

Regis University

## ePublications at Regis University

---

Regis University Student Publications  
(comprehensive collection)

Regis University Student Publications

---

Spring 2010

# Assessing the Flexibility of a Service Oriented Architecture to that of the Classic Data Warehouse

Michael Pastore  
*Regis University*

Follow this and additional works at: <https://epublications.regis.edu/theses>



Part of the [Computer Sciences Commons](#)

---

### Recommended Citation

Pastore, Michael, "Assessing the Flexibility of a Service Oriented Architecture to that of the Classic Data Warehouse" (2010). *Regis University Student Publications (comprehensive collection)*. 447.  
<https://epublications.regis.edu/theses/447>

This Thesis - Open Access is brought to you for free and open access by the Regis University Student Publications at ePublications at Regis University. It has been accepted for inclusion in Regis University Student Publications (comprehensive collection) by an authorized administrator of ePublications at Regis University. For more information, please contact [epublications@regis.edu](mailto:epublications@regis.edu).

**Regis University**  
College for Professional Studies Graduate Programs  
**Final Project/Thesis**

**Disclaimer**

Use of the materials available in the Regis University Thesis Collection ("Collection") is limited and restricted to those users who agree to comply with the following terms of use. Regis University reserves the right to deny access to the Collection to any person who violates these terms of use or who seeks to or does alter, avoid or supersede the functional conditions, restrictions and limitations of the Collection.

The site may be used only for lawful purposes. The user is solely responsible for knowing and adhering to any and all applicable laws, rules, and regulations relating or pertaining to use of the Collection.

All content in this Collection is owned by and subject to the exclusive control of Regis University and the authors of the materials. It is available only for research purposes and may not be used in violation of copyright laws or for unlawful purposes. The materials may not be downloaded in whole or in part without permission of the copyright holder or as otherwise authorized in the "fair use" standards of the U.S. copyright laws and regulations.

ASSESSING THE FLEXIBILITY OF A SERVICE ORIENTED ARCHITECTURE TO THAT  
OF THE CLASSIC DATA WAREHOUSE

A THESIS

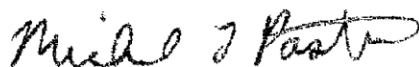
SUBMITTED ON THE 19<sup>th</sup> OF MAY, 2010

TO THE DEPARTMENT OF INFORMATION SYSTEMS  
OF THE SCHOOL OF COMPUTER & INFORMATION SCIENCES

OF REGIS UNIVERSITY

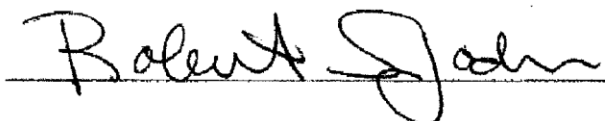
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS OF MASTER OF SCIENCE IN  
SOFTWARE AND INFORMATION SYSTEMS

BY

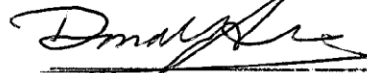


Michael Pastore

APPROVALS



Robert Sjodin, Thesis Advisor



Donald J. Ina, MCT626 Faculty



Ranked Faculty Name

### **Abstract**

The flexibility of a service oriented architecture (SOA) is compared to that of the classic data warehouse across three categories: (1) source system access, (2) integration and transformation, and (3) end user access. The findings suggest that an SOA allows better upgrade and migration flexibility if back-end systems expose their source data via adapters. However, the providers of such adapters must deal with the complexity of maintaining consistent interfaces. An SOA also appears to provide more flexibility at the integration tier due to its ability to merge batch with real-time source system data. This has the potential to retain source system data semantics (e.g., code translations and business rules) without having to reproduce such logic in a transformation tier. Additionally, the tight coupling of operational metadata and source system data within XML in an SOA allows more flexibility in downstream analysis and auditing of output. SOA does lag behind the classic data warehouse at the end user level, mainly due to the latter's use of mature SQL and relational database technology. Users of all technical levels can easily work with these technologies in the classic data warehouse environment to query data in a number of ways. The SOA end user likely requires developer support for such activities.

### **Acknowledgements**

I'd like to first of all thank my wife Ramona and children Sophia, Michael, and Andrew for giving me the impetus to continually improve myself both personally and professionally. I love you all. Secondly, I'd like to thank my thesis advisor, Professor Rob Sjodin, for all your guidance these past 15 months or so. Your thorough edits and frank assessments kept me on track with my research. Finally, I'd like to thank all the staff and students at Regis University and the National University of Ireland, Galway with whom I interacted over the past few years. Your professionalism and support were instrumental in making my learning experience both challenging and rewarding.

## Table of Contents

Abstract .....	ii
Acknowledgements .....	iii
Table of Contents .....	iv
List of Figures .....	vi
List of Tables.....	vii
Executive Summary .....	1
Chapter 1 – Introduction .....	3
1.1 Problem Statement .....	3
1.2 Project Proposal.....	4
1.3 Significance .....	5
1.4 Thesis Statement .....	6
Chapter 2 – Review of Literature and Research .....	7
2.1 Overview of the Secondary Research .....	7
2.2 Classic Data Warehousing vs. SOA .....	7
2.3 How do the Architectures Bridge the Business/Technical Divide? .....	10
2.4 How Accessible are the Architectures?.....	13
2.5 How do the Architectures Handle Data Volatility? .....	17
Chapter 3 – Methodology.....	25
3.1 Approach .....	25
3.2 Project Plan .....	25
3.3 Flexibility Benchmarks .....	26
3.4 Prevention of Bias .....	27
Chapter 4 – Project Analysis and Results .....	28
4.1 Summary of Findings.....	28
4.2 WCF Proof of Concept.....	28
4.3 Source System Adapters.....	36
4.4 Integration Adapter .....	47
4.5 End-user Access .....	53
Chapter 5 – Project History.....	58
5.1 Project Origins.....	58
5.2 Scope .....	58
5.3 Project Management.....	58
5.4 Milestones .....	59
5.5 Changes to the Plan .....	59
5.6 Evaluation.....	59
5.7 Summary .....	60
Chapter 6 – Conclusions .....	61
6.1 Statement of Findings.....	61
6.2 Major Themes Uncovered.....	61
6.3 Research Limitations .....	62
6.4 The Project in Hindsight .....	62
6.5 Research Opportunities .....	62

References ..... 64  
Appendix A – Revision History ..... 67  
Annotated Bibliography ..... 68

**List of Figures**

Figure 1: WCF Test Client Interface.....	30
Figure 2: WCF Request/Response Test .....	31
Figure 3: SOAP request .....	32
Figure 4: SOAP response .....	32
Figure 5: The IIS Host Environment.....	33
Figure 6: Access to WSDL Metadata.....	33
Figure 7: Adding a WCF Service Reference.....	34
Figure 8: The XML Method Interface.....	34
Figure 9: Consumer Access to the WCF Remote Service .....	35
Figure 10: The Investment Data Warehouse Environment.....	36
Figure 11: Security Issuer/Issue Relations .....	39
Figure 12: SMF Adapter and Consumer Interface .....	40
Figure 13: The Underlying SOAP Message Packets. ....	41
Figure 14: The WCFAccounting Class Diagram .....	42
Figure 15: The PricingVendor01 Class Diagram.....	44
Figure 16: Proprietary HTTP Price Request Post .....	44
Figure 17: The Object-Based Approach.....	45
Figure 18: The Transformation Activity Diagram .....	49
Figure 19: The Transaction Cube Adapter Class .....	50
Figure 20: Kimball's Consumer Mode Stratification.....	54
Figure 21: The Data Warehouse and SOA Tiers.....	55
Figure 22: User Integration of SOA Services .....	57



**List of Tables**

Table 1: Summary of Findings..... 28  
Table 2: Source System Extraction Flexibility Criteria ..... 37  
Table 3: Source System Extraction Flexibility Synopsis ..... 46  
Table 4: Data Transformation/Integration Flexibility Criteria..... 47  
Table 5: Data Transformation/Integration Flexibility Synopsis ..... 53  
Table 6: Thesis Revision History ..... 67

## **Executive Summary**

The classic data warehouse model has remained largely unchanged over the past 20 years. A data warehouse typically resides outside of the transactional system environment. Data are periodically fed from source systems to the warehouse in batches, where transformation, tagging, and integration processes are performed. Business users utilize the data warehouse database for a variety of tasks, including ad hoc querying, data analysis, and standard reporting. A key advantage of this architecture is that the complexities of back-end systems have all been done away with; users have access to everything on a homogeneous platform.

The evolution and adoption of Web services over the past few years, enabled in part by standards such as SOAP and XML, have greatly enhanced the ability of disparate information systems to communicate with one another using common formats. An architectural concept called service oriented architecture (SOA) has subsequently evolved that allows organizations to utilize such standards to expose their systems to the outside world in a uniform fashion. The flexibility of an SOA is apparent in business-to-business (B2B) communications because organizations no longer have to use proprietary technologies in order to communicate with one another.

Research was conducted in order to assess whether the flexibility of an SOA is applicable to the classic data warehouse. It was found that an SOA provides more flexibility for source system data extractions than the classic warehouse, at the cost of complexity on part of the source system data providers themselves. An SOA also enhances the capability of combining online and batched source system data, allowing end-users more flexibility in combining real-time queries with historical data. Additionally, the non-rigid nature of the XML storage format

allows a more cohesive coupling of metadata with source system output, thereby enhancing flexible analysis of integration and transformation processes.

An SOA was found to lack the front-end flexibility of the tools found in the classic data warehouse environment. Specifically, users in the classic environment have access to summarized data tables and easy-to-use query tools based on SQL. Users of an SOA are more likely to require developer support in order to access back-end data. This likely precludes an SOA from being used as a front-end architecture for ad hoc purposes, thereby relegating it to the back-end processing tiers for the foreseeable future.

## **Chapter 1 – Introduction**

### **1.1 Problem Statement**

This research project was initiated due to the need of a small fixed-income investment firm to phase out and replace the majority of its data warehouse code over the course of the next four years. The current codebase is written in Microsoft Visual Foxpro (VFP), an Xbase language that was widely used in the 1980's and 1990's. The vendor will discontinue extended support for the language in April, 2014.

The initial analysis of a replacement language soon led to fundamental questions about the data warehouse architecture itself and its ability to handle an increasingly complex number of application providers and data integration needs. The current investment warehouse is based on the classic warehouse paradigm, whereby source system data structures are periodically imported, transformed and integrated within a homogeneous database platform. This methodology has remained virtually unchanged since the widespread adoption of the warehouse model in the 1980's.

A number of technological advances have emerged in the intervening years as a result of the phenomenal growth of the World Wide Web. Web Services standards such as XML, SOAP and WDSL, supported by the HTTP and TCP/IP transport protocols, allow for an unprecedented level of flexibility in how disparate systems can interact with one another. The concept of a service-oriented architecture (SOA) has also gained traction as a means of making such interactions easier to deploy and manage, independent of the configuration of the participating systems. Although the SOA is today primarily used for integrating enterprise applications, it shows potential for improving flexibility in the new investment data warehouse.

## 1.2 Project Proposal

The research project will consist of two major phases. The first phase will be to conduct an in-depth study of the SOA, its underlying technologies, standards, and current vendor implementations. The primary sources for the data collection will be academic journals and other published literature. Vendor-biased literature such as white papers will be avoided, as well as web sites that provide non-substantiated opinions. A review will also be conducted of current data warehouse methodologies and will utilize data collection methods similar to those used in the SOA research.

The second, and primary, phase of the research project will be to develop a number of SOA prototype adapters that provide services commonly found in the classic data warehouse environment. A project plan will be developed that outlines the implementation of these services. The three primary layers of the data warehouse environment (extraction, transformation/load, and reporting) will serve as the guideline for developing these adapters. The flexibility of the adapters will be compared to similar functionality found in the legacy environment.

The feasibility of the research project is high. The fixed-income firm has licensed copies of Windows 2003 servers, as well as an MSDN license that includes Microsoft Visual Studio 2008. Microsoft's Windows Communications Foundation (WCF) appears to support the major components associated with an SOA architecture and will likely be used to deploy the SOA adapters. Most importantly, the management team at the fixed-income firm supports the research project as a means of not only replacing the legacy codebase but also of potentially improving the flexibility of data reporting and analysis.

### 1.3 Significance

A number of important contributions to the information technology and business fields will result from this project. First, it will be determined if a service-oriented approach to data extraction provides better flexibility than that found in traditional warehouse implementations. Classic data warehouse extraction routines are typically tightly bound to the schemas of the systems with which they interact. The black-box, response-request paradigm of the SOA may help to abstract such interfaces so that back-end upgrades have a lower impact on downstream extractions.

Another contribution of this research project will be to explore whether an SOA approach to extraction-transaction-load (ETL) allows static and real-time interfaces to coexist with one another. The traditional data warehouse ETL process is primarily batch-oriented. Data are imported from source systems in timed intervals, transformed in some way, and then stored in the warehouse repository. This research project will assess how well SOA can manage ETL in a more fluid manner. This is an important consideration for organizations that need the flexibility to tap into source systems in non-uniform time intervals. Furthermore, the exposure of ETL processes as services likely allows for a better abstraction of source systems, thereby facilitating the integration of heterogeneous systems.

Finally, this research project will explore the notion that data warehouses must reside in a relational, homogeneous database environment that is accessed via structured query language. The request-response nature of the SOA might provide more flexibility for warehouse developers and end-users alike because of its higher emphasis on the functional, rather than non-functional, aspects of data retrieval and reporting.

### **1.4 Thesis Statement**

Utilizing a service-oriented architecture in the redesign of a legacy investment data warehouse produces a system architecture that is more flexible than that found in a traditional implementation.

## **Chapter 2 – Review of Literature and Research**

### **2.1 Overview of the Secondary Research**

A thematic literature approach was conducted to identify and compare flexibility categories for both classic data warehousing and the SOA . This review not only helped the researcher to draw some initial conclusions but was also beneficial in understanding the technical aspects of the newer architecture.

### **2.2 Classic Data Warehousing vs. SOA**

#### *Classic Data Warehousing*

According to Inmon (2005, chap. 1), a properly implemented data warehouse solves a number of problems traditionally related to corporate data analysis and reporting. These include (1) data quality, (2) reporting productivity, and (3) transformation from data to information. The issue of data quality often arises when the source data from multiple systems need to be combined with one another. Unless these systems all share an integrated data dictionary, it is likely that their naming conventions and coding schemes for data elements will differ from one another. For example, one system may store a customer or supplier address in one character field while another system has normalized the distinct address elements into separate fields. Such differences are accounted for in the data warehouse through parsing and cleansing software.

Reporting productivity in a data warehouse environment is enhanced by virtue of having integrated, cleansed data in a single data repository. The onus is no longer on the user to



determine how to link data from multiple systems, or to access systems which reside on different platforms. Such functionality is taken care of by means of the extraction, transformation, and load (ETL) services in the warehouse environment. User productivity is also enhanced by means of having all integrated data in a homogeneous repository. This repository typically consists of one or more relational or star schema structures and is accessible via SQL. Ample query tools exist with which even the most novice end-user can quickly obtain information from such data stores. Kimball, et al. (2008) notes that the majority of users will end up running pre-written, parameterized reports that exist as part of a business intelligence repository (chap 11).

The transformation of data to information occurs when extracts from multiple systems are integrated with one another and uniform code semantics are applied to the output. It is also likely that subsequent data loads into the warehouse will form the basis of a historical, time-variant view of source system data. This is one of the key aspects of a data warehouse which differentiates it from a real-time system. While the latter is primarily concerned with providing transactional services in a low latency environment, the data warehouse system exists to provide ad-hoc analytical or standard reporting services to the business analyst. It should be noted that more modern information systems, especially ERP's, have come a long way in allowing ample historical data to co-reside in the production system environment. However, as Inmon (2005) notes, there exists no consistency in how much historical data each system maintains (chap. 1). The data warehouse can fill such a void by applying normalized time stamping as source data are imported. This provides a rich historical repository over time.

*Service Oriented Architecture*

The efficient alignment of business and IT objectives has been a core challenge since the early days of computing. An SOA approach to managing application deployment is seen as a positive step towards better integrating business processes with their technological implementation. As Kamoun (2007, p. 1) notes, the decades-old concept of business process management may now have an enabling technical counterpart. A key benefit of SOA can therefore be seen in its ability to deliver application services in a diverse business environment. As Dan, Johnson, and Carrato (2008) note, this provides many benefits to an organization, including (1) agile solution deployment, (2) reductions in cost through the avoidance of duplication of effort, and (3) reduced risk because of the reuse of well-tested code and run-time environments (p. 25). Deployment agility is gained through the ability of an SOA to provide access to multiple system software components at runtime in the form of middleware services. The loosely coupled nature of these services allows them to be more easily orchestrated in ways that are customized to meet the needs of the business process chain. (Schepers, Iacob, and Eck, 2008, p. 1055). Developers of such solutions can choose what services they need and then combine these into custom applications. The input parameters and output specifications for the services are available via a service registry.

The potential for cost reduction through an SOA is realized by the consolidation of service interfaces into a uniform repository. Existing business semantics can be discovered and reused in a consistent manner throughout the organization. The alternative to this is to duplicate business logic across applications, which will likely prove costly and difficult to manage. Marks & Bell (2006) claim that IT costs can in fact be reduced up to 80% through the application of an SOA, not only by reducing duplication of coding effort but also in the savings realized through reduced hardware, licensing, and integration fees (chap. 9). Reduced risk through an SOA can

be seen as a byproduct of utilizing source systems which are part of stable, well supported production environments. It is important to note that SOA as such is not so much about code development as it is about leveraging existing code bases. Legacy systems are natural candidates for integration into an SOA, owing to the key role they play in the delivery of core business functionality. Early SOA implementations have in fact been primarily oriented towards such systems. (Dan, Johnson, and Carrato, 2008, p. 25).

### **2.3 How do the Architectures Bridge the Business/Technical Divide?**

An important aspect of a flexible system architecture lies in its ability to convey business semantics to end users. This section will explore the ways in which data warehousing and the SOA bridge the technical-to-business divide..

#### *Classic Data Warehousing*

The integrated data warehouse repository provides the primary link between business users and the information contained in on-line applications. Fundamental to this concept is that the data warehouse is separated from the source systems from which it draws its data. Inmon (2005) contends that this architecture provides a number of advantages for the business user, including (1) an integrated and reconciled analytic environment, (2) a subject, rather than functional, view of data, and (3) a level of granularity sufficient to conduct a wide range of decision support activities (chap. 2). The data warehouse ETL process provides the functionality required to integrate data from multiple source systems into the homogeneous warehouse environment. A key aspect of the ETL phases is to provide the data translation and reconciliation logic necessary for maintaining and expanding the semantics of the source system

repositories. Kimball & Caserta (2004) refer to this as the “cleaning and conforming” phase of data transformation (chap. 4).

The conforming of incoming data is perhaps the most challenging part of the ETL process because it requires that output structures are modeled in a way that will meet diverse end-user needs. These structures, whether OLAP-oriented star schemas or a quasi relational operational data store (ODS), make available the subject, rather than functional, semantics of source systems to business users. The warehouse is not an application-oriented environment; its purpose is primarily to expose the underlying discrete data elements which cross-cut source production systems. For example, the insurance business user of a data warehouse is more likely to think in terms of customer and claims rather than auto and property (Inmon, 2005, chap. 2).

The level of granularity of the warehouse determines how flexible it will be for its user base. In general, a finer grained repository will provide more reporting opportunities to the business user, albeit at the cost of increased structural complexity. As mentioned previously, much of this complexity can be mitigated by implementing a business intelligence portal containing prewritten reports. Silvers (2008) contends that the use of static metadata is integral to this process because it provides users insight into the meaning and origin of data (chap 9). The metadata repository might be something as simple as a data dictionary stored in an Excel spreadsheet or part of a comprehensive system in which run-time ETL processes, user access rights, and context-based reporting are managed.

### *Service Oriented Architecture*

An SOA usually consists of decoupled middleware services that are bound to core production systems. These services make key components of the application environment

directly available to business processes and end-user applications. An SOA can be seen as providing the means for business users to easily interact with back-end systems. The benefits of such an approach are realized when new business processes need to be developed from legacy software systems. Braun (2007) contends that an SOA-based enterprise model can link process, software, and application layers within the overall computing architecture, thereby providing flexible support for business processes (p. 1218).

An SOA is more application/process than subject/data oriented. Within the context of decision support, such functionality allows the business user to obtain source system data in which semantics already conform to established business rules. The need for an intermediary processes such as an ETL sub-system might therefore be regarded as superfluous in some cases. A particular advantage to the SOA approach is that changes to underlying source system structures or platforms are abstracted via stable interfaces within the SOA services layer. A user of the SOA can then interact with services in a consistent manner, free from the burden of knowing the underlying system's schemas.

The key aspect of such functionality lies in the contractual nature of the SOA runtime environment. The SOA service catalogue is a formal contract that specifies what a service provides. It lets a consumer know what they can do and the constraints that govern how the service is delivered. The consumer can review the contract to determine both the functional and non-functional properties of the service. Service contracts typically fall under an overall SOA governance structure within the organization and are deployed via Web services. Similar to the data warehouse ETL processes, SOA service contracts are driven by metadata. Efforts over the past few years have brought forth SOA-based metadata standards such as SOAP and Web Services Description Language (WSDL). Organizations who publish their services using such

standards may be better positioned to integrate their business processes with other service providers, without incurring significant integration costs.

#### **2.4 How Accessible are the Architectures?**

The degree of flexibility in accessing data within the data warehouse and SOA architectures can be assessed by evaluating (1) how open the architectures are to users of various technical skill levels and job functions, (2) what language constructs are available for interacting with the architectures, and (3) how well current technologies and standards promote flexible data access within the architectures.

##### *Classic Data Warehousing*

The data warehousing environment is meant to be accessed directly by end-users in a number of ways, including query tools, canned reports, and business intelligence applications. The end-users themselves are apt to be more business than technically oriented. Both Inmon (2005) and Kimball (2008) identify a few classes of end users, including business specialists and operational personnel. The former are more likely to interact with the warehouse in an ad hoc manner, while the latter usually run predefined reports from the system. A few factors enable business users flexible access to the data warehouse environment. First, the data warehouse environment of today is almost exclusively hosted on some form of relational database management system (RDBMS). The primary access mechanism for the data stored therein is SQL. Codd (1970) correctly predicted that a “n-ary “ relational data store, in conjunction with a high level query language, would someday provide business users with direct access to corporate data stores.

Another reason warehouse data are readily available to most users is that software tools, SQL language constructs, and networking standards have matured over the past two decades. High level software tools such as Microsoft Access and Excel have built in query-by-example (QBE) interfaces that allow even novice users to easily join tables and apply filtering logic. The SQL language itself has undergone ANSI standardization, and is supported by most vendors. Additionally, high-level data access interfaces such as ODBC, JDBC, ADO, and OLEDB provide an easy and flexible means for users to connect to back-end databases. Finally, the emergence of TCP/IP as the predominant industry-wide networking protocol provides a uniform communication medium between computers on both local and distributed networks.

Although the data warehouse has benefited by the standardization of its underlying technologies, no overarching warehouse standard per se has ever gained traction. The W3C Common Warehouse Metamodel (CWM) was developed by a consortium of users in an attempt to provide a standard through which data can be traced and shared within the warehouse environment. As Hartman (2008) notes however, the standard's complexity is a major factor in its not gaining wide-spread adoption industry-wide (p. 52). Inmon and Kimball themselves disagree upon what logical structure the data warehouse should be, with the former arguing for a normalized data store and the latter for an OLAP-friendly star schema (Drewek, 2005). Architectural disparities such as these may challenge the user who is well versed in only one of the data warehouse models if they attempt to migrate to another one.

Also noteworthy is that the major RDBMS' have proprietary language constructs in their SQL offerings. This is due in part to vendors having already implemented portions of SQL prior to the formal standards being agreed upon. Database vendors also tend to introduce language enhancements over time in order to increase developer productivity and make their product

offerings more marketable. A data warehouse user in the Oracle environment is therefore likely to generate SQL code that cannot port to the MS SQL Server environment. Arvin (2009) provides examples of the disparities between commercial vendor's SQL dialects and the ANSI standard itself. Such differences will challenge a developer who is attempting to write portable data warehouse code.

### *Service Oriented Architecture*

In contrast to the classic data warehouse, SOA has from the beginning been primarily about enabling communications between backend systems. The technologies of the Web (e.g., HTTP, TCP/IP, SOAP, and XML) allow for disparate systems to interact with one another in a loosely coupled manner across a wide area network. Access to the SOA environment however is still largely the domain of the software developers; business user access is mainly accomplished through software applications which hide the underlying complexities of backend data fetches.

Limited user access can be attributed in part to the relative newness of the SOA and its only gradual adoption by major software vendors. There are also indications that the sheer complexity of the architecture can pose significant challenges in training both technical and business users in how to utilize it. For instance, Im, Guimaraes, and Hoganson (2004) found that although the individual components of middleware technology may be easy to grasp, assembling them all to create a N-tier solution can be a large task. Such complexities, at least in the short run, will likely keep the SOA out of the domain of the casual business user. Lopez, Casallas, and Villalobos (2007) also express concern that the SOA may not be adequately covered in the IT curriculum at many universities. This may put recent IT graduates at an initial disadvantage, and slow the adoption of SOA by technically-oriented users.



Access to the SOA environment is accomplished using a “request-response” paradigm. A client application or component, acting as a consumer, sends a request to a middleware service which in turn sends back a response. This approach is analogous to distributed component models such as DCOM and CORBA. These older technologies however tend to be harder to configure due to the tighter coupling of their runtime environments. Scribner and Stiver (2000) contend that the “heaviness” of such architectures cause them to be expensive to set up and maintain (chap. 1). An SOA in contrast can capitalize on the text-based, open standards afforded by Web-based technologies such as HTTP, XML, and SOAP. These light-weight components provide a higher degree of flexibility in both deploying and accessing the SOA across a wide range of platforms.

The request-response mode of communication of an SOA is similar in some respects to the object-based programming model seen in most modern 4GL's. A client application is apt to make a number of object-based calls via the SOAP message protocol to middleware services. The corresponding SOAP responses are then assembled by the client in order to create the desired output. An object-based approach to programming can have some advantages over declarative languages such as SQL. For instance, the user does not have to be concerned with the underlying logical schema of the system with which they are interacting. The SQL user on the other hand needs to express relations; a change in the schema requires a change in client code. The nature of object-based calls also lend themselves to a more natural business-like process description. This has the potential to allow more business-oriented users to assemble applications using SOA services.

The W3C standards that support SOA strongly favor its flexible use across platforms. As mentioned previously, Web-oriented protocols such as HTTP and TCP/IP, as well as standards

such as XML and SOAP, promote interoperability amongst systems. The XML format, like HTML, is a simplified subset of the SGML standard. An inherent strength of XML lies in its structural flexibility. This allows for the formatting of data in a large number of ways, such as hierarchical tree structures or normalized relational formats similar to those found in RDBM's. In contrast to a SQL based repository, which returns a cursor of rows and columns, a service utilizing XML as its transport format can return data in a number of formats, which Sperberg-McQueen (2005) aptly terms "semi-structured data". The W3C XQuery standard for performing SQL-like operations on XML files provides users with a more abstract way of interacting with such documents. As Seeley (2007) notes, this will help speed development and simplify access to XML files. Prior to this developers had to rely on proprietary parsing mechanisms, or the less flexible XSLT standard.

## **2.5 How do the Architectures Handle Data Volatility?**

Decision support architectures need to have mechanisms in place so that the user can understand the meaning and context of the information contained therein. The resiliency of these systems can be challenged when changes introduce semantic data conflicts in the reporting environments. Common scenarios include (1) changes to source data schemas, (2) ambiguities between source data values and definitions, and (3) changes to the reporting data schemas/services themselves.

### *Classic Data Warehousing*

#### *Changes to Source System Schemas*

Changes to source data schemas will impact the data warehouse ETL processes that directly access them. For instance, the renaming of a table or field that is accessed by a data

import process will cause an error to occur during runtime processing. One workaround is to have source systems produce an intermediary file (typically in delimited text format) that represents a de-normalized, schema-neutral representation of the source data. The data warehouse ETL process will then import and parse the flat file, without having to deal with the relationships within the backend schema from which it came. Additionally, the use of text as a transport medium allows the ETL processes more leeway in the interpretation of data types. For example, a source system might change a salary field from data type float to double, or change the decimal precision from two to four digits. The representation of these values in the intermediary file will be type neutral, allowing for greater flexibility in how they are handled downstream. Kimball and Caserta (2004) list other advantages to using flat files, such as the ability to use FTP as a transport medium and faster bulk load times (chap. 3).

There are cases however in which source systems are incapable of producing an extract file for the data warehouse. In these situations a data warehouse programmer must either make direct changes to code within the ETL processes or implement a metadata tool that provides schema mapping services at compile or runtime. The more traditional approach to ETL within the warehouse has been to maintain schema awareness within the code itself. A programmer utilizes a 3GL or 4GL language to hard code all extract routines against a known schema. The growth of relational databases in the late 1980's, along with the use of embedded SQL in source code, has greatly simplified this process. Although the use of schema-aware source code is likely to provide the fastest runtime performance, it is likely to be less resilient to change than processes that link to metadata mapping files.

A metadata mapping repository can be used to store the physical and logical aspects of the source data repositories feeding the data warehouse system. The advantage of this approach

is that a transparent, language independent view of all feeder systems is maintained outside of a static code base. The ETL programmer can then write processes which link to the metadata at run-time to build dynamic SQL statements. Changes to backend schemas then require (in theory at least) a change to the metadata repository only, not the code. A more detailed metadata file will likely contain actual fields which are joined to create output. A variety of vendor metadata-based tools exist in the marketplace for these purposes. The more sophisticated of these allow for rules-based metadata that are compiled into languages such as Java, C, or Visual Basic. In this scenario, a change to a backend schema will precipitate a change to the metadata repository, and subsequent recompiling of code. The compiled code is likely to provide better runtime performance than the on-the-fly dynamic SQL approach, albeit with potentially less flexibility on the part of programmer.

#### *Source Data Ambiguities*

The reconciliation of source system data conflicts during the ETL process is perhaps the most challenging part of providing quality information in the classic data warehouse. This issue is apt to be compounded as more source feeder systems are added, particularly if these systems intersect one another in terms of business functionality. For instance, an auto parts retailer may utilize a number of different wholesalers for their supplies. It is likely that the wholesalers will have different coding schemes for the same products. This will require the designers of the retailer's data warehouse to implement metadata crosswalk structures for reconciling coding inconsistencies.

A more vexing integration challenge can arise when data semantics between source systems differ from one another. For instance, a student records system at one university may

list a different date of birth for a student as another university. One school might also define full time and part times students differently, thereby making it difficult to define these terms when the student records from each source are integrated in the data warehouse. Bleiholder and Naumann (2008) specify the three categories of data semantic conflicts as schematic, identity, and data (p. 7).

Schematic conflicts can likely be resolved by maintaining a metadata crosswalk or mapping table. Identity conflicts are likely to require more work, because each system is providing its own version of the truth for the same element of information. Moreover, one of the systems might be the more authoritative source depending on the context of the report being run in the data warehouse. In such cases, metadata mapping back to discrete source system data elements is likely necessary. Users of the warehouse can then choose which source value to use based on their specific needs. Fritz (2006) refers to this technique as atomic data source mapping, and contends that it should be done during the design of the semantic data model.

Data conflicts occur when semantic differences cannot be resolved based on context. Although outright data errors will likely have to eventually be resolved (e.g. in the case of conflicting student ages or gender classifications), there are instances in which the original values must both be maintained in the warehouse due to ambiguity about which value is correct. This situation might occur when two different bond rating agencies differ in their credit assessment of a financial security. In this case, both ratings are represented in the data warehouse repository. The cube structures found in the OLAP-type warehouse provide an efficient means of storing data in this manner. As Bleiholder and Naumann (2008) note, SQL unions can be used to load records from the different systems into the cube, thereby retaining source system semantics (p. 20).

*Changes to the Data Warehouse Repository*

The data warehouse, like the source systems which feed it, is apt to undergo upgrades during its lifetime. Changes relating to new data structures, whether fact tables or cubes, are to be expected as new reporting requirements surface and additional source systems are integrated into the warehouse repository. Existing interfaces to the data warehouse structures are unlikely to be impacted by such additive changes. However, changes to the legacy schemas within the warehouse itself can cause problems.

Such reorganizations have the potential to break BI applications and user queries that have been developed using the older schema. This exposes a fundamental weakness of the data warehouse; like the operational systems that feed it, it exposes its underlying logical schema to end users. Changes to this schema require that an impact analysis be done in order to assess downstream risk. This can be a rather large task, depending on the scope of the changes and the number of existing interfaces to the legacy schema. Rainardi (2008) suggests treating such changes as one would an upgrade request for a production system, utilizing formal processes and conducting an impact analysis (p. 501). Another approach for handling such changes is to maintain the legacy schema in parallel with the new schema, thereby providing end-users ample time to recode their data access routines. The legacy schema is then taken off line after a sufficient grace period has expired.

*Service Oriented Architecture**Changes to Source System Schemas*

An SOA that utilizes a contractual mechanism such as SOAP is apt to be less vulnerable to schema changes in backend source systems. This is because the request/response paradigm of the SOA deals with discrete data element fetches that are independent of the logical schema from which they originate. The XML schema within the SOAP message may however contain specific field data type definitions. An outright change of a source system field data type can therefore cause the same problems as in the traditional relational model.

### *Source Data Ambiguities*

Resolving provider data ambiguities within the SOA is vital if the architecture is to be used for decision support purposes. As with classic data warehousing, the data that feeds the SOA will be drawn from a variety of sources. The SOA however will likely not follow the traditional batch-oriented paradigm when pulling this data, opting instead for real-time access to on-line systems. This scenario runs contrary to traditional assumptions that data cleansing will be done in a homogeneous data warehouse environment (Agosta, 2006). The SOA is therefore faced not only with resolving data ambiguities (schematic, identity, and data conflicts) common to the classic decision support environment, but also with orchestrating ETL services in a dynamic systems environment. As noted by Dreibelbis, et al., (2008), this paradigm shift can lead to poor data quality because the SOA by nature has always been more about data mapping than data reconciliation (chap. 2).

The concept of master data management (MDM) has arisen over the past few years as a means to address data quality issues. The MDM paradigm goes beyond implementing a static metadata repository for data cross-walking data and reconciliation. It is a service unto itself, fully integrated with other services, both real-time and static, which co-exist with it in the SOA.

This service layer wraps the functionality of various systems (including legacy data warehouses) into a logically homogeneous, metadata driven view of the enterprise. The advantage of this approach is that data semantics do not have to be duplicated between source systems and the ETL processing environment. A change to operational metadata is automatically utilized by decision support metadata.

While the MDM does appear to address how an SOA can implement ETL functionality in support of decision support services, the question remains as to how source system data ambiguities are handled. Berson & Dubov (2007, chap. 15), use customer data integration (CDI) as the backdrop for explaining how a MDM-enabled SOA can resolve such issues. They explain that a customer entity can have a variety of meanings, depending on the context in which they are interacting with a particular service within an SOA. It is expected that a customer's attributes will mean different things at different times under different conditions. A so-called identity hub is utilized to handle such nuances. The hub provides mediation services for data synchronization activities between all components of the SOA.

#### *Changes to the SOA Repository*

Changes that might occur within an SOA repository include deprecation of interfaces, changes to existing interfaces, and URL changes. The deprecation of interfaces occurs when new business functionality is applied that either supersedes or supplements old functionality. For example, a financial analysis might need to assess an investment portfolio's performance over a twelve month period. Later on it may be necessary to provide both fiscal and calendar year performance returns. Users who elect to use the deprecated interface will still get the calendar year performance by default. Method parameter overloading within the service will likely be



implemented for such scenarios. In order to provide explicit support for both calls the response SOAP message should contain metadata that explains the context of the returned data.

Outright changes to interfaces within the SOA are somewhat akin to the schema changes discussed previously in the classic data warehouse section. The addition of new interfaces, like the addition of new tables or fields, is likely to have little or no impact on BI interfaces to the SOA. However actual changes to known interfaces will require the same level of change control analysis that one would undertake with a schema change in the warehouse. This scenario is not unlikely, as an interface library is apt to become bloated over time, requiring refactoring of deprecated functionality. This is normally done in order to avoid confusion and potential misinterpretation of data. URL changes will likely occur if services are moved to another provider. This might happen after a merger or acquisition, and should be relatively easy to manage via metadata at both the requester and service provider levels.

## **Chapter 3 – Methodology**

### **3.1 Approach**

A design science methodology has been chosen as the approach for conducting primary research. The research project will consist of the design and development of SOA software adapters for the extraction and transformation/integration tiers of the data warehouse ETL environment. The flexibility of the SOA adapters will be compared to the legacy software modules used in the current data warehouse. Although the assessment of flexibility will for the most part be subjective, based on observations made by the researcher, benchmarks will be established in order to present findings in a consistent manner. Section 3.3 provides more details on what benchmarks will likely apply to each tier of the development project.

### **3.2 Project Plan**

The project will be undertaken in a series of phases. These are:

1. Planning.
  - a. Estimating project scope.
  - b. Assessing feasibility.
  - c. Assessing resource needs.
  - d. Assessing time and cost.
2. Analysis.
  - a. Define the data warehouse tiers.
  - b. Establish benchmarks for each tier.
  - c. Research the SOA and determine a development approach.

3. Design
  - a. Design the SOA tiers.
  - b. Conduct a proof-of-concept prototype.
  - c. Assess approach and make modifications.
4. Coding
  - a. Develop the extraction tier.
  - b. Develop the transformation/load tier.
  - c. Assess the consumer business intelligence tier.
5. Analysis of Results
  - a. Assess SOA functionality compared to classic warehouse code base.
  - b. Tabulate and record results.
  - c. Conduct further testing as necessary.

### **3.3 Flexibility Benchmarks**

The flexibility benchmarks will be driven by the requirements within each tier of the data warehouse environment.

1. Extraction Tier:
  - a. The ability to interact with heterogeneous back-end systems.
  - b. The ability to maintain backend schema abstraction.
  - c. The ability to handle data volatility from back-end systems.
  - d. The ability to extend the backend systems .
2. Integration Tier:
  - a. The ability to integrate disparate data sets.
  - b. The ability to reconcile source ambiguities.

- c. The ability to summarize and tag incoming data.
3. Consumer Tier
    - a. A comparison of end-user access to the classic and SOA architectures.

### **3.4 Prevention of Bias**

The foregoing benchmarks will require that a certain degree of subjectivity be used when assessing the flexibility of each architecture. It is incumbent on the researcher to approach each assessment with no pre-conceived notions about what the outcomes should be. Although the results of the secondary research do show some advantages of each architecture over the other in certain tiers, these initial findings should not and will not drive the direction of the development project and subsequent analysis. Rather, the requirements of the warehouse itself, together with the predefined flexibility benchmarks, will guide the research effort.

## Chapter 4 – Project Analysis and Results

### 4.1 Summary of Findings

The findings presented in the following table indicates that an SOA provides diminishing flexibility in comparison to the classic data warehouse as one moves from the back-end tiers to the user interface tier. The maturity of the SOA over time, by means of increased vendor support and toolsets, will likely increase flexibility on the front end.

	Classic Data Warehousing	SOA
Source System Access Flexibility	It depends on vendor implementation. An SQL back-end schema provides access flexibility, at the cost of upgrade complexity. A proprietary vendor data access approach detracts from flexibility.	A uniform approach to data access allows focus to remain more on the “what” than “how”, thus promoting flexibility. The onus is on the vendor to maintain interface integrity however.
Integration Flexibility	The warehouse load process is batch-oriented, which typically precludes real-time integration of data. Transformation logic is usually done in a top-down fashion, which reduces transparency.	The use of adapters on the back-end promotes real-time, as well as batch, integration. The use of XML allows for tighter coupling of run-time metadata with output, which promotes transparency.
End User Access Flexibility	The maturity of RDBMS and SQL technologies promotes better ad hoc flexibility.	SOA is still nascent, developer support is needed. Interfaces can be cumbersome to work with.

Table 1: Summary of Findings

### 4.2 WCF Proof of Concept

A “proof of concept” SOA adapter was developed using Microsoft’s Windows Communication Foundation (WCF) architecture. The purpose of the development was to confirm that SOA adapters could be developed and utilized in a stable manner in the target environment. The adapter was successfully deployed and tested, thereby assuring that the primary research could be conducted using WCF as the SOA platform.

The pricing SOA adapter provides an abstraction layer for fetching security prices from an external pricing provider. The vendor currently provides two means for fetching prices from its systems: (1) via a Windows GUI and (2) via HTTP requests. The SOA adapter was designed as a means of wrapping this functionality within a uniform service interface.

The functionality inherent to price requests lends itself well to the request-response paradigm of the SOA. The basis components of a price request are:

1. The requester's credentials.
2. The security identifier to be priced, typically in CUSIP identifier format.
3. The effective date to be priced. This usually represents the end-of-day market close date.

As previously mentioned, users have two options for fetching prices from the vendor. The Windows GUI is used by business-oriented users to fetch prices. The HTTP method is a programmer-oriented tool that allows for more flexible automation of price fetches.

This development and deployment environment for the pricing adapter consisted of the following components:

1. Service and client UI development language: C#, under Visual Studio 2008.
2. Runtime environment: .NET Framework 3.5.
3. SOA architecture: Microsoft Windows Communications Foundation (WCF), utilizing HTTP, SOAP, and XML.
4. SOA service host environment: Windows 2003 running IIS 6.

The first step was to create a WCF service adapter to provide prices. The adapter uses the HTTP post approach, which requires a vendor-specific request string. The method `getPrice()` was

developed as a wrapper for such functionality. The .NET IDE allows for run-time testing of services. The following figure shows the test client interface used during debugging.

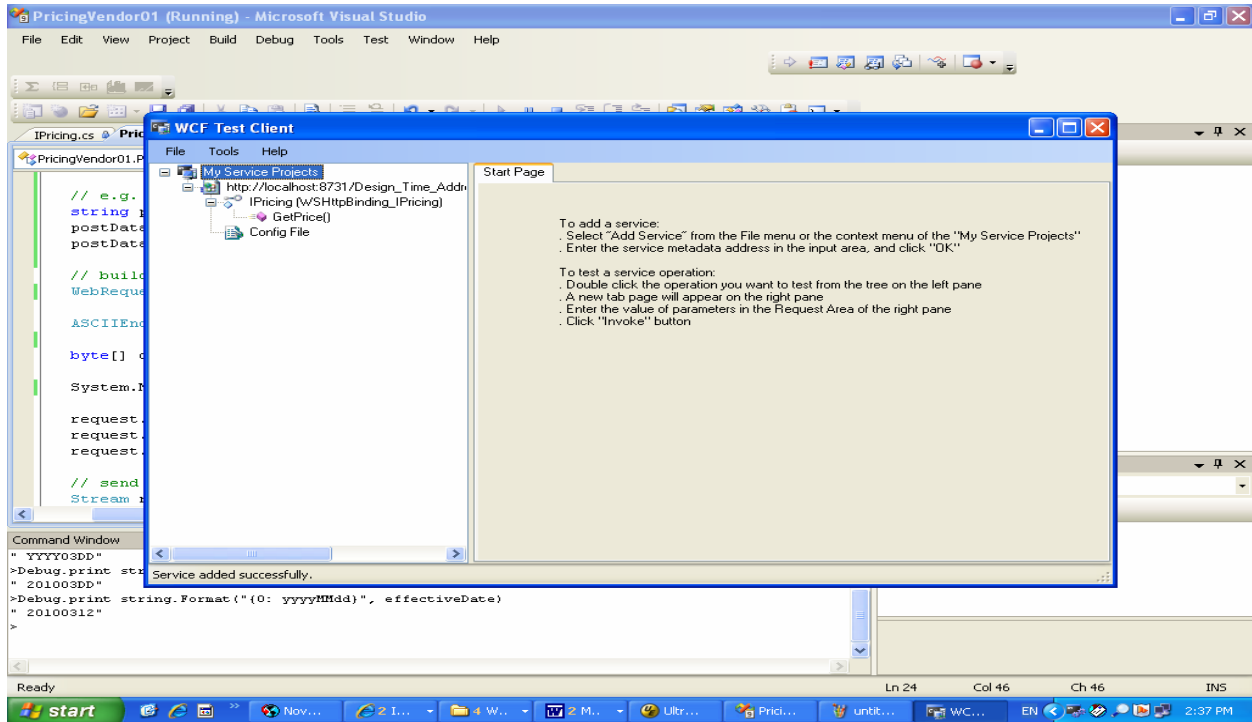


Figure 1: WCF Test Client Interface

The next figure shows the input parameters for the `getPrice()` method. The developer enters the parameters manually and then invokes the method. The response shows a returned price for the date of 2/12/2010. Note that sensitive information has been blotted out in green, as this service is accessing a production pricing data provider.

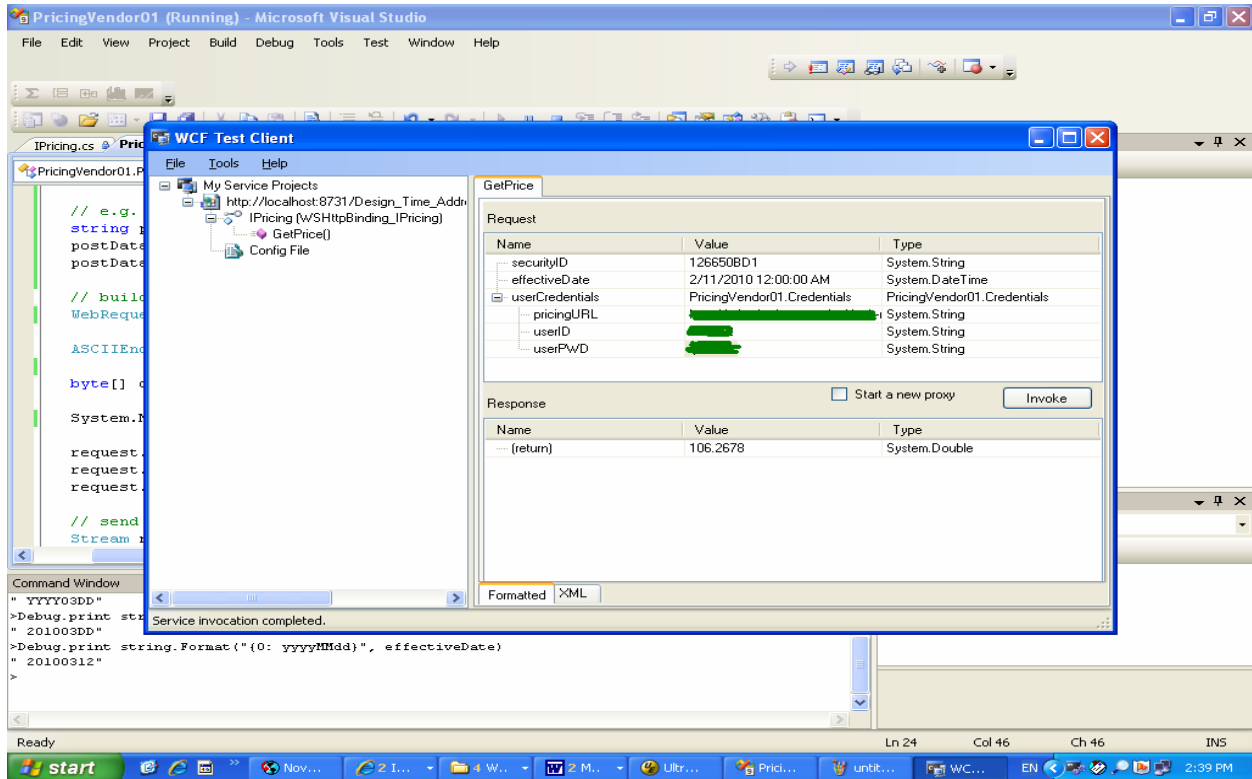


Figure 2: WCF Request/Response Test

The developer also has the ability to analyze the actual SOAP request/response envelope via the test client IDE. The next two figures show the request and response from the test conducted in the previous illustration. Again, sensitive data have been blotted out in green.

```
<s:Envelope xmlns:a="http://www.w3.org/2005/08/addressing"
xmlns:s="http://www.w3.org/2003/05/soap-envelope">
  <s:Header>
    <a:Action s:mustUnderstand="1">http://tempuri.org/IPricing/GetPrice</a:Action>
    <a:MessageID>um:uuid:78074690-42cb-4770-a5a2-01af51824955</a:MessageID>
    <a:ReplyTo>
      <a:Address>http://www.w3.org/2005/08/addressing/anonymous</a:Address>
    </a:ReplyTo>
  </s:Header>
  <s:Body>
    <GetPrice xmlns="http://tempuri.org">
      <securityID>126650BD1</securityID>
      <effectiveDate>2010-02-11T00:00:00</effectiveDate>
      <userCredentials xmlns:d4p1="http://schemas.datacontract.org/2004/07/PricingVendor01"
xmlns:i="http://www.w3.org/2001/XMLSchema-instance">
        <d4p1:pricingURL>[REDACTED]</d4p1:pricingURL>
        <d4p1:userID>[REDACTED]</d4p1:userID>
        <d4p1:userPWD>[REDACTED]</d4p1:userPWD>
      </userCredentials>
    </GetPrice>
  </s:Body>
</s:Envelope>
```



Figure 3: SOAP request

The response header section meta data has been omitted, as it is not relevant to this example.

```
<s:Envelope xmlns:s="http://www.w3.org/2003/05/soap-envelope"
xmlns:a="http://www.w3.org/2005/08/addressing" xmlns:u="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd">
  <s:Header>
    ... a lot of header metadata ...
  </s:Header>
  <s:Body u:Id="_0">
    <GetPriceResponse xmlns="http://tempuri.org/">
      <GetPriceResult>106.2678</GetPriceResult>
    </GetPriceResponse>
  </s:Body>
</s:Envelope>
```

Figure 4: SOAP response

After the client adapter was developed and tested it was deployed to a IIS host running on a Windows 2003 server. A new Web site called “PricingVendor01” was created to host the service:

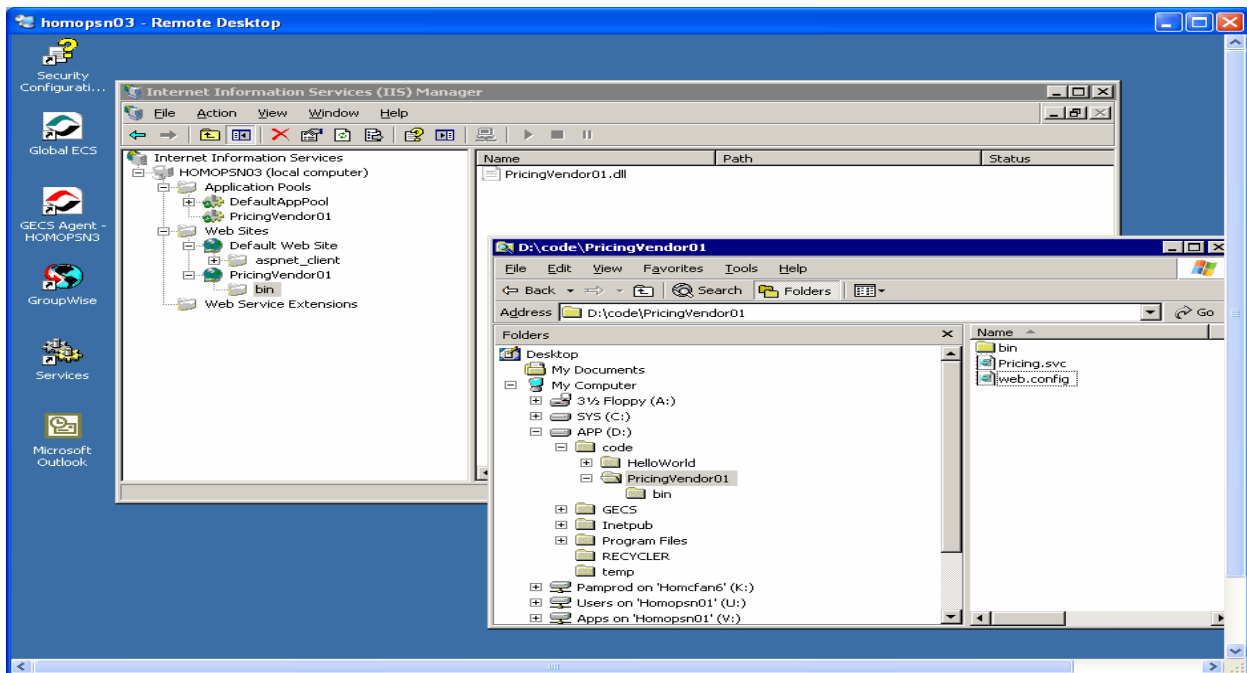


Figure 5: The IIS Host Environment

The web site publishes the service metadata via WSDL. This metadata can be queried by navigating to the .svc file that was generated by the client adapter project discussed earlier. This file will later be used by consumers of the service in order to create proxy stubs to it. The following figure shows a web page that can be used to gain access to the service's WSDL metadata.

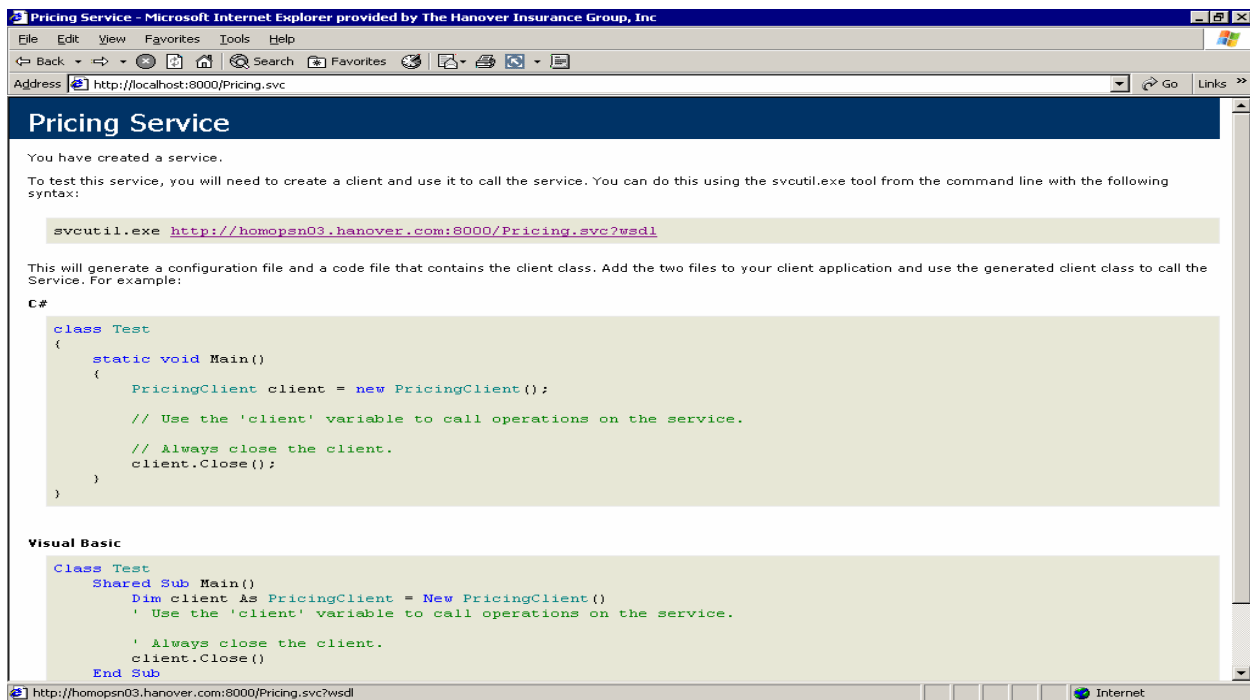


Figure 6: Access to WSDL Metadata

A client consumer application was developed in order to access and test the pricing adapter. A rudimentary C# Windows form application sufficed for this task. A proxy service stub was created by accessing the service's .svc file via the IIS URI. The next figure shows the add service dialogue. This functionality can also be accomplished via a app.config file at run-time, a more realistic approach in a production environment.

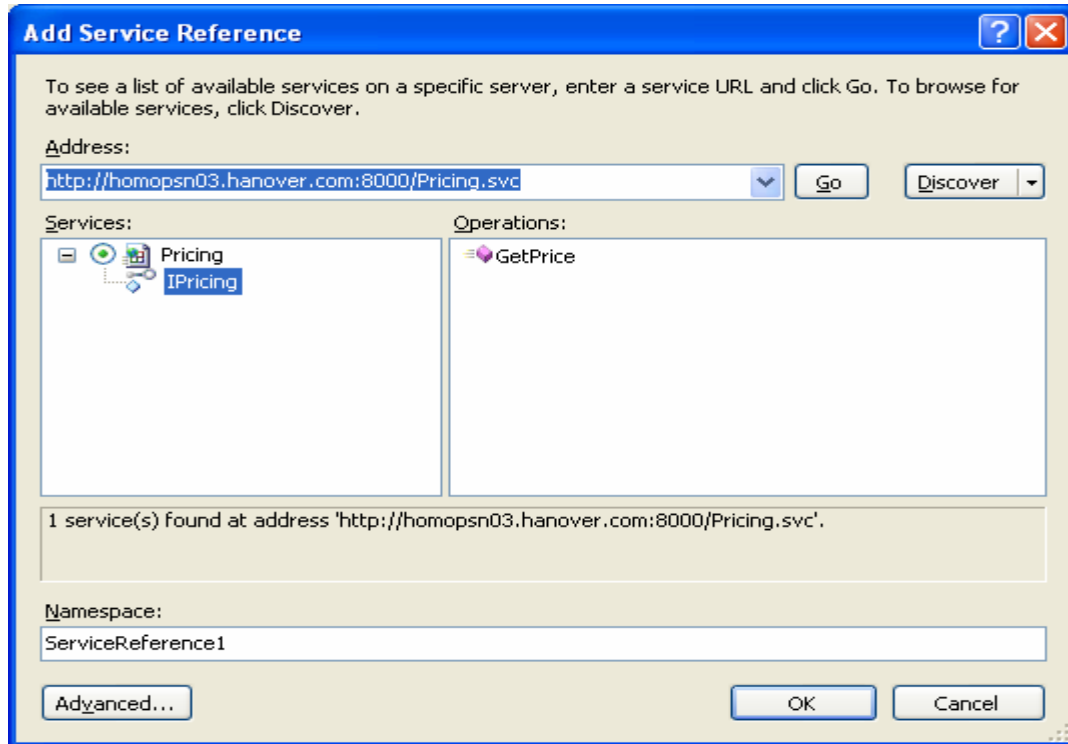


Figure 7: Adding a WCF Service Reference

The pricing UI now has access to the services provides by the pricing adapter on remote server. As expected, the adapter's interface metadata exists in XML format on the client UI side. The following figure shows the interface information for the GetPrice() method.

```

<?xml version="1.0" encoding="utf-8"?>
<xs:schema xmlns:tns="http://tempuri.org/" elementFormDefault="qualified" targetNamespace="http://tempuri.org"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:import schemaLocation="http://homopsn03.hanover.com:8000/Pricing.svc?xsd=xsd2"
namespace="http://schemas.datacontract.org/2004/07/PricingVendor01" />
  <xs:element name="GetPrice">
    <xs:complexType>
      <xs:sequence>
        <xs:element minOccurs="0" name="securityID" nillable="true" type="xs:string" />
        <xs:element minOccurs="0" name="effectiveDate" type="xs:dateTime" />
        <xs:element xmlns:q1="http://schemas.datacontract.org/2004/07/PricingVendor01" minOccurs="0" name="userCredentials"
nillable="true" type="q1:Credentials" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="GetPriceResponse">
    <xs:complexType>
      <xs:sequence>
        <xs:element minOccurs="0" name="GetPriceResult" type="xs:double" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>

```

Figure 8: The XML Method Interface

The client can now interact with the remote service in a COM-like manner. The next figure shows a price request being done via a Windows form. Sensitive data has been blotted out in green.

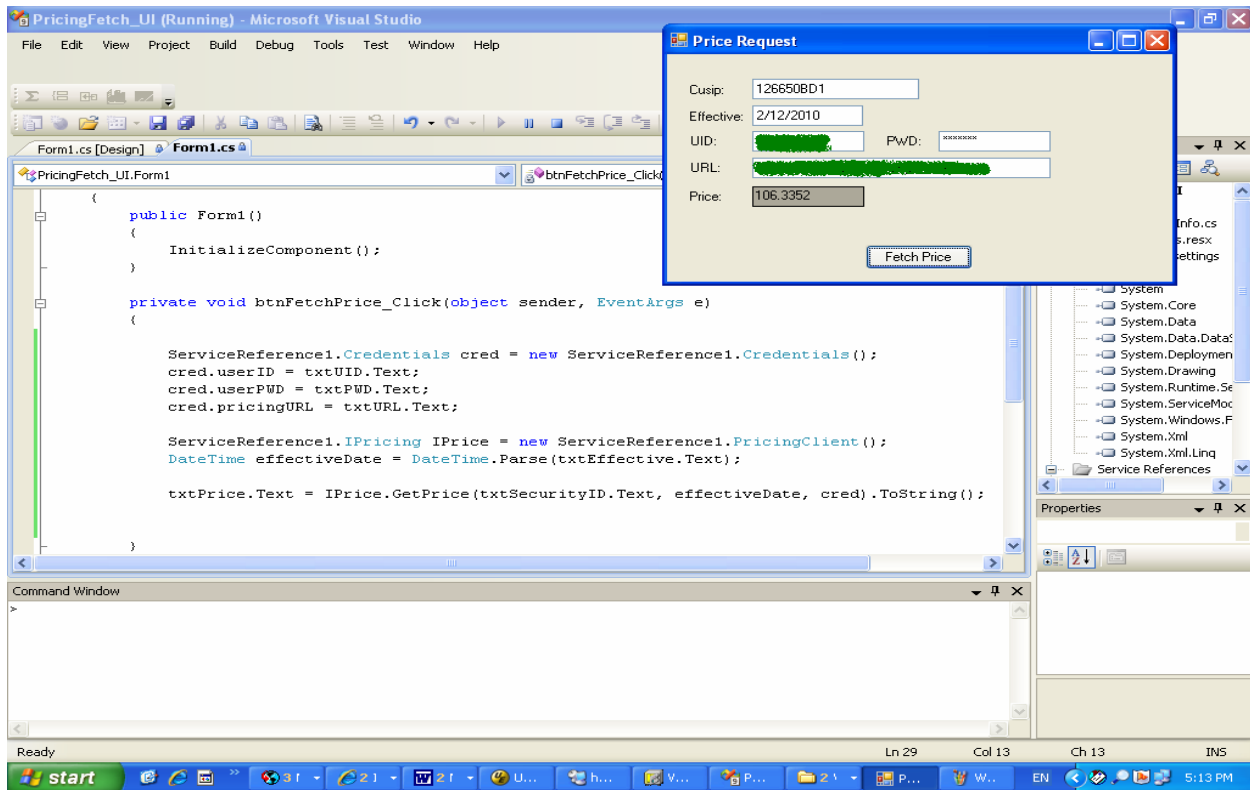


Figure 9: Consumer Access to the WCF Remote Service

### 4.3 Source System Adapters

The first flexibility assessment involved comparing source system extractions between a classic data warehouse implementation and two SOA adapters. The current data warehouse serves primarily as a reporting and decision support environment for investment analysts and fixed-income bond traders. It is also provides investment performance reports for clients. The primary data providers for the data warehouse are:

1. An externally managed trading system.
2. An internally managed accounting system.
3. An externally managed bond pricing system.

The following figure shows a high level diagram of these providers in the investment data warehouse.

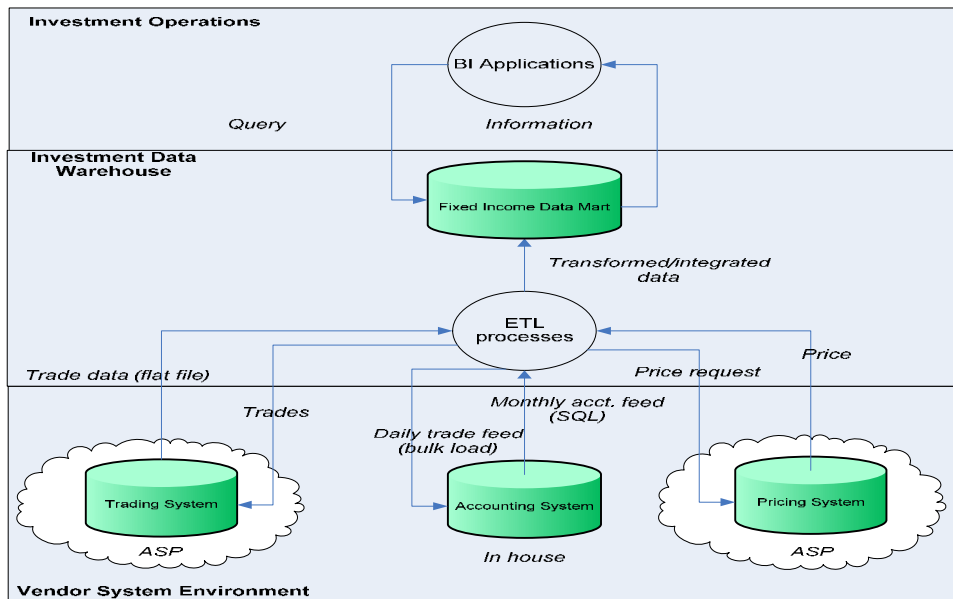


Figure 10: The Investment Data Warehouse Environment

As can be seen in the preceding diagram, access to data providers is done in a few different ways. The trade feed FTP/flat file paradigm is probably the most common method used in today's data warehouse environment. The ASCII format is portable across platforms and easy to parse and read using any number of tools. The use of ODBC to access a relational backend, as seen with the accounting system, has also become a common approach to data warehouse extractions over the past few decades. The HTTP approach is a bit more cryptic, because it involves formatting and parsing strings in an application-specific manner.

Both the ODBC and HTTP interfaces were chosen for conversion to SOA adapters as a means of assessing their flexibility to those of the current warehouse. The following benchmarks were used to judge flexibility between the classic warehouse and SOA approaches:

Criteria	Comment
Data Access	How flexibly can data be obtained?
Schema abstraction	Is one architecture more flexible in its logical/conceptual schema than the other?
Back-end changes	Are back-end changes more flexibly managed in one architecture vs. the other?
Extensibility	Does the architecture allow providers and/or consumers ample opportunity to extend functionality?

Table 2: Source System Extraction Flexibility Criteria

### *The Accounting System Adapter*

The current data warehouse extraction process for accounting data is ODS in nature; it basically copies the source accounting table schemas to the data warehouse repository via ODBC SQL calls. The returned records are stored in delimited text format and then loaded into the data warehouse tables using import tools native to that environment. This approach is fairly flexible

from both a real time and batch perspective. As long as the backend database engine is available the data contained therein can be obtained.

The accounting system schema is documented by the vendor in a HTML data dictionary. This approach suffices for query development purposes. One can easily search for keywords in the data dictionary and determine relationships between tables. However, minimal metadata exists at runtime that can be used to query the meaning of fields and their relationship to one another. A QBE tool like MS Access might be able to derive some relationships based on primary/foreign key names, but other metadata is not available to query tools.

Vendor upgrades have at times required that deprecated and/or new tables and fields are accounted for, and that queries are re-coded to account for such changes. The vendor has also upgraded their SQL data engine, in this case Btrieve, a few times over the years. This has created few issues with the queries in use in the data warehouse. However, a future change to another backend engine such as Oracle or Microsoft SQL server would likely require a wholesale recoding of SQL extraction code.

The foregoing analysis of the accounting system shows that it is flexible in terms of data accessibility. A client application requires only the proper permissions and ODBC software in order to transport backend data tables into the data warehouse environment. It is incumbent on the developer of such queries however to analyze the data dictionary in order to determine the context and usage of source data, as well as data table's relationships to one another.

Furthermore, a significant change to the backend schema or database engine type may require substantial recoding of data warehouse interfaces.

An SOA adapter was developed in order to determine if an object-based approach to extracting accounting data yields more benefits in terms of flexibility. The previous section

contains details on how adapters are developed using Microsoft .NET and the WCF architecture. The steps used to create, deploy, and consume services are covered there. It is important to note that in the upcoming examples deployment is being done from the perspective of the accounting vendor, not the data warehouse developer. Obviously an SOA adapter developed in-house would be subject to all the same back-end vagaries as that of traditional SQL code modules (e.g. schema changes, deprecation of data elements). Therefore, the question being framed is whether a data provider allows more flexibility to its consumers if the interfaces provided are object rather than SQL-based.

The WCFLibAccounting adapter that was developed contains methods for fetching security master file (SMF) and transaction data from the accounting system. The two parts of the an SMF record are its issuer and issue components. The “issue” is the security itself, which consists of the issue’s CUSIP, name, and miscellaneous accounting information. The “issuer” information relates to what entity issued the security. A one-to-many relationship exists between issuers and issues:

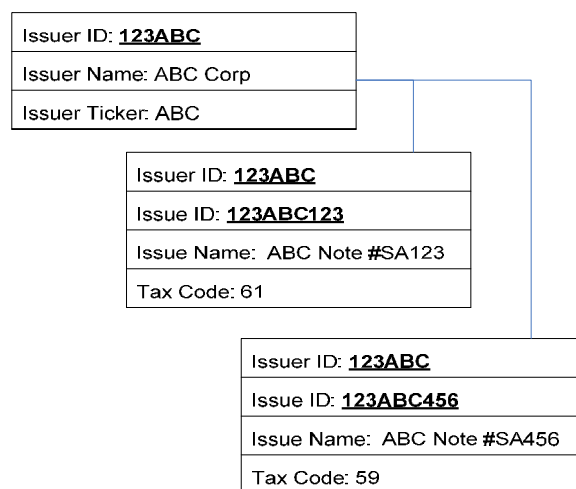


Figure 11: Security Issuer/Issue Relations



The `WcflibAccounting.getSMF()` method provides an interface to issuer and issue records. By default it fetches all current SMF's from the accounting system. A related method, `getSMFByCusip(string cusip)`, allows for discrete SMF fetches. Interestingly, WCF does not allow for overloaded method calls, which will limit implementation flexibility for that particular architecture. The `getSMF*()` methods wrap the SQL in an object-based interface. The code on the left side of the following figure is accessed by the adapter's consumer in a more abstract manner. A test client interface can be seen on the right side of the figure.

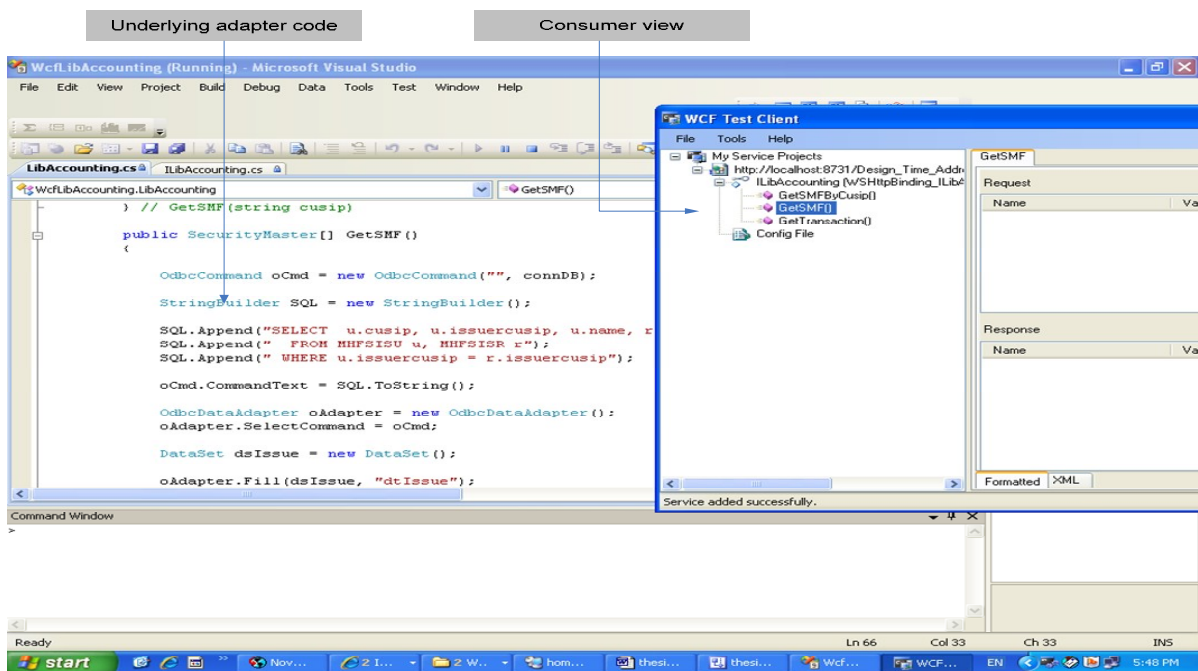


Figure 12: SMF Adapter and Consumer Interface

The `WcflibAccounting.getTransaction` method is similar to the `.getSMF()` method, except that it allows for date filtering. The investment data warehouse pulls daily and month-end transaction data. The daily data is more operationally oriented because it provides cash balances needed by bond traders, while the month-end data pulls support client performance reporting. It is

important to recognize that SOAP and XML underlie all communications between the consumer and the service. The following figure shows the actual SOAP request and response messages involved in a `getTransaction()` call:

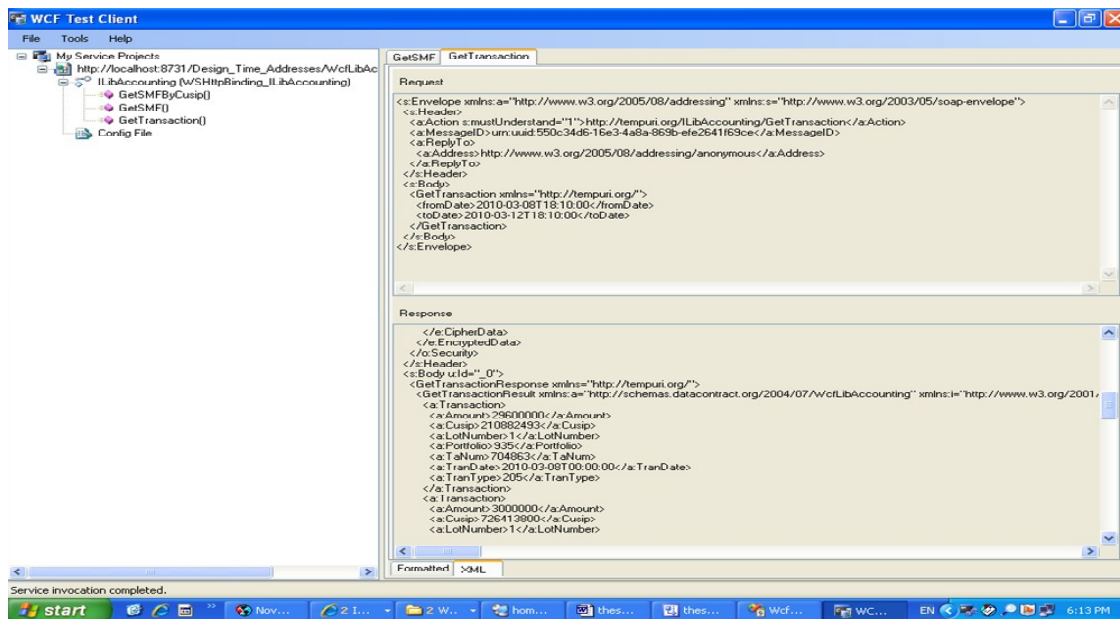


Figure 13: The Underlying SOAP Message Packets.

The following diagram shows the classes and interface that make up the accounting system adapter:

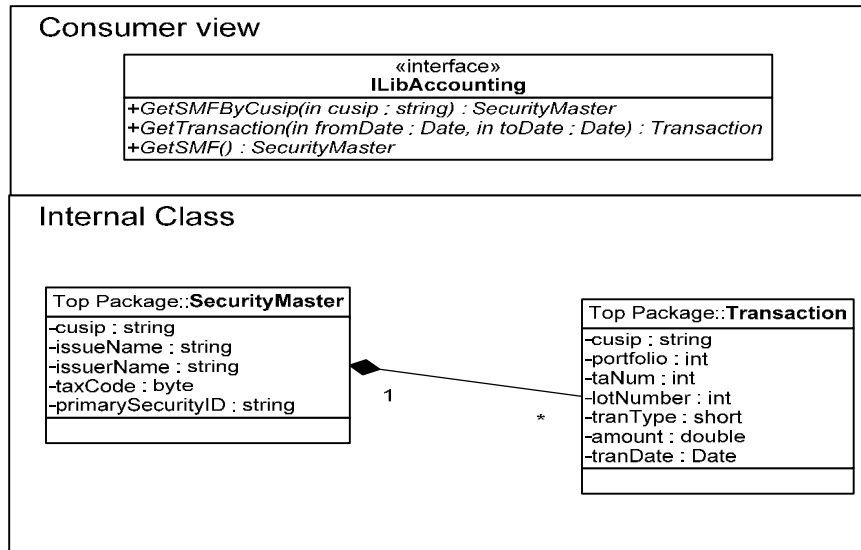


Figure 14: The WCFAccounting Class Diagram

A few observations can be made at this point relative to the flexibility of the SOA approach to that of direct SQL calls. As the preceding examples show, the SOA adapters hide the backend logical schema from the data consumer. This will provide more flexibility to the consumer if the backend schema undergoes a significant structural change in future releases. The onus is on the provider to ensure that the interfaces are maintained. On the other hand, one can argue that an object-based access to backend data might constrain an experienced SQL developer who is more than capable of fetching data in a number of ways. The stove-piping of data via strict interfaces might prove more of a hindrance to such people. However, as mentioned previously the accounting system is fed to the data warehouse using the operational data store principle, which basically consists of copying one schema to another. Subsequent complex SQL joins are more apt to be done in the data warehouse environment after initial extractions from source systems have been done.

Another observation regarding flexibility is that the XML return sets reflect the data classes defined in the adapters. This allows for a fair degree of structural flexibility over that of

SQL data sets. For instance, metadata can be added to classes in order to better explain the context of the data being returned from the source system. Such data can be co-located with data rows or can be associated with all or part of the returned dataset. This data in turn can be used downstream in the data warehousing process in a number of ways, including data cleansing, transformation, auditing, and error checking. Obviously such functionality can be incorporated into traditional SQL feeds via customized metadata. However, the use of object-based access to backend data allows for a level of cohesiveness not inherent to the row and column paradigm of traditional SQL-based systems.

A final observation is that flexibility for the consumer in the SOA environment does not necessarily equate to flexibility for the provider. As stated earlier, the examples in this section use the premise that the vendors themselves provide the adapters. This requires them to maintain interfaces to their underlying data stores, an approach that might seem anything but flexible to the developers who have to support such an architecture. It could be argued that a well documented SQL repository should suffice for the needs of most data consumers. A counter-argument is that clients are apt over time to develop custom applications around an open vendor schema. This may cause upgrade troubles if the clients have used database-specific approaches such as PL-SQL for Oracle, and the vendor later decides to switch to Microsoft SQL Server. In this light the development and maintenance of SOA adapters might provide for more flexible migrations from the consumer perspective.

*The Pricing System Adapter*

The PricingVendor01 adapter developed in section 4.2 is now used to further illustrate SOA wrapper functionality for the pricing vendor services. The following class diagram depicts this adapter:

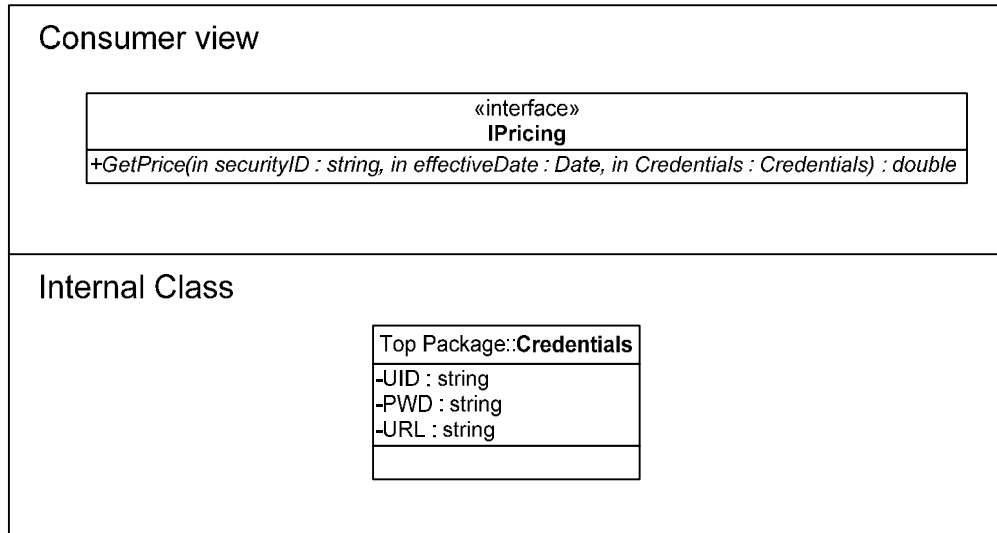


Figure 15: The PricingVendor01 Class Diagram

Prices are fetched daily for a sub-set of investment securities and at month-end for all holdings. In contrast to the accounting system, the pricing vendor provides data via HTTP requests. The requests themselves use a somewhat obfuscated calling mechanism. The following figure shows an example of a price request for two securities for the date 2/11/2010:

```
Request=get%2C(126650BD1%2C24422EQV4)%2C(prcadat%2Cid9%2Cprc%2Cdes1%2Cdes2%2Cprcnote%2Cassettxt)%2C20100211&Done=flag
```

Figure 16: Proprietary HTTP Price Request Post

As seen in this example, the vendor provides additional data such as security descriptions (des1, des2) and additional prices notes (prcnote). A SOA adapter can abstract such complexity via an object-based interface. The following figure shows the client code for a similar request:

```
txtPrice.Text = IPrice.GetPrice(txtSecurityID.Text,  
effectiveDate, cred).ToString();
```

Figure 17: The Object-Based Approach

The flexibility of an SOA using HTTP in the foregoing example is somewhat clearer than in the SQL example. The vendor has implemented an obscure way of fetching data via a proprietary implementation. An abstraction layer on top of this allows the data warehouse consumer to focus more on the “what” than on the “how” of data fetching. Furthermore, consider if the vendor later decides that HTTPS is a more appropriate mechanism for transporting sensitive pricing data over the Web. This would require the consumer to recode their interfaces. A SOA middleware adapter on the other hand provides a suitable means for hiding connectivity issues from the data consumer, who can continue to use regular HTTP.

This same line of reasoning can be applied to other non-relational file structures such as fixed length text or vendor proprietary formats. A structural change to these inputs can introduce subtle data quality issues in the downstream data warehouse processes. For example, the trading system illustrated earlier provides a nightly fixed length text feed to the data warehouse via FTP. Various investment security types exist within the same file, each with its own fixed length format. One line in the file might be parsed in a particular manner because it is a commercial mortgage, while a corporate security is handled in another way. Additionally, data elements might get added to the end of lines as users request new information from the vendor. In this

scenario the parsing burden is shifted to the data warehouse extraction process. It can be argued that a SOA adapter employing an interface would allow more flexible, if not safer, access to such data.

### *Recap*

Returning to the benchmarks outlined in table at the beginning of this section, a few conclusions can be made regarding the flexibility of classic data warehousing and the SOA in the extraction layer of a decision support environment.

	Classic Data Warehousing	SOA
Data Access	It depends on the implementation. Relational DB's are good, proprietary access methods can be challenging for the consumer.	The interface is always object-based, which provides consistency in how data elements are accessed.
Schema abstraction	The consumer is more bound to the logical back-end schema. This can be good if the developer is well-versed in the language being used and how relationships exist on the back-end.	The consumer is buffered from the back-end schema. This allows the focus to be more on the "what" than the "how" of data access.
Back-end changes	The consumer is apt to be affected by substantial schema changes on the back-end, especially by dialect changes due to database migrations.	The back-end changes will not affect the consumer, as long as the provider maintains the same interfaces.
Extensibility	Access to an open schema, especially a relational one, provides a lot of flexible approaches to data extraction.	The consumer is locked into a specific interface. This can detract from flexibility in terms of data extraction. Also, the data provider has to do a lot of work in order to develop and maintain data flexible access to data.

Table 3: Source System Extraction Flexibility Synopsis

#### 4.4 Integration Adapter

This section compares the flexibility of the SOA to classic data warehousing for transforming and integrating source system data. The following benchmarks were established for this purpose:

Criteria	Comment
Data Integration	Does the architecture allow for flexible integration of disparate data sources?
Data Reconciliation	Does the architecture have mechanisms for reconciling incoming data?
Data Transformation	Does the architecture flexibly perform transformation tasks such as data tagging and summarization?

Table 4: Data Transformation/Integration Flexibility Criteria

##### *The Integration Adapter*

The current investment data warehouse utilizes an operational data store, into which source system data are imported, transformed, and merged. The data are also summarized and stored in system tables, which are then used as the basis for queries and reports. An integration SOA adapter was developed in order to assess its flexibility the to that of the legacy processes. The accounting and pricing system adapters were used as the inputs to the integration adapter. The following test scenario was established to assess the flexibility benchmarks shown in the preceding table:

Using the WCFLibAccounting.Transaction service contract, fetch all transactions for a monthly period.

Scan all rows of the return data set and make the following transformations:

- For sales-type transactions (transaction type starts with “2”) tag the transaction as “SALE”. For purchase-type transactions (“1\*”) tag the transaction as “PURCHASE”.



- Tag the month and year of the transaction (e.g. “1/2/2010” will get tags “01” and “2010”).
- Fetch the associated issuer/issue information via the WCFLibAccounting.SecurityMaster data contract, based on the transaction CUSIP. If an issuer cannot be found flag apply “UNKNOWN” for the issuer name and apply a metadata “MISSING\_ISSUER\_NAME” flag. Do the same for the issue name. If the issue tax code contains (20,21,30,31) flag it as “TAX\_EXEMPT”, else flag it as “TAXABLE”. If no issue was found, default the issue tax status to “UNKNOWN”.
  - Fetch the associated closing price from the PricingVendor01.IPricing service contract, based on the transaction CUSIP and date. If a price is not returned from the pricing adapter, default to 0 and apply a metadata tag called “MISSING\_CLOSE\_PRICE” to the data row.

The following diagram expresses the preceding logic in UML activity diagram format:

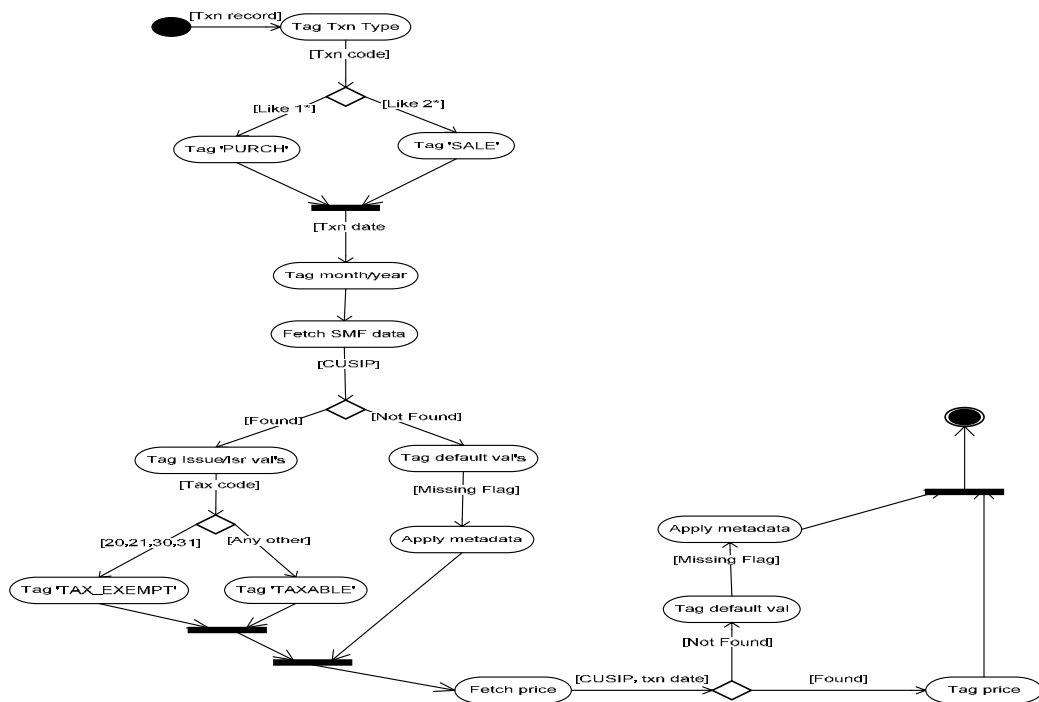


Figure 18: The Transformation Activity Diagram

An SOA adapter called dwTxnToCube was developed in order to implement the preceding logic. The adapter utilizes the services of both the WCFAccounting and PricingVendor01 SOA adapter interfaces for fetching data. The integration of these adapters was done using the same techniques discussed in section 4.2. Data transformations are made within methods of the integration service itself. The adapter’s output is a structure (txnCube) that represents the summarized data in cube format. The following diagram illustrates the relationships between the source adapters and the cube output.

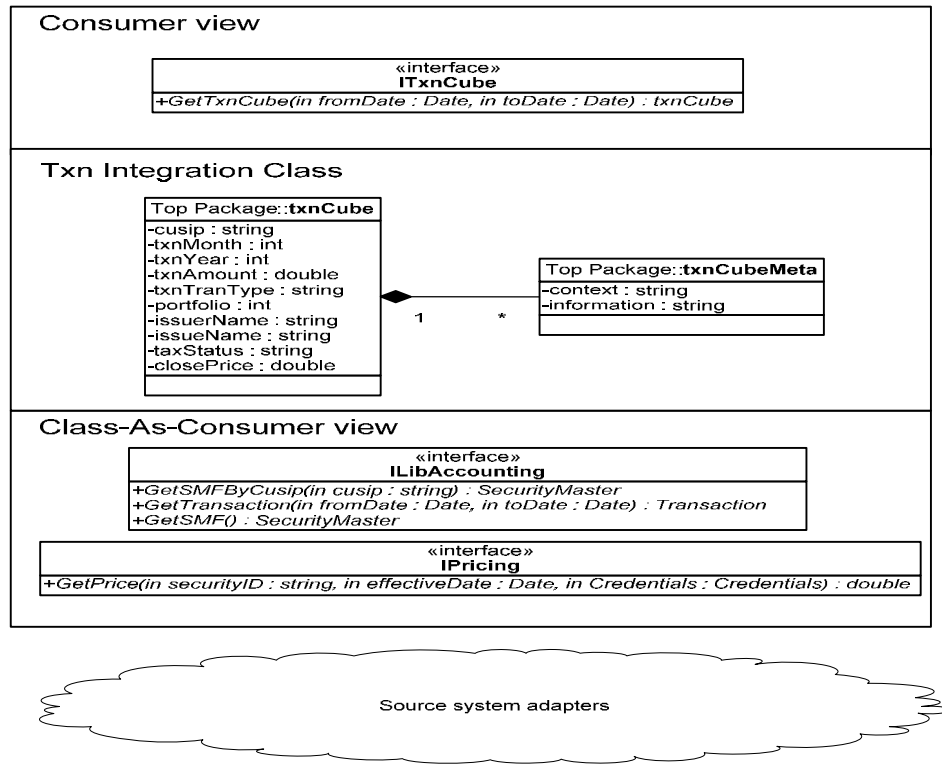


Figure 19: The Transaction Cube Adapter Class

The development of the SOA integration adapter was helpful in determining the flexibility of such an approach to that of classic data warehousing. In terms of data integration, the SOA once again benefits from the use of source system interfaces rather than vendor-specific data access methods. This allows, for instance, the merging of the accounting transactions and their associated security master data without having to know the underlying source system schema. The developer is not completely shielded from the relations within the accounting system however. For example, the accounting system utilizes the CUSIP identifier for internally-defined numeric codes, while the primarySecurityID identifier contains the industry-recognized identifier. The latter ID is used to fetch prices from the IPricing interface.

Despite such semantic challenges, one can see that the txnCube interface provides the consumer convenient access to an integrated cube structure. It can be argued that the classic data warehouse provides such functionality already via cube tables in the warehouse repository. Additionally, SQL views can be created that perform back-end joins for the user, thereby performing similar abstract integration services to those found in the SOA adapter. The SOA adapter however provides the user the flexibility of real-time, as well as, historical access to integrated data. In fact, it is likely that the ITxnCube interface will be used to both load warehouse data tables on a recurring basis, as well as to provide direct access to back-end production systems. The user need not be aware of the difference.

Data reconciliation within the SOA adapter is not different than that of the code modules within the classic data warehouse. Data anomalies are identified, tagged, and recorded. The SOA's use of XML may provide some structural flexibility over that of the classic approach. As seen in the previous diagram every txnCube instance has an associated 1:M txnCubeMeta instance. The metadata class allows for storage of data exceptions, as well as other information regarding the extraction. The classic data warehouse approach can utilize similar techniques. For instance, during the creation of a cube structure all exceptions can be written to a log file or log table, which contain pointers back to the cube's rows. The SOA approach will also do this if the adapter is being used to populate a data repository. The use of classes in the SOA however provides the opportunity to serialize and deserialize object instances in and out of a repository. This provides the flexibility of retaining the original object relations and hierarchies.

Data transformation is probably the most important step in the ETL process. The classic data warehouse traditionally does this through a top-down program code module. Raw data are brought into a temporary schema and integrated. The data are then scanned and tagged based on

transformation rules stored either in program code or in metadata structures such as cross-walk tables. More modern transformation processes might utilize object-based tagging routines. This technique allows for more flexibility in tagging data because transformation semantics are not reproduced in many different modules. The SOA class utilizes such an approach. A consumer who wishes to know whether a transaction is a purchase or sale will use the services of the ITxnCube adapter. The adapter might also contain metadata that explains the rationale for a transaction being tagged a specific way (i.e. [TRANTYPE >= 100 and TRANTYPE <= 199] = “PURCHASE”).

The example used in this section provides only a simplified view of such transformation logic. A more robust ITxnCube interface would contain a method for each type of transformation used to create the cube. This allows a greater degree of flexibility than the classic approach because consumers are directly exposed to the inner workings of how data transformations occur, supporting Inmon’s view of the data warehouse as the information hub of the enterprise. The classic data warehouse on the other hand normally consists of a static data dictionary and perhaps a crosswalk diagram.

### *Recap*

Returning to the benchmarks outlined in table at the beginning of this section, a few conclusions can be made regarding the flexibility of classic data warehousing and the SOA in the integration layer of a decision support environment.

	Classic Data Warehousing	SOA
Data Integration	Back-end schemas are more likely to be exposed, complicating joins. Data are batched to a static repository. Real-time access to data is typically done outside of the warehouse,	Use of adapter interfaces simplifies joining systems. Semantic issues can still exist, however. The output can be static or real-time, allowing for

	requiring integration to be duplicated.	flexible access.
Data Reconciliation	Data anomalies are typically written to logs, which can be cumbersome for audit purposes.	The use of classes and object serialization allows for tighter integration of metadata and warehouse data.
Data Transformation	Typically done in a top-down fashion. Users normally do not have access to internal logic, except through offline data dictionaries.	Transformation logic can be made more granular via class methods, allowing the user more insight into how data are tagged.

Table 5: Data Transformation/Integration Flexibility Synopsis

#### 4.5 End-user Access

The previous sections dealt with the back-end mechanics of how data are fetched, integrated and transformed. Such activities fall in the realm of the source system and data warehouse processing tiers. Obviously, the data warehouse exists for the end-user. As Kimball (2008) shows, the end-user, or consumer, interacts with the data warehouse in a number of ways:

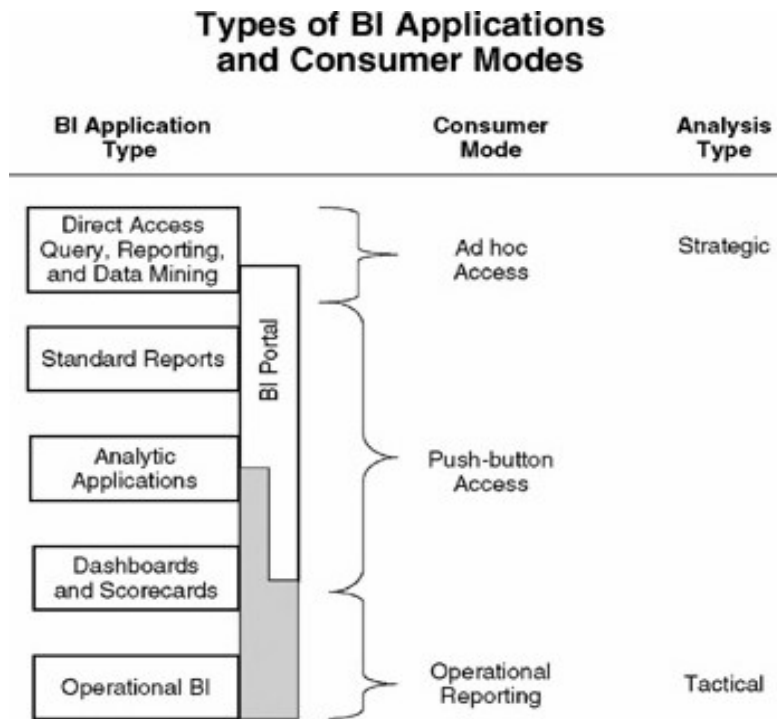


Figure 20: Kimball’s Consumer Mode Stratification

As the previous figure shows, four of the five consumer tiers are accessed via canned reports or preconfigured BI applications. A data warehouse software developer will typically configure these applications for the user, either using vendor-specific tools or a software development environment such as .NET or Java. The consumer then uses a parameterized GUI or Web form to query the underlying data, unaware of the actual structure of the warehouse repository.

The ad hoc/strategic consumer of the data warehouse however interacts with the data warehouse in a less structured manner. These so-called power-users utilize query tool and drill-down tools in order to analyze data stored in data warehouse ODS or star schema structures. The question is whether an SOA will help or hinder such users to flexibly access and query such data. The criterion for this section is whether the classic ODS and/or star schemas provide more flexibility than the more abstract, object-based SOA.

As the following diagram shows, user access to the classic data warehouse and SOA architectures differ significantly from one another:

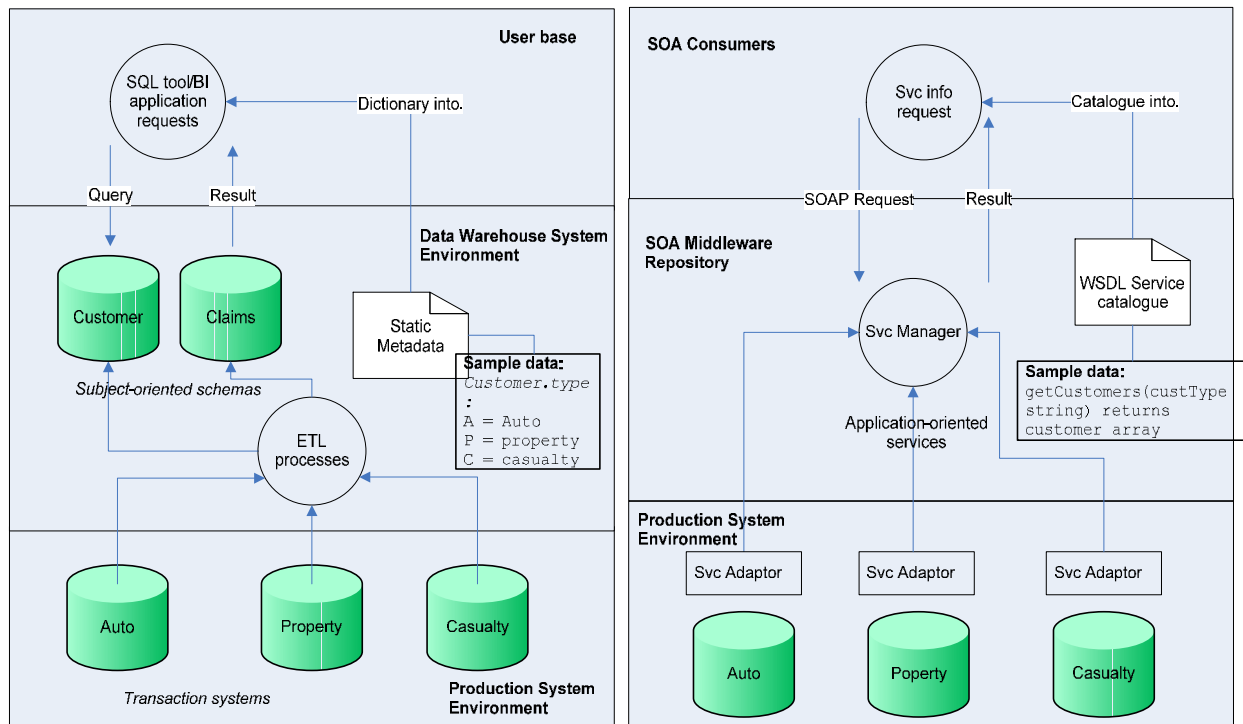


Figure 21: The Data Warehouse and SOA Tiers

The middle tier of the classic data warehouse contains static database tables. These may either mimic the source database system's schema or be summarized in a star schema dimensional model. There are a number of advantages to this approach from a flexibility perspective. A finely grained warehouse schema allows ad hoc approaches to querying and integrating data. A business user who is well versed with the semantics of the backend systems can become very productive in a short period of time. Most query tools allow the user to view underlying schemas and to generate fairly complex SQL via a GUI interface. Users can also integrate the contents of the data warehouse repository with their own data repositories using ODBC or data import tools, thus expanding the reporting environment to meet their own needs.



The SOA on the other hand presents the user with interfaces to the underlying data. While this can prove expedient for the untrained user, it might in fact hinder the productivity of the experienced data warehouse user. For example, the ITxnCube developed in the previous section contains a method called GetTxnCube() that returns a cube structure for a specific transaction time period. A power user may find this method cumbersome to use if they wish to apply additional filters on the cube, such as tax code type. The data set must first be requested and then further filtered in another processing environment (probably a temp folder on the server or the user's desktop). The user will also likely find it awkward to join returned data with other data sets. For instance, the user of the GetTxnCube() method might want to query prior month-end prices in order to report on month-to-date price swings. To do this, the classic data warehouse user simply drags another table alias into the QBE and does a join. The SOA user however has to do post-data querying in another environment such as MS Excel, or request that additional methods be added to the ITxnCube service.

This is not to say that the SOA approach is inflexible for ad hoc operations. It does require however that the classic warehouse user accept a paradigm shift in how they interact with the data. Less technical users may in fact like the request/response approach for fetching data. The SOA method calls resemble the function calls one sees with macro-enabled tools like Microsoft Excel. An Excel user might for instance utilize a macro call to populate the spreadsheet (via a shim .DLL) and then utilize parameter-based macro calls to the IPricing interface to bring in prior month pricing data (again, via a shim .DLL):

	A	B	C	D	E	F	G	H	I	J	K	L	M
1	Cusip	IssueName	TaxStatus	TranType	TotAmount	Month	Year	AvgPrice	PriorMonthAvg	Diff			
2	1234567AAA	City of St. Louis	TaxExemp	Sale	4500000	1	2009	95.45	=macroPrice_G	3.12%			
3	1234567AAA	City of St. Louis	TaxExemp	Purchase	3500000	1	2009	93.78	94.00	-0.23%			
4	3456789BBB	Png Corp	Taxable	Sale	400456	1	2009	100.12	99.78	0.34%			
5	3456789BBB	Png Corp	Taxable	Purchase	98000	1	2009	101.45	100.50	0.95%			
6	etc...												

Figure 22: User Integration of SOA Services

The “join” between the initial return set and the price object takes place via explicit method calls to the macroPrice() function. This technique is fairly unambiguous and has the advantage of hiding the underlying schemas from the user. Significant developer support will likely be needed to accomplish this however.

### Recap

The classic data warehouse appears to provide more flexibility for the power user who needs to perform ad hoc and analytical queries on the underlying schema. The SOA on the other hand requires a certain degree of programming know-how in order to consume the services of backend systems. However, the flexibility of the SOA can be seen in its ability to shield the end user from the back-end schema of the warehouse itself.

## **Chapter 5 – Project History**

### **5.1 Project Origins**

The primary driver for this project was the phasing out of vendor support for the current software language used to manage an investment data warehouse. A review of the .NET architecture revealed opportunities to revamp the approach to accessing, integrating, and deploying financial data using an SOA. An initial analysis of this approach revealed that the use of an SOA was not so much a question of feasibility as it was of flexibility. In other words, would the SOA help or hinder the operational aspects of managing the warehouse process compared to the classic data warehouse approach? This research project set out to answer that question.

### **5.2 Scope**

The project scope was framed by the three primary tiers of the classic data warehouse environment: (1) data extraction, (2) data integration/transformation, and (3) data access. The secondary research conducted as part of the literature review helped to uncover trends that could be further explored and validated in the primary research.

### **5.3 Project Management**

The project consisted of two major phases, secondary research and primary research. The secondary research involved a literature review of both the classic data warehouse and the SOA, from which initial conclusions were drawn regarding flexibility. The primary research project was undertaken in a waterfall manner. A feasibility study was conducted using

Microsoft's WCF/.NET architecture. The three tiers of the classic data warehouse were then analyzed and developed and/or assessed.

#### **5.4 Milestones**

The major project milestones were:

- a. Completion of the first thesis module and acceptance of the thesis statement by my advisor.
- b. Conducting secondary research.
- c. Periodic reviews with my thesis advisor.
- d. Completion of secondary research.
- e. Completion of the second thesis module.
- f. Obtaining .NET developer license and a Windows 2003 server resource.
- g. Completion of primary research.
- h. Combining, analyzing and summarizing findings.
- i. Submitting research to advisor for review.

#### **5.5 Changes to the Plan**

The only major change to the plan was the initial timeline. This project was originally estimated to be completed by October 2009. However, an unexpected software project delayed work on the thesis for about four months.

#### **5.6 Evaluation**

Overall, I am satisfied with how the thesis project went. It is sometimes difficult to foresee obstacles, both professional and personal, that might interfere with the timeline.

### **5.7 Summary**

The research project generally went according to plan, with the exception of an unexpected delay due to a large software project at work.

## **Chapter 6 – Conclusions**

### **6.1 Statement of Findings**

An SOA provides flexibility for data warehouse activities in the backend tiers relating to data access and integration. This flexibility is mainly due to the abstracting of source system schemas via object-based interfaces. Additionally, the availability of source system SOA adapters allows for more seamless combining of real-time and batched data in the integration tier. Flexibility diminishes however as one approaches the end-user tier. A power-user in the classic data warehouse is likely to have more flexibility because of the relative ease in which query tools can be used to access relational or star schema data stores in that environment. The SOA approach requires developer support, mainly due to the relative newness of this technology.

### **6.2 Major Themes Uncovered**

The following themes were uncovered through the primary and secondary research:

1. An SOA, based on the .NET/WCF architecture, is fairly easy to set up and deploy.
2. Source system SOA adapters allow abstraction of source schemas, thereby easing upgrades, and simplifying interfaces.
3. The use of SOA adapters for real-time system access, combined with warehouse batch data, can provide the user a more flexible means of concurrently viewing both current and historical data.
4. The SOAP and XML standards provide an intuitive and extensible way in which to format request and response messages. Additionally, the semi-structured nature of XML allows for tighter integration of business data and operational metadata.

5. The classic data warehouse is based on mature technologies and standards. A power-user with knowledge of the business semantics can approach ad hoc reporting and data analysis in a flexible manner. The SOA end-user will likely rely on BI applications, which limit flexibility.

### **6.3 Research Limitations**

The research was conducted using a Windows platform and Microsoft development tools. Although this reflects the actual production environment at my company, it leaves open the question of the ability to deploy an SOA in a heterogeneous system environment. For instance, can a Java developer easily create clients that consume WCF back-end services, and vice-versa? How about communications between non-Windows and Windows servers? Such questions provide future research opportunities.

### **6.4 The Project in Hindsight**

The overall project went as planned. I initially attempted to cover too much ground in the literature review, which ended up leading to a significant rewrite of that section. It is important to review an outline with one's thesis advisor before getting too far into a section.

### **6.5 Research Opportunities**

This project revealed the following research opportunities:

1. The interoperability of an SOA across different system platforms.
2. The use of XML as a transport medium for large data sets.
3. The use of a master data management service for real-time data integration and reconciliation.

4. The use of XML/XQuery as an alternative to RDBMS/SQL for decision support data storage and reporting.



### References

- Arvin, T. (2009). *Comparison of different SQL implementations*. Retrieved June 20, 2009, from <http://troels.arvin.dk/db/rdbms/>.
- Berson, Alex, and Lawrence Dubov (2007). *Master Data Management and Customer Data Integration for a Global Enterprise*. McGraw-Hill/Osborne. Books24x7.
- Braun, C. and Winter, R. (2007). *Integration of IT service management into enterprise architecture*. ACM Press, New York, NY, 1215-1219. Retrieved May 18th, 2009, from <http://doi.acm.org>.
- Codd, E. F. 1970. *A relational model of data for large shared data banks*. Commun. ACM 13, 6 (Jun. 1970), 377-387. Retrieved June 14, 2009, from <http://doi.acm.org>
- Drewek, K. (2005). *Data Warehousing: Similarities and Differences of Inmon and Kimball*. Retrieved June 12, 2009 from <http://www.b-eye-network.com/view/743>.
- Fritz, David (2006). *The Semantic Model: A Basis For Understanding and Implementing Data Warehouse Requirements*. Retrieved August 10, 2009, from <http://www.tdan.com/view-articles/4044>.
- Hartmann, S. (2008). *Überwindung semantischer Heterogenität bei multiplen Data-Warehouse-*

*Systemen [Overcoming semantic heterogeneity in multiple data warehouse systems].*

University of Bamberg Press. Retrieved June 14, 2009, from [http://www.opus-](http://www.opus-bayern.de/uni-bamberg/volltexte/2009/160/pdf/StHartmannDiss.pdf)

[bayern.de/uni-bamberg/volltexte/2009/160/pdf/StHartmannDiss.pdf](http://www.opus-bayern.de/uni-bamberg/volltexte/2009/160/pdf/StHartmannDiss.pdf)

Inmon, W. H. (2005). *Building the Data Warehouse*, Fourth Edition. John Wiley & Sons.

Books24x7.

Inmon, W. H. (2007). *Content versus Semantic Changes*. Retrieved August 10, 2009, from

<http://www.b-eye-network.com/view/6388>.

Kimball, R., Caserta, J. *The Data Warehouse ETL Toolkit: Practical Techniques for Extracting,*

*Cleaning, Conforming, and Delivering Data.* (2004). John Wiley & Sons. Books24x7.

Kimball, Ralph. *The Data Warehouse Lifecycle Toolkit*, Second Edition. (2008). John Wiley &

Sons. Books24x7.

Rainardi, Vincent. (2008). *Building a Data Warehouse: With Examples in SQL*. Springer-

Verlag, New York, NY. Books.google.com.

Schepers, T. G., Iacob, M. E., and Van Eck, P. A. (2008). *A lifecycle approach to SOA*

*governance*. ACM, New York, NY, 1055-1061. Retrieved May 18th, 2009, from

<http://doi.acm.org>.

Scribner, Kennard, and Mark C. Stiver. (2000). *Understanding SOAP*. Sams. Books24x7.

Seeley, R. (2007). *XQuery, the SQL for SOA, wins final W3C approval*. Retrieved June 14<sup>th</sup>, 2009 from

[http://searchsoa.techtarget.com/news/article/0,289142,sid26\\_gci1240463,00.html](http://searchsoa.techtarget.com/news/article/0,289142,sid26_gci1240463,00.html)

Sperberg-McQueen, C. M. (2005). *XML <and semi-structured data>*. *Queue* 3, 8 (Oct. 2005), 34-41.

### Appendix A – Revision History

Date	Comments
5/19/10	8 <sup>th</sup> draft to thesis advisor - Minor edits.
5/6/10	7 <sup>th</sup> draft to thesis advisor - Included thesis statement, sub-sections in introduction section. - Additional APA formatting.
4/18/10	6 <sup>th</sup> draft to thesis advisor - Complete primary research, chapter 4. - Completed remaining chapters. - Shortened chapter 2.
3/20/10	5 <sup>th</sup> draft to thesis advisor for review - Edited chapter 2, Literature Review. - Completed chapter 3, Methodology. - Completed section 4.2 of chapter 4, primary findings, extraction adapter. - Added Appendix B. Proof of Concept. - Added Appendix C, Glossary of Terms. - APA formatting started. - Added chapter stubs according to template.
10/03/09	4 <sup>th</sup> draft to advisor for review - Moved fig's 1 - 4 closer to the text in which they are referred to. - Figure's 18a,b show examples more appropriate to a data warehouse request. - Expanded on how SOA can better handle backend system changes. - Changed parent section heading from "Flexibility Criteria" to "Secondary Research: Flexibility Criteria". Added Sub-section numbering. - Started section "Primary Research: A SOA Prototype"
8/24/09	3 <sup>rd</sup> draft to advisor for review - Flipped illustrations vertically - Added new section: "How well do the architectures handle data volatility?"
6/22/09	2nd draft to advisor for review. Changes: - Added illustrations to existing sections. - General grammar fixes to existing sections. - Removed irrelevant sections. - Added new section: "How accessible are the architectures?"
6/1/09	1 <sup>st</sup> draft to advisor for review.

Table 6: Thesis Revision History

### **Annotated Bibliography**

Kamoun, F. (2007). A roadmap towards the convergence of business process management and service oriented architecture. *Ubiquity* 8, 14 (Apr. 2007), 1-1.

The author explores the growing trend towards the convergence of business process management (BPM) and SOA, and how such a partnership better aligns the business' goals with that of a loosely coupled and flexible IT infrastructure. Although challenges exist, including BPM and SOA vendor service mismatches and in-flux standards, the payoff gained through being able to provide flexible services to customers makes such a convergence inevitable. Companies should begin planning for such a future by addressing some key areas, including executive sponsorship, enforcing a governance model, training, and prototyping.

(Source: ACM Web Site <http://doi.acm.org/10.1145/1247272.1247273>)

Im, T., Guimaraes, M., and Hoganson, K. (2004). An N-Tier Client/Server course: a classroom experience. In *Proceedings of the 42nd Annual Southeast Regional Conference* (Huntsville, Alabama, April 02 - 03, 2004). ACM-SE 42. ACM Press, New York, NY, 42-45.

The authors describe the outcomes of a graduate level course project they designed that touched upon various aspects of n-tier development, including a PDA interface, synchronization middleware, ODBC, and a relational database. An interesting and challenging aspect of the course project involved the use of Sybase's Mobilelink and Ultralite products for synchronization of the backend Oracle database with the front end PDA application. This required the use of a schema definition that mapped the Oracle data to a smaller middle-tier

database that in turn directly communicated with the PDA devices via MobileVB. This article is relevant to understanding the challenges that arise in implementing an n-tier computing environment, even when the individual components are fairly high-level and easy to learn.

(Source: ACM Web Site <http://doi.acm.org/10.1145/986537.986548>)

Silvers, Fon. "Chapter 9 - Metadata". Building and Maintaining a Data Warehouse. Auerbach Publications. © 2008. Books24x7.

The author explains and illustrates the important role that metadata plays in the maintenance and usage of a data warehouse. Two types of metadata are discussed, static and dynamic. Static metadata provides definitions of data dimensions in the warehouse and is primarily aimed at assisting the business user obtain information. Dynamic metadata contains information relating to ETL processes and the state of the database itself, thereby making more an operational tool. Metadata repositories can be modeled in a number of ways, including dimensionally, relationally, or in a distributed manner. This chapter provides insights into how metadata can be applied to expose the business and operational semantics of the data warehouse to end users. This information is also useful in exploring what tools are available to the warehouse designer and how the reliability and manageability of such an environment can be better ensured.

(Source: [http://common.books24x7.com.dml.regis.edu/book/id\\_26399/book.asp](http://common.books24x7.com.dml.regis.edu/book/id_26399/book.asp))

Inmon, W. H. (2005). " Chapter 2 - The Data Warehouse Environment". Building the Data Warehouse, Fourth Edition. John Wiley & Sons. Books24x7.

The author presents the fundamental aspects of the data warehouse environment: subject-oriented, integrated, non-volatile, time-variant. According to the author these factors provide the user a more stable environment in which to conduct decision support activities. The subject-oriented nature of the data warehouse contrasts with the primarily functional nature of source operational systems. The author contends that this makes the job of the data warehouse user easier than that of the “classical system” user, because data reconciliations have already been accounted for in the ETL process and the granularity of the data in the warehouse allows for more thorough analysis. This chapter provides insight into how classic data warehouse technology provides a link between the business user and source system data. This contrasts with the user/system relationship in the SOA environment, which in many cases concerns a direct link to the production data store.

(Source: [http://common.books24x7.com.dml.regis.edu/book/id\\_12458/book.asp](http://common.books24x7.com.dml.regis.edu/book/id_12458/book.asp))

Lopez, N., Casallas, R., and Villalobos, J. (2007). Challenges in Creating Environments for SOA Learning. In Proceedings of the international Workshop on Systems Development in SOA Environments (May 20 - 26, 2007). International Conference on Software Engineering. IEEE Computer Society, Washington, DC, 9.

The authors contend that the widespread use of SOA requires that academia begin integrating it college-level course curriculums. The challenges to this include an already full IT curriculum at most colleges, the complexity of creating course content (i.e. business processes) that supports SOA training, and the requirement that instructors possess the requisite skills to teach SOA. Students must also be prepared to learn the theoretical and practical concepts of the SOA

framework, as well as being afforded ample lab time in order to apply such concepts. This article enforces the notion that academia must take a lead in formalizing training in support of new systems paradigms in order to prepare students for the workforce.

(Source: ACM Web Site <http://dx.doi.org/10.1109/SDSOA.2007.3>)

Bleiholder, J. and Naumann, F. (2008). Data fusion. *ACM Comput. Surv.* 41, 1. (Dec. 2008), 1-41.

The authors explain how a data fusion layer within the data warehouse ETL processing environment helps to resolve the semantic conflicts which often arise during heterogeneous source system data integration. Data fusion specifically addresses data conflicts which arise when one system reports a different value than another (e.g., system “A” reports a student’s date of birth as 1989, while system “B” reports 1990). The authors present a few strategies that can be applied in handling such anomalies. These range from ignoring data conflicts altogether all the way through resolving conflicts through rules-based metadata. A SQL approach to implementing the approaches is illustrated using JOIN and UNION operators. Finally, the authors provide a survey of current toolsets in the market and their level of support for fusion strategies. This article is a useful primer for the ETL analyst who needs to consider what approach will provide the most efficient means for resolving data conflicts within the warehouse environment.

(Source: ACM Web Site <http://doi.acm.org/10.1145/1456650.1456651>)



Agosta, Lou. (2006). Data warehousing raises the bar on SOA. *Information Management Online*, May 18, 2006. Accessed April 16, 2009.

The author asserts that data warehousing must emerge from its traditional data-centric role in order to better integrate with on-line systems in a SOA environment. A paradigm shift will occur as services, rather than databases, become the central integration vehicles within an organization. So-called third generation data warehouses are expected to reside closer to source operation systems and have lower latency. The author sees the SOA as enabling such functionality, though he does concede that careful consideration needs to be made as to how much data volume can reasonably be handled in a service-centric manner. This article provides insight into the applicability of a SOA in meeting the decision support needs of modern computing environments. As noted in other readings however, a SOA is still in its nascent phases; future implementations will likely have to blend classic data warehouse and SOA methodologies as technologies mature and best practices emerge.

(Source: <http://www.information-management.com/news/1055711-1.html>)

Liu, Z. H., Krishnaprasad, M., Warner, J. W., Angrish, R., and Arora, V. (2007). Effective and Efficient Update of XML in RDBMS. *SIGMOD '07*. ACM, New York, NY, 925-936.

The authors assert that although much headway has been made in optimizing XML query functionality natively within RDBM's, work still needs to be done to address concurrency and other transactional issues associated with DML operations on such data. A number of Oracle 10g XML DML functions are discussed, along with considerations a developer should take into

account when designing and implementing XML DML in the database. The nuances of XML DML are also discussed as they pertain to the underlying storage approach the designer chooses (i.e. CLOB, object-relational, or hybrid). Additionally, the authors provide an overview of how the Oracle approach to XML DML compares to the nascent W3C XQuery Update Facility. This article is relevant for the developer who is considering leveraging a RDBMS such as Oracle in order to perform ETL operations against XML data within the SOA framework.

(Source: ACM Web Site <http://doi.acm.org/10.1145/1247480.1247589>)

Dreibelbis, Allen (2008). "Chapter 2 - MDM as an SOA Enabler". Enterprise Master Data Management: An SOA Approach to Managing Core Information. IBM Press. Books24x7.

The authors explain how MDM can complement the SOA by providing the means for maintaining data understanding, trust, and consistency across the organization. An important aspect of this lies in managing changes that occur as new systems are brought on line and the semantics (both technical and business) of existing systems change. Information as a Service (IaaS) is introduced as a means of managing the data inconsistencies which such changes can bring about. An interesting contrast is made between IaaS and traditional ETL services which utilize MDM capabilities in a more static sense. The authors argue that the latter approach will not scale well if an attempt is made to integrate real-time services. This viewpoint supports Agosta (2006), who sees limitations in classic data warehouse approaches for real-time and decision support system integration.

(Source: [http://common.books24x7.com.dml.regis.edu/book/id\\_27521/book.asp](http://common.books24x7.com.dml.regis.edu/book/id_27521/book.asp))

Marks, Eric A., and Michael Bell. (2006). "Chapter 9 - SOA Business Case and Return on Investment Model". *Service-Oriented Architecture: A Planning and Implementation Guide for Business and Technology*. John Wiley & Sons. Books24x7.

The authors claim that an organization can expect up to 80% savings in IT costs through the implementation of a SOA. These costs are realized not only in reduced development costs through software reuse but also in savings in system integration and software licensing costs. A SOA process value model is presented, which is based on the Soh and Markus IT value model. The model is used by the organization to develop a strategy for the creation, consumption, and measurement of return value for SOA services. A series of metrics are presented by which the organization can then assess progress towards their strategic objectives. This chapter supports the proposition that SOA can lead to increased cost savings within an organization. It is important to note that a strategic vision must guide such an endeavor in order to realize this potential.

(Source: [http://common.books24x7.com.dml.regis.edu/book/id\\_16924/book.asp](http://common.books24x7.com.dml.regis.edu/book/id_16924/book.asp))

Campbell, D. (2005). *Service Oriented Database Architecture: APP server-lite?*. In *Proceedings of the 2005 ACM SIGMOD international Conference on Management of Data* (Baltimore, Maryland, June 14 - 16, 2005). SIGMOD '05. ACM Press, New York, NY, 857-862.

The author explains that the loosely coupled nature of SOA requires that SQL Server be able to act as an independent service for outside requests; this is accomplished by means of enhancements to the product, including a resident common language runtime (SQLCLR) and

HTTP web service in the server memory space, enhanced transactional management via a service broker, and distributed memory notification process that alerts external caches when relevant data has been changed. This article shows how SOA as an architectural methodology may impact how vendors design and enhance their offerings, further blurring the lines between products such as application and database servers.

(Source: ACM Web Site <http://doi.acm.org/10.1145/1066157.1066267>)

Dan, A., Johnson, R. D., and Carrato, T. (2008). SOA service reuse by design. In Proceedings of the 2nd international Workshop on Systems Development in SOA Environments (Leipzig, Germany, May 11 - 11, 2008). SDSOA '08. ACM, New York, NY, 25-28.

The authors assert that a centralized governance policy for SOA services will better ensure their reuse and extensibility in the enterprise environment, thus leading to reduced costs for software development and minimized risk in managing runtime environments. Four key challenges are identified in regard to governance: (1) establishing common definitions across business lines, (2) managing service creation, (3) managing discovery of services and access to specific functionality, and (4) managing service enhancements. The authors agree with Schepers (2008) that SOA is still in its nascent stage in terms of enterprise-wide deployments. This leads to the conclusion that the governance policy presented is based at least in part on practices which have proven successful in other large-scale project methodologies. For instance, the authors propose that a chain of responsibility be established in order to create a common service language across the enterprise (p. 26). The concept of stewardship of this sort has strong ties to enterprise data

warehouse projects, whereby business stewards are charged with ensuring semantic consistency across business lines. See Kimball (2008, chap. 2) for more details on the data steward role.

(Source: <http://doi.acm.org/10.1145/1370916.1370923>)

Kimball, Ralph, and Joe Caserta. (2004). "Chapter 4 - Cleaning and Conforming". The Data Warehouse ETL Toolkit: Practical Techniques for Extracting, Cleaning, Conforming, and Delivering Data. John Wiley & Sons. Books24x7.

The authors explain the importance of the transformation portion of the extract-transform-load (ETL) process within the data warehouse. This process ensures that incoming data are checked for errors and inconsistencies and that sufficient metadata is created for auditing data quality issues. Techniques are presented by which data sets can be sampled for errors through so-called "data quality screens", which check for reasonableness both in regards to data context and individual values of data. Data conforming is then undertaken in order to integrate incoming data and store it into appropriate dimensions within the star schema. This reading provides an insight into how the data warehouse ETL process creates applies quality management and integration services so that business users can utilize data. Practical ETL design considerations are also presented which should be incorporated into the overall warehouse design process.

(Source: [http://common.books24x7.com.dml.regis.edu/book/id\\_11237/book.asp](http://common.books24x7.com.dml.regis.edu/book/id_11237/book.asp))

Kimball, Ralph. (2008) "Chapter 11 - Introducing Business Intelligence Applications". The Data Warehouse Lifecycle Toolkit, Second Edition. John Wiley & Sons. Books24x7.

The author describes the role of business intelligence (BI) applications within the data warehouse environment. The BI application exists primarily to provide less technically-oriented persons access to pre-written, parameterized reports. The BI report developers are typically persons who are comfortable using query tools and who are subject-area experts in a given field. A BI solution can range from strategic portal implementations to tactical operational reports deployed via dashboards. This chapter provides many insights into the applicability of data warehousing in meeting the reporting needs of an organization across a wide spectrum of activities. Of particular interest is the discussion relating to portal management and metadata access to services. This is helpful in understanding how data warehouse information can be accessed by users.

(Source: [http://common.books24x7.com.dml.regis.edu/book/id\\_24441/book.asp](http://common.books24x7.com.dml.regis.edu/book/id_24441/book.asp))