This implementation works perfectly fine for the clinical systems. However, there are certain disadvantages to this approach. Firstly, it is a typical tightly coupled reporting solution. Each clinical system has its own set of standards and possibly distinct vocabulary, tight coupling
entails building a new interface for a new clinical system that STI would support. Considerable time and effort is spent on implementing similar functionalities. Besides, any enhancements in one software component require entire application upgrade. Secondly, users do not have the capability to freely configure and customize reports. Therefore, for any updates or requirement changes, appropriate code changes have to be made in the application and pushed as an upgrade. These factors make STI’s constant involvement in supporting the application inevitable. The scope for scalability is limited. Our goal here is to minimize STI’s intervention in the support and management of report generation activity by empowering the users to customize their own reports. Essentially enable the users to –

- Determine the source of content for data fields in the reports.
- Customize report layouts
- Schedule and monitor report generation activity
Chapter 2: Related Work

Several approaches, architecture techniques, frameworks have been designed and implemented for solving the data integration problem. We will look at the Extract Transform Load (ETL) technique implemented by data warehouses and how it is used for integrating data. Next, we analyze software product line architecture technique used for designing and implementing a suite of applications that have common core assets and are targeted towards a particular market segment. We will see how this architecture technique saves time and effort spent on implementing similar functionalities in different systems and why it still fails for the application in question. We then study enterprise service bus – a framework for implementing service oriented architecture – and see its application in implementing a complete solution for outbound interfacing.
2.1 Extract, Transform, Load (ETL)

Database management systems have been a time-honored solution to most of the data integration problems. Databases evolved to serve as a repository for integrating logically related records or structured data. As the volume of data and hence, presence of databases grew, so did the need for sharing or merging these data repositories. Data-warehousing concept came forth to provide a mechanism for unifying these data sources.

Figure 6: Simple schematic for a data warehouse - the ETL process.
The term Data Warehouse was coined by Bill Inmon in 1990, which he defined in the following way: "A warehouse is a subject-oriented, integrated, time-variant and non-volatile collection of data in support of management's decision making process". A warehouse system typically extracts the data from several sources, applies appropriate transformations and loads it into a single logical file. The WHIPS Prototype for Data Warehouse creation and maintenance defines data warehouse as a repository of integrated information from distributed, autonomous, and possibly heterogeneous, sources. In effect, the warehouse stores one or more materialized views of the source data. Data warehouse thus greatly facilitates reporting and analysis. Besides, there are some salient advantages to this approach – Firstly, query execution does not involve data translation and communication with remote sources thus making query execution and analysis, efficient. Secondly, the warehouse provides a common data model for all data regardless of its source. Thus the users don't have to perform query optimization over heterogeneous data sources.
2.1.1 Discussion

Extract, Transform, Load (ETL) Technique is the most common form of integration being done today and generally consists of a complex maze of applications, scripts, manual processes and FTP file transfers. The batch transfer job that extracts data, involves an application exporting its data to a common neutral format such as an intermediate database file or flat file with some kind of delimiter to separate data. The data representations at the source and receiving applications are likely to be different and hence the source file is run through a process for performing appropriate translation into the target format. The intermediate file is then FTPed to a staging area where it is cleaned up by a merge-and-purge process and imported from by the target data store.
ETL, in essence, performs bulk data transfer and batch update. Although it makes query execution and optimization efficient, this form of data transfer and data transformation is prone to problems. The major drawbacks of this approach are: unreliable data transfer, lack of data validity, undesirable downtime and latency of data gathering. ETL follows a tightly coupled architectural approach. If the data in the source database gets updated, the warehouse will contain the older data unless the entire ETL cycle is re-processed. In case of failures too, the ETL cycle might need a restart. If the transfer succeeded but load failed, a lengthy reconciliation will be needed to deal with partial import. If the extract and load process takes hours to complete, then compensating for failures can be very costly in terms of time. Even if everything works just fine, there is an inherent element of latency to a warehousing system due to the extract, transform and load processing times.

Data warehousing focuses on data storage, tools to extract, transform and load data into a repository, and the means to retrieve and manage metadata. A product line, on the other hand, refers to a development paradigm for creating a set of systems from core set of assets by using common means of production.
2.2 Software Product Line

Product Line technique has been long employed in the field of mechanical engineering – for manufacturing a set of CNC tools, set of automobiles, set of aircrafts etc. – where a pool of basic parts carefully designed for reuse, are configured and assembled in a common factory to create unique variation of the products that address a particular market segment. Software product line applies to software systems implemented in this way. According to the Software Engineering Institute,

“A software product line is a set of software-intensive systems sharing a common, managed set of features that satisfy the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way.”

Product line adoption involves moving from some form of developing software-intensive systems with a single-system mentality to developing them as a software product line. The adoption objective is to

- Have a core asset base, supportive processes, and organizational structures

- Develop products from that asset base in a way that achieves business goals
• Institute mechanisms to improve and extend the software product line adoption effort as long as it makes sense.  

The characteristic difference between a product line in manufacturing and that in software engineering is that manufacturing processes start with producing essential parts and exploit commonalities along the assembly line opportunistically, whereas software product lines start with predicting reuse in one or more products. Once the software product line has been established, components are identified as re-usable only if they can be applied to more than one system and re-using them would be cheaper than re-implementing them. Such components are grouped as core assets. These core assets are ideally designed with variation points – that is, places where they can be quickly tailored in pre-planned ways. Building a new system then becomes a task of accessing appropriate assets, tweaking them as per requirement, and assembling them into one system. The key here is to be able to –

• Identify the scope of product line – Scope of a product line identifies what systems is the organization willing to build as a part of its line or what systems it might be asked to build in future. The scope should be broad enough that sufficient number of products will be derived to justify the development effort,
and it should be narrow enough that the effort required to develop individual products do not outweigh effective cost savings.

- Identify and support the variation points – Variation points could be identified at any time during the software development life cycle. Some are identified during requirement elicitation, some at design time while some during implementation. Still others could be identified at the time of implementing subsequent products in the product line. Depending at the granularity of variations, they can be achieved by making wholesale changes at architectural level or by changing aspects of a particular element.

- Evaluate the product line architecture – An instance or variation of architecture should be evaluated to make sure that the variation points allow sufficient flexibility while keeping the performance costs and effort low.
2.2.1 Requirement Modeling

The essence of a software product line is the disciplined, strategic re-use of assets in producing a family of products by discriminating between what is expected to remain constant across all family members and what is expected to vary. G. Chastek et al., have described four interrelated work products that form the product line requirements model. Requirements are elicited, analyzed, specified, and verified through these work products. These work products are based on object modeling, use-case modeling, and feature-modeling techniques and bring forth the product line requirements:

- The use-case model specifies the product line stakeholders and their key interactions with the product line. Those stakeholders will verify the acceptability of the product line (and of the requirements).

- The feature model specifies the stakeholders’ views of the product line. Feature modeling involves modeling functional and quality features of the product line, identifying the commonalities, and abstracting the product-specific features.
• The object model specifies the product line responsibilities that support those features. Based on the information gathered through feature and use-case modeling, object modeling packages the system responsibilities into objects.

• The dictionary defines the terminology utilized in the work products and supports a consistent view of the product line requirements.

2.2.2 Discussion

Software Product Lines can potentially result in several organizational benefits including large-scale productivity gains, decreased time to market, increased product quality and decreased product risk, increased market agility and customer satisfaction and mass customization.

However, there are limitations to the extent this paradigm could be applied for implementing the interface for EHR systems. Firstly, each clinical system that STI interfaces with has standards and vocabulary for identifying and accessing data, possibly different from the rest. In such a scenario, re-use is limited. Secondly, mass customization, in software product lines, is achieved at the implementation level. Once the variation points are identified, different instances of the product line are implemented.
by making changes at the architectural level or by changing certain properties at the element level. For example, if a particular element needs to be omitted from the system, then the build procedure for that product can be modified so that it does not build that particular element. Changing aspects of an element require code changes. For example, in object-oriented systems, specializing or generalizing a class can bring variations. Users of the system cannot implement all these modifications. These variations need to be well established in the scope of the product-line, thus requiring STI’s constant involvement in implementing the interfaces.

2.3 The Enterprise Service Bus (ESB)

Dave Chappell, in his book “Enterprise Service Bus”\textsuperscript{12}, defines ESB as “a standards-based integration platform that combines messaging, web services, data transformation, and intelligent routing to reliably connect and coordinate the interaction of significant numbers of diverse applications across extended enterprises with transactional integrity.” An extended enterprise here represents an organization and its business partners, which could be separated by both business and physical boundaries.

Enterprise service bus is essentially a framework for implementing a service-oriented architecture. Its functionality can be compared to that of a data bus in the CPU, although at a higher level of abstraction. In that it acts as a message broker, enabling the
applications interfacing with it to communicate with each other. ESB thus makes services available to these applications in a unified manner, thereby removing the complexities of implementing point-to-point interfaces and data translations and making the services on the ESB inherently reusable.

Some of the important characteristics of ESB are:\(^1\)

- **Pervasiveness** – ESB can be adapted to suit the integration needs spanning an entire organization and its business partners.

- **Highly distributed, event-driven SOA** – The integration components in an ESB are loosely components. Hence these can be deployed across widely distributed geographic deployment topologies and yet, are accessible as shared services from anywhere on the bus.

- **Selective deployment of integration components** – Users can selectively deploy components such as adapters, data transformation services, and content-based routing services when and where they are needed, and can be independently scaled.
• Security and reliability – All components deployed on and communicating via ESB are ensured of reliable messaging, transactional integrity, and secure authenticated communications.

• Orchestration and process flow – An ESB allows data to flow across any applications that are plugged into the bus, irrespective of whether they are local or remote.

• Incremental adoption – An ESB can be used for integrating systems incrementally – starting with smaller projects and building into a much integration network, which can be remotely managed from anywhere on the bus.

• XML support – An ESB can take advantage of XML as its “native” datatype.

• Real-time insight – An ESB provides the foundation to enable real-time insight into live business data.

2.3.1 Discussion

Enterprise Service Bus is capable of providing the services necessary to address the challenges mentioned in 1.1.1 above. It enables service-oriented architecture, which
implies that the systems built on top of an enterprise service bus would be loosely coupled. It can seamlessly interface with different service providers and consumers thereby solving the problem of accessing heterogeneous data sources. Security and ability to provide real-time insights are the key features of ESB. We will see in the subsequent chapters, how enterprise service bus framework can be leveraged in the context of data integration for a reporting application.

2.4 Mashups

Mashup is a web 2.0 technology used to combine information from multiple data sources to deliver a new service. The term mashup implies easy, fast integration, frequently done by access to open APIs and data sources to produce results that were not the original reason for producing the raw source data. A mashup application is typically comprised of loosely coupled components: APIs, mashups hosting site, and the consumer’s web browser.
A mashup web application has two parts:

- A new service delivered through a Web page, using its own data and data from other sources.

- The blended data, made available across the Web through an API or other protocols such as HTTP, RSS, REST, etc.

The client generally accesses the mashup through a Web page containing the mashup. The mashup then accesses third-party data by querying relevant APIs or through other protocols such as RSS or REST and processes that data to add value for end users. This mechanism accomplishes more than just simple embedding of data from another site to
form a compound page – it links the data retrieved from different sources based on keywords used for querying and provides the user with a new service.

2.4.1 Mashup Classification

Presentation-centric/Data-centric

Mashups can be classified as data-centric or presentation-centric according to the services they create. Presentation mashups create a graphical presentation layer for the end users by combining data from different sources. A mashup displaying tourist locations pulled from Flickr on Google Map API is an example of presentation centric mashup. On the other hand, Data centric mashup focuses on data extraction and integration without requiring graphical presentation. RSS news feed is an example of data centric mashup.

Consumer/Data/Enterprise/Business

Depending on what are the sources and how they integrate, mashups can also be classified as

- Consumer Mashups
- Data Mashups
• Enterprise Mashups

• Business Mashups

Table below describes the characteristics of various types of mashups.
<table>
<thead>
<tr>
<th>Scope</th>
<th>What it does?</th>
<th>Benefit</th>
<th>Architecture/Customization</th>
<th>Market Positioning</th>
</tr>
</thead>
</table>
| Consumer Mashups (Presentation-centric) e.g. Google Maps | - Works at presentation layer.  
- Combine data from different websites and use a simple unified GUI to display information. | - Little or no coding  
- Drag and drop functionality  
- Expertise is not required. | - URL aggregation  
- Uses of widgets and gadgets  
- UI positioning | - For web-end users  
- Not suitable for implementing business applications |

Table 1: Mashups – summarized.

Continued
Table 1 continued.

<table>
<thead>
<tr>
<th>Scope</th>
<th>What it does?</th>
<th>Benefit</th>
<th>Architecture/Customization</th>
<th>Market Positioning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Mashups e.g. combination of several news feeds into one</td>
<td>Combines multiple data sources into a new data source</td>
<td>- Little or no coding</td>
<td>- Data aggregation</td>
<td>- Do not provide necessary capabilities to solve business problem</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Interface with multiple back-end systems</td>
<td>- Single representation of data from multiple sources</td>
<td>- Security is an issue</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Can cross organizational boundaries and hence very flexible</td>
<td></td>
<td>- Maintenance aspects of mashup not user-friendly</td>
</tr>
</tbody>
</table>

Continued
Table 1 continued.

<table>
<thead>
<tr>
<th>Scope</th>
<th>What it does?</th>
<th>Benefit</th>
<th>Architecture/Customization</th>
<th>Market Positioning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enterprise Mashups</td>
<td>Uses general mashup techniques within company's own internal applications.</td>
<td>- Faster Answers: user has access to both internal and external information without a middleman</td>
<td>- Integration of information from enterprise search engines, Web services, messaging systems, BI engines and data integration solutions with information from external services.</td>
<td>- No commonly accepted assembly model</td>
</tr>
<tr>
<td>e.g. creating a market</td>
<td></td>
<td>- Improved resource use: business users have the ability to assemble custom situational information solutions, from data sources they may have previously used.</td>
<td></td>
<td>- Uncertainty of truthfulness of data (how recent, what source, how is it processed.. Etc)</td>
</tr>
<tr>
<td>share report by</td>
<td></td>
<td></td>
<td></td>
<td>- Low levels of support by major software firms</td>
</tr>
<tr>
<td>combining an external</td>
<td></td>
<td></td>
<td></td>
<td>- Low awareness and</td>
</tr>
<tr>
<td>list of all houses sold</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>in last week with</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>internal data about</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>which houses agency</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>sold.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
- New opportunities:
  Because mashups let users combine data in New ways, they can support processes or decisions that were untenable even in the recent past.

realization of the potential of mashups by the business community.

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Continued
<table>
<thead>
<tr>
<th>Scope</th>
<th>What it does?</th>
<th>Benefit</th>
<th>Architecture/Customization</th>
<th>Market Positioning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Business Mashups</td>
<td>Focus data into single presentation and allow for collaborative action among businesses and developers. It is a combination of all the above mashups</td>
<td>- Useful for integrating business and data services</td>
<td>- Integrate business and data services</td>
<td>- Low levels of support by major software firms</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- To develop new integrated services quickly</td>
<td></td>
<td>- Low awareness and realization of the potential of mashups by the business community.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Make these services tangible to the business user thru user-friendly interfaces</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 1 continued.
2.4.2 Discussion

Mashups let users assemble internal and external data in an opportunistic way, thereby opening avenues for new business opportunities. While technologies and standards for implementing service oriented architectures have mainly focused on service-to-service interaction, a mashup approach to SOA can help facilitate the service-to-user interaction. In the later chapters, as we discuss the solution for the data integration problem based on ESB, we will see the need for mashup approach and analyze how mashups based on ESB could compliment and extend each others capabilities.
Chapter 3: Solution Approach

Figure 5 represents the current architecture. The reporting database server triggers the report generation process in accordance with the scheduled jobs. Data is pulled from appropriate data stores, reports generated and FTPed to destination directories. Besides, this server also takes care of error handling in the system. Clearly, there is an extensive scope for modularizing the architecture to minimize dependency and to expose re-use. The overall goal of the proposed system is to implement loosely coupled architecture that allows users to mass customize the report generation process for EHR systems while adhering to the access control.

3.1 Modified system architecture based on ESB:

There are two aspects of the proposed system - A dynamic User interface and adaptable Enterprise Service Bus. The enterprise service bus framework is leveraged for designing an architecture that enables sharing of services to facilitate effective data integration. The GUI on the other hand enables users to map report fields to a data source (usually a table
column in a database). While users visually map the fields, user interface should dynamically create XML code that ESB understands and uses for performing data translation. Figure 8 represents this architectural approach.

![Outbound Interface based on ESB](image)

**Figure 8: Outbound Interface based on ESB**

### 3.1.1 ChainBuilder ESB

ChainBuilder ESB\(^{15}\) is a Java Business Integration (JBI) compliant Enterprise Service Bus (ESB) provided by Bostech Corporation. ChainBuilder ESB consists of simple, easy-to-use GUI tools and drag-and-drop functionality for designing a process flow thus allowing developers to visually create and assemble ESB.
3.1.1.1 Key Features of ChainBuilder ESB

- Open Standard Based – JBI compliance enables solution architects to mix and match components and technologies from a list of JBI vendors and communities. ChainBuilder ESB also supports technology standards such as WSDL, HTTP, SOAP, JDBC as well as industry standards like Health Level 7 (HL7) and EDI.

- Integration with disparate sources – Chainbuilder ESB allows database integration with full transaction support using JDBC and/or ETL. It also supports message integration – meaning it can map XML as well as non-XML formats such as fixed, variable, EDI X12 and HL7. ESB supports traditional communication protocols like TCP/IP and FTP.

- Easy to develop – ChainBuilder ESB uses Eclipse IDE as the standard development platform. Besides, drag-and-drop functionality and accompanying wizards enable developers to easily create a process flow, map or other artifacts without hand coding.

- Easy to Extend – ChainBuilder ESB contains an extension framework called “User Point of Control (UPoC) that allows extending pre-built components by
plugging in Java or Groovy code components.

- Monitoring and Control – Chainbuilder ESB’s Administrative Console is an AJAX-based web interface used to perform remote monitoring, administration and configuration of alerts on run-time JBI components and service assemblies.
Owing to all these characteristics, ChainBuilder ESB was selected as a framework for performing data integration in the reporting application. It played a key role by providing following services:
• Message parsing

• Data transformation

• Content-based routing

• Sequencing

• JDBC access to relational databases

3.1.2 User Interface

The user interface displays a basic template for report that the user wants to configure. User can pick up from there and perform any modifications as need might be. All the report templates are implemented in the application before it is deployed on the hospital machines. However, not all the templates should be visible to all hospitals. When the application is deployed, the kind of clinical system will determine which templates are relevant and need to be made visible. The data source is specified during the application installation and is used by the JDBC service engine to pull out data.
The GUI that displays one template at a time, should have the following capabilities:

- The user should be able to choose the data source for each field in the report. So, against each report field should be displayed a list of tables/columns from which the user can select.

- User should be able to reposition labels/fields in the report so as to modify the report layout.

- User should be able to rename labels on the report.

- User should be able to add/remove fields from the report. Removing fields is straightforward. For addition of fields, there should be a list of fields from which the user can add. This list has all the columns from the tables that are used for populating the values in that particular report.

- A section should be provided for user to be able to upload custom images (e.g. company logo)

- In some hospitals, a barcode appears on the report instead of patient’s medical
record number (MRN). This capability should be taken care of.

3.2 Scope for future work

Once the skeleton of the report has been configured through the user interface, ESB takes over the task of generating periodic reports. The template provides the report layout that users can freely modify to define custom formats, layouts and mapping. An ESB-specific XML file has to be generated corresponding to this user defined map. This XML file is used by ESB for performing the data translations. The first main challenge was to be able to generate this XML file through the user interface.

Secondly, Chainbuilder ESB currently does not support generating a formatted file such as .doc or .pdf. It however, supports generation of an XML file. The next big challenge was to create a PDF report from this XML file. This conversion is possible, but requires a stylesheet. The stylesheet should essentially correspond to the layout that user will specify over the UI. We analyzed two approaches for doing this –

- Using FOP Bridge – This open source plugin for eclipse converts a FO compatible XML file to PDF using a stylesheet. The stylesheet in this case needs to be manually written or there should be some way to generate chunks of stylesheet components corresponding to each component of the template
displayed on user interface. Implementing this concept is not only time consuming but it will also lead to a lot repetitive work since many reporting tools (e.g. Jasper Reports\textsuperscript{16}) are already doing the same thing. Reporting tools such as Jasper, however, cannot provide the required solution since they face a major limitation in that they cannot interface with more than one data source at a time leading to a rigid and monolithic application.

- Lay, L"uttringhaus-Kappel\textsuperscript{17}, propose an alternative approach. The paper presents a method for generating Java Swing GUIs automatically from the XML Schema instances based on compile-time generated templates and the XML persistence feature of Java 2. This approach also requires stylesheet, but the stylesheet remains constant and does not change with changes in the GUI. It is the XML Schema that changes – automatically. This method not only helps save the effort of creating the GUI manually, but also eases the task of GUI re-generation, may the schema change.
Chapter 4: Implementation/Results

We will discuss the implementation with the help of application snapshots and see how the challenges mentioned in 1.1.1 are addressed.

The Access Control component is independent from the other business logic component as access control is a cross cutting concern. Any user who wants to use the outbound interface including its own UI would have to access it by first having to authorize and authenticate him or herself. Only those reports and hence datasources are made accessible to the users, which are relevant to his/her clinical system. This addresses the challenge related to access management (described in 1.1.1.4).
4.1 Snapshots

4.1.1 CBESB Format Editor

Figure 10: ChainBuilder ESB format Editor

Chainbuilder ESB supports standard file formats such as EDI X12 and HL7. Along with that we can also create custom message formats - fixed or variable. These custom files could contain simple comma separated values or complex group nodes and iterations of...
group nodes. The messages could be fixed length or variable, delimited or tagged. Chainbuilder ESB thus provides sufficient flexibility for accommodating custom formats and hence addresses part of the challenge related to accessing heterogeneous data sources (mentioned in 1.1.1.2)

4.1.2 CBESB Map Editor
Figure 11: ChainBuilder ESB map editor

The Chainbuilder ESB map editor provides certain pre-built services and support for various data transfer operations such as copying or combining fields, performing math operations, connecting to and reading from a database, lookup and iteration on group of fields. It also supports conditional processing and miscellaneous functions such as sending an output file, adding comments and other user formatted fields (e.g. datetime).

The user can also write scripts for custom processing, to perform any operation not supported by ESB.
4.1.3 CBESB Driver File Editor

Figure 12: ChainBuilder ESB Driver File Setup

CBESB currently supports access to MYSQL, SQL Server, Oracle, and Derby. The driver file editor is used for adding the appropriate database drivers to the assembly.
4.1.4 CBESB Connection File Editor

Figure 13: ChainBuilder ESB Connection File Editor

The connection file editor uses the driver file (created in previous step) and allows setting up a connection to that database. The connection properties can be configured here. Although creating connection file is optional (connection properties can be created
whenever needed), it is recommended since the same connection file can be used at all the places thus minimizing the probability of error and making configuration or changes to the configuration easy.

4.1.5 CBESB Component Flow Editor
The component flow editor is one of the most important editors. The entire assembling of components is done through the component flow editor. The component flow in previous diagram shows an assembly consisting of a script component, JDBC service engine, Transformation Engine, a file writer component and a sequencer at the heart of it all. The sequencer controls the flow of data and sequence of operations. The edge incoming to sequencer indicates the component that provides input. The edges marked with numbers indicate the sequence in which individual services are invoked or operations performed. Each component acts as a placeholder. In the sense, a component is simply an interface and the actual implementation of that function is provided in individual XML files (since XML is the native type). In Figure 14, the script component requires a script that typically has a query for retrieving data from the database. The transformation component requires a map file (created in 4.1.2) for performing the data translation. The JDBC component uses the connection file created in 4.1.4 and so on. Such an implementation ensures that the components are not hard-coded and hence loosely-coupled. This approach also supports mass-customization because the any configuration provided by the user can be converted to XML and provided to ESB through these components. Thus, without making any changes in ESB, the users can customize the layouts and mapping as per required. Thus component flow diagram is the key in
addressing challenges related to loose coupling and mass customization (1.1.1.1 and 1.1.1.3)

4.1.6 CBESB Admin Console

Figure 15: ChainBuilder ESB Admin Console – Components/Control
This AJAX-based web interface allows users to deploy and control JBI components and service assemblies. It also allows monitoring status of the assemblies, setting up alerts and error logging for debugging purposes. The screenshot above displays some pre-built components/services that can be started, shutdown or refreshed using the controls provided. User can deploy any custom components through this console.
The service assembly is deployed and started.

The end points can be monitored. Status “Up” indicates normal functioning. Any red
marks in status indicate some sort of error or exception that can be looked up in the Error DB.

Once the assembly is up and running, the input file would be fetched from the appropriate directory or database could be queried for input based on the connection properties. This file would be appropriately processed and sent out to the output directory. The time interval depends on the configuration set up at the input component (either script or file reader).

### 4.2 Key Contributions

In the context of challenges discussed in Chapter 1, the goal of my work has been to implement a service-oriented architecture for report generation and management that empowers user to mass customize the reports as well as the process for EHR systems. Key contributions of my work are as follows:

- *Re-designed the outbound interface.* The existing interface was extensively coded in .NET platform. A complex maze of modules controlled the report generation process. Implementing an interface this way took a few months of man-hours. The architecture was then re-designed to make use of the enterprise service bus
framework. After the initial set up and installation of required software, assembling services and components in the ESB to achieve the same configuration as the original interface took not more than 4 hours.

- *Generation of HL7 reports.* The new architecture was successfully utilized for generation of HL7 reports.
Chapter 5: Conclusion and Future Work

5.1 User Interface

The component flow in the service assembly guides the process orchestration in the ESB. ESB however does not require hardcoding the map file in a service assembly. This aspect of the enterprise service bus can be leveraged to support mass customization when implementing the outbound interface. The user provides the translation rules for the map file when he customizes the reports through the user interface. This user interface has to be implemented.

5.2 Mashup based solution

Mashup development is essentially a user-driven application design process involving ad-hoc composition of several data outputs of existing services or API’s to deliver value-added services or applications in a new context. For the end-users, this approach brings flexibility and speed in delivering new valuable services. A mashup application is
typically based on data sources such as REST\textsuperscript{19} or SOAP-based services as well as lightweight protocols like RSS or ATOM. The content aggregation may be done either by the server or by the client while the browser controls user interaction and rendering of the application.\textsuperscript{20}

Enterprise Service Bus allows providers to develop and publish services. Consumers access these services via standard interfaces while ESB handles the underlying complexity of translation and orchestration of the services. Thus enterprise service bus as well as mashup integrate with data sources and enable service composition, but mashups do it differently – they allow users to drive the design process.

In the context of the need for user interface mentioned in 5.1, future work would be to implement an enterprise mashup built on top of an ESB. An ESB can help expose archaic or complicated data sources as mashup-consumable sources, while mashup support and extend ESB by providing great input for ESB tools\textsuperscript{21}. This collaboration can be leveraged to create dynamic, user-centric solutions that serve over and above the data integration problem in reporting systems.
References


3. Evans DC, Nichol WP, Perlin JB (April 2006). "Effect of the implementation of an enterprise-wide Electronic Health Record on productivity in the Veterans Health Administration".


9 http://www.sei.cmu.edu/productlines/


15 http://www.chainforge.net/

16 http://jasperforge.org/projects/jasperreports

17 Patrick Lay, Stefan L"uttinghaus-Kappel; Transforming XML Schemas into Java Swing GUIs


21 John Crupi and Chris Warner. Enterprise Mashups Part I: Bringing SOA to the People