Analysis of XML and COIN as Solutions for Data Heterogeneity in Insurance System Integration

by

Dilong Zeng

Submitted to the Department of Electrical Engineering and Computer Science
In partial fulfillment of the requirements for the degree of
Masters of Engineering in Computer Science
At the
MASSACHUSETTS INSTITUTE OF TECHNOLOGY
June 2001

© Dilong Zeng, MMI. All rights reserved.

The author hereby grants to MIT permission to reproduce and distribute publicly
paper and electronic copies of this thesis document in whole or in part.

Author .......... Department of Electrical Engineering and Computer Science
May 22, 2001

Certified by ....

Stuart E. Madnick
John Norris Maguire Professor of Information Technology
Thesis Supervisor

Accepted by .............................................

Arthur C. Smith
Chairman, Department Committee on Graduate Students

BARKER LIBRARIES
Analysis of XML and COIN as Solutions for Data Heterogeneity in Insurance System Integration

by

Dilong Zeng

Submitted to the Department of Electrical Engineering and Computer Science
On May 22, 2001, in partial fulfillment of the requirements for the degree of Masters of Engineering in Computer Science

Abstract

Online insurance aggregation is a new trend in re-intermediation that can transform the insurance industry. Many insurance carriers have implemented web services that can sell insurance products directly to consumers on the Internet without any help from insurance brokers. To counter the threat to their traditional business model, insurance brokerage firms are implementing aggregators that add value to consumers by providing convenience and price comparison features. The implementation of online insurance aggregators, however, is hindered by data heterogeneity problems caused by syntactical and semantic differences in the data schemas of the different carriers’ web services. The effectiveness of potential solutions, which include XML, XSLT, SOAP, Biztalk, and data standardization, is evaluated. XML and XSLT, which are excellent candidates for pair-wise solutions, do not scale when there are many web services and frequent changes to their data schemas. SOAP and Biztalk place heavy emphasis on document delivery, but avoid addressing data heterogeneity issues. Data standardization, which can be effective for applications where changes in data schema are rare, is not flexible enough to handle the frequent changes to the web services’ data schemas. A novel approach that builds on the strengths of the Context Interchange (COIN) framework is proposed. This new framework provides a scalable solution that allows carriers to change their data schema without requiring significant changes in the aggregator and without affecting the other carriers’ web services.

Thesis Supervisor: Stuart E. Madnick
Title: John Norris Maguire Professor of Information Technology
Acknowledgments

I would like to thank Professor Stuart Madnick and Dr. Michael Siegel for their support and guidance on this project. Their advice and suggestions were invaluable.

I would especially like to thank Frank Manola from MITRE for his guidance and support. His research on the Semantic Web, XML, RDF, and DAML + OIL had inspired many ideas for my thesis.

I would also like to acknowledge the other students who are in the COIN team, including Allen Moulton, Tom Lee, Aykut Firat, and Harry Zhu.

I would like to thank my colleagues at AIG, who have taught me a great deal about the insurance industry.

Finally, I would like to thank my family and friends for all their support during this past year.
Table Of Contents

1.0. Introduction and thesis road map ................................................................. 7
  1.1. The need for web service aggregation in the insurance industry ............... 9
  1.2. Traditional business models of selling insurance ................................. 9
  1.3. Insurance carriers selling direct on the Internet .................................. 10
  1.4. The threat to the insurance brokerage firms ......................................... 11
  1.5. Insurance brokerages’ new role as aggregators .................................... 12
  1.6. Internet pure-play insurance brokerages ............................................. 13
  1.7. Revenue model for aggregators .......................................................... 13
  1.8. Motivations for insurance carriers to participate in aggregation ............ 15

2.0. Technological challenges in current implementations of web service aggregation ... 16
  2.1. Disparate implementation of the carriers’ web services ........................ 16
  2.2. Complication of the dynamic web applications .................................... 17
  2.3. Data validation challenges ................................................................. 17

3.0. Data interoperability problems in current implementations .................... 19
  3.1. Data requirement differences ............................................................ 19
  3.2. Syntactical data heterogeneity .......................................................... 19
  3.3 Data context Heterogeneity ................................................................. 20

4.0. The capabilities and limitations of XML ................................................ 22
  4.1. The capabilities of XML .................................................................... 22
  4.2. Problems that XML can’t solve ......................................................... 24
        4.2.1. Helping search engines understand the semantics of web data ....... 24
        4.2.2. Syntactical data heterogeneity ................................................. 25
              4.2.2.1. Vocabulary differences ............................................. 26
              4.2.2.2. Structural differences ............................................. 26
        4.2.3. Semantic data heterogeneity .................................................. 27

5.0. The capabilities and limitations of XSLT ................................................. 28
  5.1. Vocabulary differences ...................................................................... 28
  5.2. Structural differences ........................................................................ 28
  5.3. Semantic differences .......................................................................... 29

6.0. The capabilities and limitations of SOAP and Biztalk .............................. 30
  6.1. Overview of SOAP ........................................................................... 30
        6.1.1. Envelope ............................................................................ 30
        6.1.2. Data types ........................................................................... 31
        6.1.3. RPC .................................................................................. 31
        6.1.4. SOAP and online aggregation ............................................. 31
  6.2. Overview of Biztalk ........................................................................... 32
        6.2.1. Document identification ....................................................... 33
        6.2.2. Document receipt ............................................................... 33
6.2.3. Delivery and processing deadlines ........................................ 33
6.2.4. Attachments ................................................................. 33
6.2.5. Biztalk and online aggregation ........................................ 34

7.0. Data standardization efforts in the insurance industry ................. 35
  7.1. ACORD data standards .................................................. 35
  7.2. Problems with data standardization ................................... 35

8.0. Scalability – the problem that remains unsolved ......................... 38
  8.1. Data standardization ..................................................... 38
  8.2. XML, SOAP, and Biztalk .............................................. 38
  8.3. Scalability problem example ......................................... 38

9.0. Context Interchange (COIN) Framework .................................. 47
  9.1. Architectural overview .................................................. 47
  9.2. The process of context mediation ...................................... 48
  9.3. The domain model ....................................................... 49
  9.4. COIN and the Semantic Web ......................................... 50

10.0. COIN-Insurance Extension (IE) ......................................... 52
  10.1. Design Criteria ......................................................... 52
  10.2. Classification of new data requirements ............................. 52
  10.3. Discussion of strategies for accommodating new data requirements ..... 55
  10.6.0. COIN-IE Framework ............................................... 59
    10.6.1. The family container .............................................. 59
    10.6.2. The head sibling object ......................................... 60
    10.6.3. Family conversion axioms ...................................... 61
    10.6.4. Modifying the domain model .................................... 61
    10.6.5. Aggregator’s mappings to the domain model .................. 62
    10.6.6. Quoting services’ mappings to the domain model ............ 62
    10.6.7. Analysis of design ............................................... 63
    10.6.8. Alternative design .............................................. 64
    10.6.9. Domain model management concerns ........................... 65
    10.6.10. Managing the message size and name space .................. 65
    10.6.11. Trade-off analysis .............................................. 67
    10.6.12. Application of the new design to web service aggregation .... 68

11.0. Conclusions ........................................................................ 85

12.0. Bibliography ....................................................................... 86
List of Figures

1.1. Traditional business model of selling insurance .................................................. 10
1.2. Carriers selling direct .......................................................................................... 10
1.3. AIGdirect.com ...................................................................................................... 11
1.4. Online aggregator .................................................................................................. 12
1.5. Insure.com ............................................................................................................. 13
1.6. InsureHiTech.com .................................................................................................. 14
8.1. Scalability example – period 1 ................................................................................. 39
8.2. Scalability example – period 2 ................................................................................ 40
8.3. Scalability example – period 3 ................................................................................. 42
8.4. Scalability example – period 4 ................................................................................ 44
9.1. COIN architecture .................................................................................................. 48
9.2. Sample domain model ............................................................................................. 49
9.3. RDF graph .............................................................................................................. 50
10.1. Family container ..................................................................................................... 59
10.2. COIN-IE period N example .................................................................................. 60
10.3. COIN-IE period N+1 example .............................................................................. 60
10.4. COIN-IE domain model ........................................................................................ 61
10.5. Alternative design .................................................................................................. 64
10.6. Name space management ...................................................................................... 66
10.7. COIN-IE application - period 1 .............................................................................. 67
10.8. COIN-IE application - period 2 .............................................................................. 69
10.9. COIN-IE application - period 3 .............................................................................. 72
10.10. COIN-IE application - period 4 ............................................................................ 77
1.0. Introduction and Thesis Road Map

Internet technologies and the World Wide Web have introduced waves of disruptive changes into the market dynamics of many industries. In the finance industry, for example, the business model of full service brokerages like Merrill Lynch was seriously threatened by online discount brokerages such as eSchwab and eTrade, which allow investors to execute trades for a much lower cost. Similarly, in the insurance industry, the business model of insurance brokerage firms are threatened by the web services of the insurance carriers, which can sell insurance products directly to consumers on the Internet without any help from an insurance brokerage firm. To counter the threat to their business model, insurance brokerage firms are responding by implementing online insurance aggregators that add value to consumers by allowing them to obtain and compare insurance offerings from multiple insurance carriers. This is valuable to consumers because they no longer have to go to each insurance carrier’s web service and fill out a separate insurance application for each carrier. Chapter one of this thesis provides an in-depth discussion of how the Internet has challenged the existing business model of selling insurance and how insurance brokerage firms are fighting back by implementing aggregators.

The implementation of online insurance aggregators, however, is hindered by system incompatibility challenges and data heterogeneity challenges. The system incompatibility challenges, which are described in chapter two, arise because the web services of the different insurance carriers are implemented independently in different ways. These web services are not designed to handle the needs of the aggregator. The data heterogeneity challenges, which are described in chapter 3, arise because the data schemas of the different insurance carriers’ web services have syntactical and semantic differences. Frequently, different web services would use different vocabulary and different data structures to represent the same data elements. To make matters worse, some web services have different semantic interpretations and usage for data elements with the same name and same data structure.

Chapters four, five, six, and seven examine and evaluate the effectiveness of popular system integration and interoperability solutions, which include XML, XSLT, SOAP, Biztalk, and data standardization. XML and XSLT, which are excellent candidates for pair-wise solutions, do not scale when there are many web services and frequent changes to their data schemas. SOAP and Biztalk place heavy emphasis on document delivery, but avoid addressing data heterogeneity issues. Data standardization, which can be effective for applications where changes in data schema are rare, is not flexible enough to handle the frequent changes to the web services’ data schemas.

Chapter eight discusses the problem of scalability and adaptability to changes in data requirements over time, which XML, XSLT, SOAP, Biztalk, and data standardization do not solve effectively when they are used alone. This chapter also presents a highly detailed example showing that XML-like languages do not provide a scalable solution that is flexible enough to deal with frequent changes in data requirement over time.

Chapter nine offers an overview of the main concepts of the Context Interchange (COIN) framework, which will be used in chapter ten to create a solution for addressing the problem of scalability and adaptability for online insurance aggregators.
The COIN framework offers an excellent solution for solving data heterogeneity problems in data aggregation. During the aggregation process, data from disparate sources with different context assumptions are mapped to a common domain model by using context axioms and elevation axioms. Then conversion axioms are used to perform context mediation, and the results are delivered to the end users in the desired context.

The proposed solution in chapter ten uses a COIN-like approach to map the different insurance carriers’ data schemas to a common domain model using conversion axioms. However, the difference here is that the domain model is expected to be incomplete and is expected to change over time as new data requirements are revealed. To deal with the changes to the domain model, a special mechanism is used to manage the mappings within the domain model so that the different web services don’t have to change their mappings to the domain model when the domain model evolves. This provides the aggregator and the web services a scalable solution for adapting to changes in data requirements over time.
1.1. The Need for Web Service Aggregation in the Insurance Industry

The Internet is a revolutionary technology and platform that enables people and businesses to communicate more effectively. The Internet, however, is also a disruptive technology that can topple existing business models. New business models based on Internet technologies have created havoc in the financial brokerage industry in the past few years. Similar types of changes are dawning on the insurance brokerage industry.

The popularity of the Internet has offered a new channel of distribution for insurance carriers to bypass insurance brokerages and sell products directly to their customers online. This new business model poses a serious threat to the traditional insurance brokerage firms. To protect their business, many insurance brokerage firms are assuming the new role as aggregators for insurance products on the Internet.

1.2. The Traditional Business Model of Selling Insurance

Prior to the arrival of the Internet, the traditional business model of selling insurance through insurance brokerage firms was beneficial for both the insurance carriers and the consumers.

By outsourcing the role of selling insurance to the national network of insurance brokerage firms, insurance carriers realize significant cost savings and are able to focus on their core competencies. Without the brokerage firms, each insurance carrier would need to set up a large number of sales offices to reach consumers in different geographical locations across the nation. This is extremely costly. In addition, each carrier would need to train and maintain a huge sales force, which is also a large expenditure. By outsourcing the sales function to the insurance brokerage firms and only paying them when there is a sale, the insurance carriers can reach a huge customer base without incurring astronomical expenses in marketing.

Insurance brokerage firms also provide significant benefits to the consumer. Without these firms, the consumer would need to visit the sales office of each insurance carrier to do price comparison. This process requires a great amount of effort, and most consumers would probably not have the time to visit more than two or three insurance carriers. Under this scenario, the consumer is most likely to just visit the sales office of the nearest or the most well known insurance carrier. The high search cost prevents the consumer from getting the best price and the best insurance package. With the help of the insurance brokerage firms, the search cost is reduced significantly for the consumer. The consumer only have to visit one insurance brokerage firm, and that firm helps the consumer obtain quotes on insurance policy premium from many insurance carriers. In addition, the brokerage firm helps the consumer compare the insurance packages and pick the package that suits the needs of that particular consumer the most. As a result, the brokerage is quite valuable to the consumer. Figure 1.1 on the next page illustrates the traditional business model for selling insurance, where the network of insurance brokerage firms help to facilitate the selling of insurance products from the insurance carriers to the consumers.
However, this model also has its flaws because insurance brokerage firms do not always act in the best interest of the customers. For example, the brokerage may be inclined to recommend the carrier that pays the brokerage the highest commission. In addition, each brokerage firm only works with a limited number of insurance carriers. Thus the customer will not be able to obtain insurance packages from carriers who don’t work with the brokerage that he is using.

### 1.3. Insurance Carriers Selling Direct on the Internet

The arrival of the Internet has created new opportunities and a new channel of distribution for insurance carriers. These firms are no longer limited by the geographical barriers because their web services can just as easily reach customers in Alaska as customers in New York. The insurance carriers only have to set up a single web service on the Internet, and they can reach customers all across the nation. The need for intermediaries and brokerage firms is greatly diminished.

Realizing the potential of the Internet, insurance carriers such as AIG and Geico are creating web services that allow consumers to buy insurance from them directly online without paying a commission to any insurance brokerage firm. By selling insurance products directly to consumers, the insurance carriers can increase their profit by capturing the 15% commission on the insurance premium that would have been given to a brokerage firm under the traditional business model. In addition, insurance carriers can increase their competitiveness by giving away a part of that 15% commission to the consumer in the form of a lowered insurance premium. As a result, there is strong incentive for insurance carriers to sell directly to their customers online. Figure 1.2 below further illustrates this new business model for insurance carriers.
1.4. The Threat to the Insurance Brokerage Firms

Under the traditional business model, insurance brokerage firms get a commission that is up to 15% of the insurance premium when an insurance policy is sold through the brokerage to a new customer. The customer account is owned by both the brokerage firm and the insurance carrier. Every time the customer renews the insurance policy, the brokerage firm gets another commission. Thus each customer account provides a continuous stream of cash flow for the insurance brokerage firm.

Under the new business model of carriers selling directly to the customers online, the customer account is solely owned by the insurance carrier. The brokerage firm plays no role in this new business model, and its revenue stream is zero. If more and more customers purchase insurance directly from the carriers, then the brokerage firms would lose market share, and their revenue would fall sharply. This process has the potential of wiping out the brokerage firms in the insurance industry if these firms don’t react quickly and appropriately.

One immediate measure that the brokerages can take to discourage the insurance carriers from selling direct is by threatening to stop doing business with carriers who are selling direct. This approach may work if a small number of carriers are trying to sell direct. However, if all the carriers are trying to sell direct, then this approach will be less effective.

Figure 1.3. Screen shot of AIGDirect.com, the web store for the carrier AIG.
1.5. Insurance Brokerages’ New Role as Aggregators

To protect their business, many insurance brokerage firms started to implement an online version of their business. In addition to providing information and advice on insurance products in their web services, these firms also assumed the role as aggregators for the quoting services of the carriers. Figure 1.4 below illustrates the business model for aggregators. As an aggregator, the brokerage firm allows the consumer to fill out a single insurance application on its web service, and that data is sent to the quoting web services of the participating carriers. Within seconds, the carriers’ web quoting services would respond with quotes on the premium that they charge, and the consumer can view and compare the quotes on the aggregator’s web site. By being aggregators, the brokerage firms are offering the consumers a valuable service. The consumer no longer have to go to each insurance carrier’s web service and fill out a separate insurance application for each carrier. The consumer only have to fill out a single insurance application at the aggregator’s web site. The search cost is reduced significantly, and the consumer saves both time and effort in his online insurance shopping experience. In addition, under this new service model, the consumer no longer have to wait for days for the insurance carriers to respond with a quote. Instead, it only takes a few seconds for them to receive quotes on the premium charged by the different insurance carriers. Insweb.com, Insure.com, InsureHiTech.com, and Bulwarkco.com are examples of online aggregators for the insurance industry. All these firms are still in the development phase of their aggregator implementation.

![Figure 1.4. The insurance brokerage firm’s new role as an online aggregator.](image)

In many ways, insurance brokerages are already aggregators before the arrival of the Internet. Under the traditional service model, a client would fill out the insurance application for each carrier at the brokerage’s sales office. To streamline this process, the brokerage may create an application that asks for the union of the data needed by each insurance carrier. After collecting the data on paper, the insurance application is either mailed or faxed to the insurance carriers. After reviewing the application, the carriers would mail or fax their response back to the brokerage. This process usually takes about three to ten business days. For online aggregators, on the other hand, this process is much faster. The cycle time can be as low as a few seconds.
1.6. Internet Pure-play Insurance Brokerages

A new type of insurance brokerage firms are emerging, and these are the Internet pure-play insurance brokerages. These firms don’t have a brick and mortar presence to work with customers. Instead, they work with customers solely through the Internet. The web services of these firms provide general information and advice to consumers on how to select insurance products. In addition, they allow consumers to fill out a single insurance application and get quotes from many participating insurance carriers. These firms are only paid when an insurance policy is sold through their web services. Thus the ability of these firms to successfully implement an aggregator service is even more crucial for survival. Examples of Internet pure-play brokerages include Insure.com and IntelliQuote.com. IntelliQuote.com is currently working with 200 different insurance carriers.

1.7. Revenue Models for Aggregators

Because the business model for selling insurance on the Internet still needs time to mature, the revenue model for the aggregators differs from one brokerage firm to another. The brick and mortar brokerages want to charge the same commission in their online services as in their traditional services; charging a lower commission online could hurt their brick and mortar business. In addition, these firms want to continue to share the customer account with the insurance carrier, and they want another commission every time the customer renews the insurance policy.

Figure 1.5. An Internet pure-play aggregator, Insure.com.
On the other end of the spectrum, some Internet pure-play brokerages only charge a one-time fee that is 1% to 5% of the insurance premium, and the customer account is owned solely by the insurance carrier. This is a much lower fee than the 15% commission charged by most traditional brokerages, and it could offer these new brokerages a competitive advantage. However, whether this new revenue model is sustainable or brings in enough cash flow to cover the brokerage’s expenses is still a question that needs to be answered.

Between these two extremes are brokerages that charge a commission of 1% to 5% but maintain partial ownership of the customer account. These brokerages get a commission every time the customer renews the insurance policy just like the traditional brokerages.

The three revenue models mentioned above are all being tested by the insurance aggregators. Which model will prevail is an issue that will become clear over time. This section is only intended to provide an introduction into the economics of online insurance aggregation. A thorough analysis of the economic issues is beyond the scope of this thesis.

![Figure 1.6](https://www.insurehi-tech.com/app/isi/index/aboutus/figure.png)

Figure 1.6. The online aggregator InsureHiTech.com, which has a brick & mortar presence for 128 years as Maloy Insurance in New Jersey.
1.8. Motivations for Insurance Carriers to Participate in Aggregation

Insurance carriers don’t like the price comparison feature of aggregators. Nevertheless, many insurance carriers are working with aggregators for several reasons. The aggregators serve as an extra channel of distribution and bring them new customers. In addition, insurance carriers are accustomed to the price comparison services of traditional brokerages already, and this is something that they know how to cope with. Finally, if a carrier does not work with the aggregator but the carrier’s competitors do, then the carrier could lose potential new customers to the competitors. As a result, many insurance carriers are actively participating in web service aggregation efforts with the brokerage firms. Insweb.com, for example, is working with over forty different insurance carriers ranging from AIG to The Hartford.
2.0. Technological Challenges in Current Implementations of Web Service Aggregation

While the business concepts for web service aggregation are straightforward and simple, the implementation and technological aspects of web service aggregation are complicated. The problem is that the web services of each carrier are implemented independently, and they could be implemented on different platforms using incompatible software technologies. In addition, the data requirement of each web service could be different, and the structure of the data for each web service could also be different. Worst of all, these web services could have data elements with the same name and structure but different context and meaning. As a result, the implementation of web service aggregation is a formidable challenge. The following sections will further explore these issues.

2.1. Disparate Implementation of the Carriers’ Web Services

The quoting web services of the insurance carriers are consisted of two parts: front-end user interface and back-end data processing applications. The front-end is usually an insurance application in HTML format, and frequently Javascript is embedded in the HTML to perform client-side data validation. Customer data is transmitted from the front-end to the back-end via the HTTP POST method. The back-end, which is housed in the carrier’s web servers, is consisted of applications that process the customer’s insurance application data and calculate the premium for the insurance policy. These applications are frequently call the rating engines.

Because the insurance carriers completed their implementations independently, their rating engines could be implemented using different software technologies on different platforms. For example, consider the following scenario: Web service A is implemented as Active Server Pages for the IIS Server on the Windows platform; Web service B is implemented as a Perl CGI program that runs on a Linux server; Web service C is implemented as Java Servlets on the Netscape Application Server on the Solaris Unix platform. Under traditional client server computing paradigms, the task of having the aggregator interoperating with all these disparate systems would be a nightmare. Even for applications that are implemented in the same language for the same platform, interoperability is a formidable task. For applications that are implemented in different languages for different platforms, interoperating would almost be an impossible task.

Fortunately, all these web services have one thing in common. They all communicate with the user’s web browser through the HTTP protocol. Using the common HTTP protocol, these web services can send messages to each other. These messages may include data or service requests. As a result, communication with these disparate systems becomes a feasible task for the aggregator.

For quoting services that present the insurance application on a single HTML page, the task of sending data to the quoting service is relatively easy for the aggregator. The aggregator simply mimics the web browser and sends all the customer data at once using a single HTTP POST
request. The quoting service would process the aggregator’s data as if it came from a customer’s web browser.

For quoting services that split an insurance application into multiple HTML pages to increase the usability of the user interface, the task becomes a bit more difficult to coordinate for the aggregator. The aggregator would need to divide the customer data into chunks where each chunk of data corresponds to the answers to a single page of questions. The data would need to be sent via multiple HTTP POST requests. This is more cumbersome, but it is still manageable.

### 2.2. Complication of the Dynamic Web Applications

To enhance the usability of their quoting services, some carriers componentized their insurance application into many dynamic web pages. Instead of having one long form on a single HTML page, which could discourage customers from completing the insurance application, the carriers only ask two or three fundamental and non-invasive questions in the first HTML page. Based on the customer’s answers to the previous questions, the next sets of questions are generated dynamically; questions that don’t apply to this particular customer would never be asked. This type of user interface is much more friendly and appealing to the customer.

From the perspective of the aggregator, these dynamic quoting services are the hardest to work with. Because the set of questions generated for each customer could be different at each stage, the aggregator wouldn’t know what data to submit to the quoting service at each stage. The only way for the aggregator to work with these carriers is for the carriers to implement a separate web interface to work with the aggregator. This could be a simple task if the carriers used a loose-coupling approach and separated the user interface from the business logic and data processing. However, many carriers used a tight-coupling approach that integrates some of the data processing and business logic into the user interface to increase usability and performance. Implementing a separate interface for the aggregator would be much more difficult for these carriers. There exist methods that these carriers can use to decouple the business logic from the presentation layer of their systems, but a discussion of these approaches is beyond the scope of this thesis.

### 2.3. Data Validation Challenges

To ensure the correctness of the incoming data, most carriers incorporated data validation into their web services. Many carriers embedded the data validation code into the user interface in the form of Javascript code. This allows the data validation to be carried out rapidly on the client machine without any network latency, and the user is prompted to make corrections when errors are found. Alternatively, some carriers wanted to separate the data validation from the user interface, and they perform data validation at the server end. While the second approach is superior from a software engineering perspective, it is not that user friendly. Users get impatient when they have to wait for ten seconds or more for the server to respond with a message indicating that they made an error on one of the data fields. The user’s tolerance for network delay is quite low. A discouraged user will often not complete the form, and the carrier loses a potential customer.
From the perspective of the aggregator, the trade-offs are completely different. Client-side data validation, which is good for the user, creates a disadvantage for the aggregator. The aggregator can’t benefit from the carriers’ client-side data validation code because the aggregator has a separate web interface for the user. As a result, the burden of doing data validation is shifted to the aggregator, and it has to make sure that the data that it sends to the quoting services is error free. Otherwise it will get an error message or an erroneous quote from the quoting service.
3.0. Data Interoperability Problems in Current Implementations

The problem of data heterogeneity for web service aggregation can be classified into three categories. These are: data requirement differences, data syntax differences, and data context differences.

3.1. Data Requirement Differences

When insurance carriers calculate the premium for a particular insurance policy, they need to assess the risk and the probability of the customer filing a claim. To measure this risk, insurance carriers have independently developed algorithms for measuring the risk of each insurance product. Carriers who measure risk more accurately will suffer less losses in claims. As a result, these algorithms are guarded as company secrets, and the carriers refine their algorithms over time independently. Because each carrier measures risk in a different way, the data that they use to measure risk are likely to have some differences. As a result, the questions on the insurance application of each carrier are likely to be different in some aspects also.

The challenge to the aggregator is how to satisfy the disparate data requirements of these carriers. Currently, aggregators are solving this problem by looking at the insurance application of these carriers to identify data elements that they have in common and data elements that are unique to each carrier. The data elements in common are merged, and the carrier-specific elements are separated. The end result is that the data elements used by the aggregator’s insurance application is the union of the data elements of all the participating carriers. This approach ensures that the data requirements of all the carriers are satisfied. The disadvantage of using this approach is that the aggregator’s insurance application may get very lengthy. Nevertheless, this approach is still superior to forcing the customer to fill out a separate insurance application for each carrier.

3.2. Syntactical Data Heterogeneity

The web interface of most carriers expects a pre-defined data stream to come from the customer’s web browser. After all, data sent through the web browser using the HTTP POST method are in delimited text format, where data and data field names are separated by “=”, “+”, and “;” symbols. Thus the quoting web services rely heavily on these delimiters and the position of the data field names to properly extract the relevant data from the data stream. The extracted data is then converted into data objects that the data processing and quoting applications know how to work with.

Since the quoting services can only handle data in a pre-defined format, the aggregator must deal with each of these pre-defined formats to successfully interoperate with these quoting services. The good news is that all these quoting services are using the HTTP POST method; thus the delimiters are the same for all the data formats. The problem is that different carriers could be using different data field names for the same data field. For example, for the data field “customer name”, one carrier could use the field name “clientName” while another uses the field name “customerName”. To complicate the matter, another carrier could split it into three tags: “customerFirstName”, “customerMiddleName”, and “customerLastName”. For platforms such
as Unix, it matters whether the characters are upper case or lower case. Thus “customernme” will be treated differently from “CustomerName”.

Dealing with these syntax differences is a challenge for the aggregator. To work with these carriers’ quoting services, the aggregator must keep track of which data field name is being used by which quoting services, and it has to prepare a different data format for each quoting service. This approach works for aggregating a small number of quoting services where the number of data fields is small. However, this approach is not scalable. If the aggregator wishes to work with many insurance products from many insurance carriers, then this can get problematic. For N insurance carriers and M insurance products, the aggregator would have to maintain M x N pair-wise data format solutions. This is analogous to the situation that plagued traditional Electronic Data Interchange (EDI) implementations. For N businesses to interoperate, traditional EDI requires N x N pair-wise solutions, and this approach is not scalable [GP 2000].

Chapter ten of this thesis will propose a solution for this problem using concepts from the Context Interchange Framework.

3.3. Data Context Heterogeneity

Because the quoting services of the insurance carriers are developed independently, it is possible for the carriers to have the same name for a data field but different context assumptions about the data. For example, we have two insurance applications for worker’s compensation insurance, and both of them have the data field “netIncome”. For application A, “netIncome” is expected to be the net income before taxes. For application B, “netIncome” is expected to be the net income after taxes. If the aggregator neglects the context assumptions and assumes “netIncome” to refer to the same quantity for both carriers, then one of the carriers will return a wrong quote for the premium on the worker’s compensation policy.

Another serious source of data heterogeneity is that different carriers ask for data in different granularity of detail. Consider the following scenario.

**Carrier 1**

Data fields:
- totalNumberOfWorkers
- numberOfUnionWorkers
- numberOfNonUnionWorkers

Context Assumptions:
- totalNumberOfWorkers = numberOfUnionWorkers + numberOfNonUnionWorkers
  All workers are full time workers.

**Carrier 2**

Data fields:
- totalNumberOfWorkers
- numberOfFulltimeWorkers
  numberOfParttimeWorkers
Context Assumption:
\[
\text{totalNumberOfWorkers} = \text{numberOfFulltimeWorkers} + \text{numberOfParttimeWorkers}
\]

The question for this scenario is whether “totalNumberOfWorkers” refers to the same quantity for the two carriers. When the context assumption that carrier A is only interested in full time workers is revealed, we know that carrier A and carrier B have different definitions for the term “totalNumberOfWorkers”. Without this context assumption being clearly stated, the aggregator would have no way of knowing whether the two carriers have the same meaning for the term “totalNumberOfWorkers”. If the aggregator assumes the term to have the same meaning for both carriers, then one of the carriers would receive the wrong data from the aggregator, and the calculated premium from that carrier would be wrong.
4.0. The Capabilities and Limitations of XML

Extensible Markup Language (XML) is a simple language that enables meta-data to be added to data. The computing industry has high hopes for XML, and it believes that XML will revolutionize the Internet the same way HTML did [GP 2000]. There are many claims that XML will help to bring about the semantic web and that XML will solve the system interoperability problems of commercial firms. However, is XML truly a silver bullet that can solve everyone’s interoperability problem? Will XML solve the data heterogeneity problems for the insurance aggregators? This chapter of the thesis will provide an overview of the capabilities and the limitations of XML.

4.1. The Capabilities of XML

XML is a useful text-based markup language that can add meta-data to web documents to facilitate the processing of these documents by computer software. For example, a user is using a search engine to look for articles that are written by Tim Berners-Lee, and one of the web sites has the following article in HTML format.

```html
<html>
<body>
<h1>Resource Description Framework</h1>
<h2>Ralph Swick</h2>
<p>
Resource Description Framework is an important component of the semantic web envisioned by Tim Berners-Lee.
...
</p>
</body>
</html>
```

This article is definitely not something that the user is looking for. However, this article is likely to be returned in the query result set because it contains the key words “Tim Berners-Lee”. On the other hand, if the same document is presented in the following XML format, then the processing of the data will be easier for the search engine.

```xml
<xml>
<title>Resource Description Framework</title>
<author>Ralph Swick</author>
<content>
Resource Description Framework is an important component of the semantic web envisioned by Tim Berners-Lee.
...
</content>
</xml>
```
If the search engine has been trained to recognize XML tags such as `<author>`, it should recognize that Tim Berners-Lee is not the author of this article.

Another important use of XML is for messaging between software systems through the Internet. Prior to the arrival of XML, many companies were using traditional EDI approaches for communication between their systems. First these companies have to define EDI data standards, which were frequently in binary or hexadecimal format. After defining the data standards, these companies had to create proprietary software that can encode and decode messages in these formats. The proprietary software needed for encoding and decoding EDI messages are expensive and time consuming to both develop and maintain. With the help of XML, messaging between systems will become much easier. There are three main reasons. The first is that XML messages are in text format. This makes it easier for the developers to work with XML messages. The second reason is that good XML parsers are currently freely available from IBM, Microsoft, and many other large software companies. Finally, systems that use XML for messaging through the Internet don’t need to use expensive private data networks. This can save firms a great amount of money because they can leverage existing Internet infrastructure, the HTTP protocol, and their existing web clients and server components to lower application development costs.

Theoretically speaking, HTML can also be used as a messaging format between systems, even though it is rarely being used for this purpose. Consider the next example.

A system on the aggregator’s server needs to send a message to a system on the carrier’s server, and the message is formatted using HTML as shown below.

```html
<html>
<body>
<table>
<tr><td> customer name </td> <td> John Smith </td> </tr>
<tr><td> product </td> <td> automobile insurance </td> </tr>
<tr><td> age </td> <td> 65 </td> </tr>
<tr><td> state </td> <td> New York </td> </tr>
</table>
</body>
</html>
```

To extract the data it needs, the carrier’s system could do text search to identify field names such as “age” or “state”. Then it can skip the next two tags “</td><td>” to read in the value of the field until it reaches the “</td>” tag. This method works, but it is not an elegant solution. This type of data extraction code can also be difficult to maintain. If the same data is encoded in the following XML format, then the data extraction will be much easier.

```xml
<xml>
<customerName> John Smith </customerName>
<product> automobile insurance </product>
<age> 65 </age>
<state> New York </state>
</xml>
```
Upon receiving the XML message, the carrier can use a XML parser to create a document object model (DOM) tree. Then it can look at each node of the tree to identify the tag names and extract the relevant values. This approach for processing messages is much more elegant, and it is less error-prone than the HTML approach. The code for processing the XML messages is also easier to maintain.

4.2. Problems that XML Can’t Solve

While XML enables meta-data to be added to data, this does not guarantee that software systems will be able to understand the semantics of the data. In addition, even when XML is used as the messaging format for communication between systems, the problem of syntactical and semantic incompatibilities between the data of the different systems does not go away.

4.2.1. Helping Search Engines Understand the Semantics of Web Data

Using XML to add meta-data to data does not guarantee that the software systems reading this data will understand the semantics of the data. The following example will show that XML is not too much better than HTML in helping programs understand the semantics of the data.

A user is using a search engine to look for web sites that sell SONY MP3 players for less than $275, and one of the web sites have the following data in HTML format.

```
<html>
<body>
<table>
<tr><td> MP3 Player </td></tr>
<tr><td>SONY </td><td>300</td></tr>
<tr><td>Creative</td><td>275</td></tr>
<tr><td>Diamond</td><td>225</td></tr>
</table>
</body>
</html>
```

Because the key words “MP3 Player”, “SONY”, and “275” appear in the web page, it is very likely that the search engine will return this web page in the query result set even though it does not have what the user is looking for. Now the question is what if the same data above is expressed in the following XML format?

```
<xml>
  <item>
    <type> MP3 Player </type>
    <brand> SONY </brand>
    <itemPrice> 300 </itemPrice>
  </item>
</xml>
```
Will a search engine be able to find what the user wants? The search engine will likely to include this page in the result set because it has the key words “MP3 Player”, “SONY”, and “275”. However, how will the search engine understand that the price of the SONY MP3 player is 300? If it is trained to look for the <price> tag, it will fail because the price is included in the <itemPrice> tag; there is no way that the search engine can know ahead of time all the different tags that can be used to represent price.

The only way for the search engine to successfully fulfill this request is if it has software components that are hard-wired to recognize this particular data schema. The hard-wired code will be able to recognize that the number between the <itemPrice> tag is the price of the SONY MP3 player and that it is not less than 275.

The problem with this approach is that there can be infinite number of data schemas on the World Wide Web, and there is no way that a search engine can be hardwired to recognize all of them. In addition, when the XML data schema changes, the code that is hard-wired to recognize this XML data schema will no longer work. As a result, simply using XML to add meta-data to data does not guarantee that software systems will be able to understand the semantics of the data.

Currently, the World Wide Web Consortium (W3C) is trying to solve this problem by creating standard XML schemas. Each industrial sector would have their own standard XML schema, and if every firm uses the standard XML schema for its industry, then the above problem can be greatly alleviated. For example, if all the electronic retail shops on the Internet use the <price> tag for the price of items instead of some other proprietary tag, then it will become much easier for a shop-bot to find what it is looking for. Even though the shop-bot may not truly understand the semantics of the data, it can at least identify the XML tag that it is looking for and produce more relevant result sets for the user.

### 4.2.2. Syntactic data heterogeneity

There are two types of syntactic data heterogeneity issues that can’t be easily solved even when XML is used to represent application data. The first problem is differences in data vocabulary. The second problem is differences in the structure of the data element and differences in the granularity of detail provided by the data elements.
4.2.2.1. Vocabulary differences

System interoperability problems can arise when two systems use different vocabularies to represent the same data elements. Consider the following example.

Brokerage A’s system and carrier A’s system use the data element <customerName> in XML messages transmitted between the two systems.
Brokerage B’s system and carrier B’s system use the data element <clientName> in XML messages transmitted between the two systems.

Now brokerage A wants to work with both carrier A and carrier B. Brokerage A’s system is currently able to extract the data element <customerName> from XML messages. However, brokerage A’s system does not have any code for recognizing the element <clientName> even though <customerName> and <clientName> are semantically equivalent. Unless brokerage A manually add code to recognize the element <clientName>, its system will not be able to handle messages from carrier B that contain this element.

This example illustrated that using XML does not automatically solve the problem of data vocabulary differences in system integration.

4.2.2.2. Structural differences

When two systems use different data structures to represent the same data elements, the sharing of these data elements between the two systems can be a problem. Consider the following example.

Brokerage A’s system and carrier A’s system use the data element <address> in XML messages transmitted between the two systems.
Brokerage B’s system and carrier B’s system use the data element <address> <street> <city> <state> <zipcode> </address> in XML messages transmitted between the two systems.

Now carrier A wants to work with both brokerage A and brokerage B. Carrier A’s system currently have code for extracting the <address> element from brokerage A, but it doesn’t have any code for extracting the more detailed <address> element from brokerage B. To solve this problem, carrier A would need to manually add code to its system to recognize brokerage B’s <address> element and concatenate the values from the <street>, <city>, <state>, and <zipcode> sub-elements into a single string value, which is what carrier A’s system needs.

Similarly, if carrier B wants to work with both brokerage A and brokerage B, then there will be a problem, and it is actually worse in this case. Carrier B needs the address data to be broken down into <street>, <city>, <state>, and <zipcode>. Brokerage A, however, provide all that information in a single string value inside its <address> element. Theoretically speaking, carrier B can use the comma delimiter to identify the different sub-elements in brokerage A’s <address> element. However, if the use of commas is inconsistent within brokerage A’s <address> element, then carrier B will have a hard time processing the values from brokerage A.
4.2.3. Semantic data heterogeneity

While syntactic differences between the data of two systems can be easily identified, semantic differences between the data of the two systems can be more difficult to identify and correct. If two systems have different semantic interpretations for the same data and these differences are not reconciled, then serious errors can result when the two systems share data. Consider the following example.

Brokerage A’s system and carrier A’s system use the data element <assetValue> in their XML messages, and they assume that <assetValue> refers to the book value of the asset.
Brokerage B’s system and carrier B’s system use the data element <assetValue> in their XML messages, and they assume that <assetValue> refers to the market value of the asset.

Now carrier A wants to work with both brokerage A and brokerage B. Because both brokerage A and brokerage B are using the same XML tag <assetValue>, carrier A may not realize that brokerage B is using the market value instead of the book value of the asset. Carrier A’s system would definitely not be able to catch this problem because it has no understanding of the semantics of the data; it has only been coded to look for the <assetValue> tag. If the semantic differences are not detected and not reconciled, then carrier A will calculate the wrong insurance premium for brokerage B’s customers. This can cause either carrier A or brokerage B’s customers to lose money.

While the capabilities of XML are limited when it is used alone, the power of XML can be increased by using it with other technologies such as the Extensible Stylesheet Language – Transformations (XSLT). The next chapter will discuss how XML can be used with XSLT to solve some of the problems mentioned in this chapter.
5.0. The Capabilities and Limitations of XSLT

Extensible Stylesheet Language – Transformations (XSLT) is a language designed for transforming XML data from one format to another format. The input of XSLT programs is XML data. The output of XSLT programs can be XML data in another format; it can also be HTML or data in another format [KM 2000].

XSLT can be used to solve many of the problems mentioned in chapter three and chapter five. This section will illustrate how XSLT can be used to solve some of these problems.

5.1. Vocabulary differences

If two systems use different vocabulary for the same data element, XSLT can be used to reconcile these differences. In the example in section 4.2.2.1, carrier B’s system can only recognize the tag <clientName>, but brokerage A’s system uses the tag <customerName>. When brokerage A sends data to carrier B, it can use the following XSLT code to convert the <customerName> tag to the <clientName> tag.

```xml
<xsl:template match = "customerName"/>
  <clientName>
    <xsl:value-of select="."/>
  </clientName>
</xsl:template>
```

Using a similar approach, the syntactic data heterogeneity problems mentioned in section 3.2. can be solved using XSLT if the data being shared is coded in XML format. When the aggregator sends data to the different carriers, it can use a different XSLT template for each carrier to transform XML data from its internal format to the format of the corresponding carriers.

While XSLT provide a good solution for resolving vocabulary differences, there is a drawback. If the aggregator changes its data schema, then it would need to change the XSLT template for everyone of the carriers that it works with. For aggregators such as Intelliquote.com, which works with over 200 carriers, changing 200 XSLT templates can be quite a bit of work. This issue will be further explored in chapter eight.

5.2. Structural differences

When two systems use different data structures to represent the same data element, XSLT can sometimes be used to reconcile the structural differences. In the example in section 4.2.2.2, brokerage B uses the element <address><street><city><state><zipcode></address> to store a customer’s address while carrier A uses the <address> element to store a customer’s address in one single string. When sending data to carrier A, brokerage B can use the following XSLT code to produce the data in the format that carrier A can process.
However, if carrier A wants to send address data to brokerage B, then it is not obvious how XSLT can be used to break the single string value in <address> into the <street>, <city>, <state>, and <zipcode> components. Some other approaches such as writing a Java program are needed to solve this particular problem.

5.3. Semantic differences

XSLT code does search and replace only. It has no understanding of the semantics of the data that it processes. In the example in section 4.2.3, where <assetValue> in system A refers to book value while <assetValue> in system B refers to market value, XSLT can’t help at all. Because the market value of an asset can fluctuate randomly relative to the book value, there is no formula for converting one to the other. Without a conversion rule, XSLT can’t help at all.

For the data context heterogeneity problems mentioned in section 3.3, XSLT can provide a solution if the if the conversion rules for mapping the aggregator’s data to the carriers’ data are known. To solve the context problems, developers from the aggregator would need to talk to developers from the carriers to identify the context differences between their data. After manually identifying these differences, then the aggregator’s developers can create conversion rules and code them in XSLT format. These XSLT templates will perform the necessary conversion to make sure that “totalNumberOfWorkers” from carrier A only includes full time workers while “totalNumberOfWorkers” for carrier B includes both fulltime and part-time workers.

While XSLT can be used to solve the context problems manually, it does not mean that XSLT programs can understand the context of the data. It is the human developers who understand the context and the semantics of the data, and the XSLT program is merely an application of their knowledge in software form.
6.0. The Capabilities and Limitations of SOAP and Biztalk

There has been a great amount of excitement around Microsoft’s Biztalk in the computing industry. Microsoft claimed that Biztalk will enable companies to effortlessly integrate their systems and solve their system interoperability problems. However, Biztalk is an extension of Simple Object Access Protocol (SOAP), and the SOAP community have not make such claims. Are Microsoft’s claims just hype or is Biztalk truly a silver bullet solution? In addition, how is SOAP and Biztalk different from XML and can they solve the problems that XML can’t? Finally, can Biztalk and SOAP solve the interoperability problems between the aggregator and the carriers? This section will attempt to answer these questions.

6.1. Overview of SOAP

Simple Object Access Protocol (SOAP) is a lightweight protocol developed by a consortium of software firms for the exchange of information between systems in a distributed environment. SOAP, which is encoded using XML, is consisted of three main components. The first component is the message envelope, which describes what is in the message and how to process the message. The second component is a set of encoding rules for expressing data types. The third component is a convention for representing remote procedure calls (RPC) and responses.

A sample SOAP message from [SOAP 2000] is provided below. This message contains a request for the last traded price of Disney’s stock.

```xml
<SOAP-ENV:Envelope
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
 <SOAP-ENV:Body>
 <m:GetLastTradePrice xmlns:m="Some-URI">
  <symbol>DIS</symbol>
 </m:GetLastTradePrice>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

6.1.1. Envelope

The envelope component of a SOAP message has three elements. The first is the “Envelope” element, which is mandatory for the message. This element may contain namespace declarations as well as additional attributes. The second element is “Header”, which is optional. This element may contain message extensions for authentication, transaction management, payment, or other purposes. Two important attributes for header entries are the “actor” attribute and the “mustUnderstand” attribute. The “actor” attribute is used to indicate the recipient of a header element. The “mustUnderstand” attribute indicates whether a header element is mandatory or optional for a recipient to process. The third element in the envelope component is “Body”, which is mandatory for the message. The “Body” element provides a mechanism for exchanging mandatory information with the ultimate recipient of the message. This element is typically used for marshalling RPC calls and error reporting.
6.1.2. Datatypes

SOAP provides encoding specifications for many datatypes. In some sense, this makes SOAP look like a part of a programming language. The specified SOAP datatypes are listed below.

- Simple types
  - String
  - Enumeration
  - Array of bytes
- Polymorphic accessors
- Compound types
  - Compound values
  - Structs
  - References to values
- Arrays
  - Partially transmitted arrays
  - Sparse arrays
- Generic compound types

6.1.3. RPC

One of the design goals of SOAP is to enable distributed systems to exchange RPC calls. RPC calls and responses are both carried in the SOAP “Body” element. To make a RPC call, the following information should be provided:
  - The Universal Resource Identifier (URI) of the target object
  - A method name
  - An optional method signature
  - The parameters to the method
  - Optional header data

6.1.4. SOAP and online aggregation

Theoretically speaking, SOAP can be used for communication between the aggregator’s systems and the carrier’s system. However, SOAP only provides a message envelope; the content of the message is entirely in XML. As a result, SOAP can do no more to solve syntactic and semantic data heterogeneity problems than XML. To be fair, SOAP should not be criticized because the SOAP community have not made any outrageous claims about SOAP’s ability to solve data heterogeneity problems. SOAP was designed for transporting messages between systems, and it serves that function well.
6.2. Overview of Biztalk

Biztalk is an extension of SOAP 1.1 by Microsoft. In some sense, it can be view as Microsoft’s version of SOAP with a few enhancements in document routing and delivery. The major enhancements include:
- Document identification
- Document receipts
- Delivery and processing deadlines
- Attachment handling

A sample Biztalk document from [BIZ 2000] is provided below. This document is a purchase order for the book “Essential Biztalk”. The destination of this document is the “Book Order Department” of “Booklovers Anonymous”.

```xml
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance">
  <SOAP-ENV:Header>
    <dlv:delivery SOAP-ENV:mustUnderstand="1"
      xmlns:dlv="http://schemas.biztalk.org/btf-2-0/delivery"
      xmlns:agr="http://www.trading-agreements.org/types/">
      <dlv:to>
        <dlv:address xsi:type="agr:department">Book Order Department</dlv:address>
      </dlv:to>
      <dlv:from>
        <dlv:address xsi:type="agr:organization">Booklovers Anonymous</dlv:address>
      </dlv:from>
    </dlv:delivery>
    <prop:properties SOAP-ENV:mustUnderstand="1"
      xmlns:prop="http://schemas.biztalk.org/btf-2-0/properties">
      <prop:identity>uuid:74b9f5d0-33fb-4a81-b02b-5b760641c1d6</prop:identity>
      <prop:sentAt>2000-05-14T03:00:00+08:00</prop:sentAt>
      <prop:expiresAt>2000-05-15T04:00:00+08:00</prop:expiresAt>
      <prop:topic>http://electrocommerce.org/purchase_order/</prop:topic>
    </prop:properties>
  </SOAP-ENV:Header>
  <SOAP-ENV:Body>
    <po:PurchaseOrder xmlns:po="http://electrocommerce.org/purchase_order/"
      po:Title="Essential BizTalk"/>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```
6.2.1. Document identification

Biztalk uses the <properties> tag to store document identification information as a Biztalk specific header entry in a SOAP message. Within the <properties> tag are four mandatory elements: <Identity>, <sentAt>, <expiresAt>, and <topic>. Both <sentAt> and <expiresAt> are time instances. <identity> is an Universal Resource Identifier (URI) reference that uniquely identifies the Biztalk document for purposes such as logging, trading, or error handling. <topic> is also a URI reference.

6.2.2. Document receipt

The document receipt is another Biztalk-specific header entry in a SOAP message. It contains an universally unique identifier of the Document and a time stamp for the time at which the document is received at the destination.

6.2.3. Delivery and processing deadlines

Biztalk has two types of deadline elements. The first is the delivery deadline represented by the tag <receiptRequiredBy>. If the document is not delivered to the destination by the time specified by <receiptRequiredBy>, then this document may no longer be accepted because it may contain out of date information. The second deadline element is the processing deadline, which is specified by the <expiresAt> tag. If the processing of the document is not completed by the time specified by <expiresAt>, then the document is considered to be null and void.

6.2.4. Attachments

To support attachment handling, Biztalk uses the <manifest> header entity to store attachment information. Inside the <manifest> element are <reference> elements that provide the URI of the attachment and a description of the attachment.

Provided below is an example of a Biztalk document with two attachments, which is also from [BIZ 2000]. This document is a completed insurance claims form. One of the attachments is a signed claims document. The other attachment is a photo of a damaged car.

```xml
<?xml version='1.0'?>
<SOAP-ENV:Envelope
   xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:SOAP-ENV:Header> <!-- delivery and properties header entries omitted for brevity -->
  <manifest xmlns="http://schemas.biztalk.org/btf-2-0/manifest">
    <reference uri="#insurance_claim_document_id">
      <description>Insurance Claim</description>
    </reference>
    <reference uri="CID:claim.tiff@claiming-it.com">
      <description>Facsimile of Signed Claim Document</description>
    </reference>
  </manifest>
</SOAP-ENV:Envelope>```
6.3. Biztalk and online aggregation

Is Biztalk mostly hype or is it truly a silver bullet solution as the Microsoft fans have claimed? Based on the above discussion of Biztalk’s capabilities, it seems that Biztalk does not deal with the issue of data heterogeneity at all. What Biztalk offers is an electronic envelope for delivering XML messages. It is not responsible for the data schema of the XML messages. If carrier A can only process XML messages created according to data schema A and the aggregator sends it a Biztalk message with the XML data created according to a different data schema, then carrier A will not be able to process the aggregator’s XML data. Thus Biztalk does not solve the problem of data heterogeneity. Because the content of Biztalk messages are in XML format, Biztalk can do no more to solve syntactic and semantic data incompatibility problems than XML.

The function fulfilled by Biztalk is analogous to the function fulfilled by the Post Office. The Post Office allows an American to send a letter written in English to a Mexican. However, if the Mexican can’t understand English, then the communication is useless. Similarly, if two systems are designed to process data according to different data schemas, then using Biztalk does not help them interoperate at all.
7.0. Data standardization efforts in the insurance industry

Many firms believe that data standardization is the solution to the data heterogeneity problems in their web services. If every firm uses the data standards to define their data schemas, then the web services of all these firms will be able to interoperate.

7.1. ACORD data standards

In the insurance industry, the function of creating data standards is delegated to ACORD, which is a consortium of insurance brokerage firms, insurance carriers, and insurance software providers. Representatives from the participating insurance firms meet regularly to create data standards for insurance products. [AC 1999]

The process of creating data standards involves the following steps. First the firms submit their data requirements. Then the firms work together to create a draft version of the data standard that meets these data requirements. Frequently, large firms will submit their own version of the draft data standard, which has been optimized for the needs of the submitting firm. The data standards committee votes to determine which version of the draft is the best one. The chosen draft version of the data standard is posted on the ACORD web site (http://www.acord.org) for insurance firms to review and provide feedback. After being reviewed, updates are made to the draft, and a revised version of the draft standard is posted on the web site. After several iterations of revisions, the draft becomes a recommendation. In many ways, this resembles the way web standards are created in the World Wide Web Consortium (W3C).

ACORD have created EDI data standards for many lines of insurance products. Automation Level 3 (AL3) is ACORD’s newest data standard format for traditional EDI implementations. AL3 standards have been created for many personal insurance products such as auto and life insurance. AL3 standards for commercial insurance products, however, are still in the early stage of the development cycle. AL3 data standards are not readable by people. They need to be encoded and decoded with proprietary software packages sold by ACORD. Currently AL3 standards are still in the early adoption phase.

Due to the popularity of XML, ACORD is starting to develop XML data standards for insurance products. Because XML data standards are much easier to use than AL3, ACORD is switching its focus from AL3 to XML. Currently draft versions of XML data standards have been created for several lines of personal as well as commercial insurance products. These include Personal Auto, Life, EPLI, D&O, Worker’s compensation, Fiduciary, and Commercial Fleet; just to name a few. The complete list of XML data standards can be found at http://www.acord.org.

7.2. Problems with data standardization

Not all insurance firms are represented in ACORD. Joining ACORD requires an annual membership fee that is not cheap. Small insurance companies may not be able to justify this expense on their budget. Insurance firms that are not members are not allowed to submit data requirements and vote on data standards. They can only view the data standards created by ACORD. As a result, ACORD data standards may not meet the data requirements of many firms.
For example, in the insurance application for business owner protection insurance in one of the leading U.S. insurance carriers, there are questions regarding the number of employees in the applicant firm. The required data can be represented in the following XML format, which only shows the XML start tags for the sake of simplicity.

```xml
<numberOfEmployees>
    <numberOfDomesticEmployees>
        <numberOfFullTimeEmployees>
            <numberOfUnionEmployees>
                <numberOfNonUnionEmployees>
                    <numberOfPartTimeEmployees>
                        <numberOfUnionEmployees>
                            <numberOfNonUnionEmployees>
                                <numberOfForeignEmployees>
                                    <numberOfFullTimeEmployees>
                                        <numberOfUnionEmployees>
                                            <numberOfNonUnionEmployees>
                                                <numberOfPartTimeEmployees>
                                                    <numberOfUnionEmployees>
                                                        <numberOfNonUnionEmployees>

The current version of ACORD’s XML data standard for business owner protection insurance, however, has the number of employee element in the following XML formats.

```xml
<NumberOfEmployees>
    <FullTimeEmployees>
    <AverageNumberOfEmployees>
    <NumberOfFullTimeEmployees>
    <NumberOfPartTimeEmployees>
    <NumberOfFullTimeResidenceEmployees>
```

It is quite obvious that the current version of ACORD’s XML data standards does not satisfy all the data requirements of this particular insurance carrier. When the carrier wants to know the number of non-union domestic employees that the applicant has, the ACORD XML standards can’t be used to convey this information because they do not have any data element for representing this data.

Even for the data elements that are available in ACORD standards, there can be ambiguity problems. For example, when the `<NumberOfFullTimeEmployees>` tag is used in an XML message, how should this particular carrier interpret it? Does the `<NumberOfFullTimeEmployees>` include non-union employees? Does it include foreign employees?

When the ACORD data schema documentation is consulted, the following definition is provided for the context of `<NumberOfFullTimeEmployees>`: “The number of full time employees that is associated with the policy, risk, or classification exposure”. This data schema definition provides
no information about whether non-union or foreign employees are included. This example further illustrates that data standards will not be able to satisfy the data requirements of all insurance carrier’s systems.

ACORD’s data standardization efforts also face the same challenges that plagued all data standardization efforts. The data requirements of firms change over time due to changes in the business requirements that may be induced by industrial or organizational factors [GH 2000]. When the data requirements change, the existing version of the data standards may no longer meet the firm’s data needs. This requires changing the data standards. However, changing the data standards takes time, and it needs approval from the other members of ACORD. In addition, other firms have created software systems based on the existing data standards. Creating new standards would mean that their software systems would no longer be compliant with the newest data standards, and this could force them to alter their systems, which requires effort and money. Thus there will be resistance to changes to the data standards. As a result, data standards will frequently not meet the dynamic data requirements of many firms.
8.0. Scalability – the problem that remains unsolved

The previous four chapters showed that data standardization, XML, Biztalk, and SOAP are not optimal solutions for data heterogeneity problems.

8.1. Data standardization

Data standardization provides a good solution if everyone is using the same data standards for communication and have the same semantic interpretations for the data standards; in this case, there would be no data heterogeneity problems. However, if changes in the data requirements of the business systems are frequent and current data standards can’t accommodate these changes, then data standards do not provide an adequate solution because it takes a long time to create new data standards. Thus data standardization does not provide a scalable solution that can adapt to frequent changes in data requirement.

8.2. XML, SOAP, and Biztalk

XML, SOAP, and Biztalk are good messaging technologies for facilitating communication between distributed applications. However, if one system is designed to handle messages based on one data schema and another system is designed to handle messages based on a different data schema, then neither XML, SOAP, or Biztalk can provide an automated solution. Human intervention is needed. System designers are needed to manually create a mapping between the two data schemas. If the data schema for either system changes, then system designers are needed to analyze the change and manually create a new mapping between the two data schemas. If there are N system each using a different data schema and all N systems need to communicate with each other, then there are \( (N^2 - N)/2 \) or \( O(N^2) \) mappings between the data schemas that must be maintained manually. If anyone of the systems change its data schema, then all the other N-1 systems would need to change their mappings to this system’s data schema. This approach is not scalable and is not flexible enough for dealing with frequent changes to the data schema of business systems.

The following example further illustrates this point.

8.3. Scalability Problem Example

This example simulates a four period scenario in which the data requirements in the aggregation change in each period. In period one, the aggregator is only working with two carriers: AIG and Travelers. In period two, Hartford enters into the aggregation with new data requirements. In period 3, AIG changes it data requirements. In period four, Hartford changes its data requirements. For all four periods, the partial data schema of the carriers and the aggregator are shown. The conversion rules needed by the aggregator for mapping its data schema to each of the carrier’s data schema are also shown.

In this example, the data schema and the conversion rules are presented in pseudo code in XML-like format. The aggregator’s data schema satisfies the union of the data requirements of all the carriers. Thus every time a carrier has new data requirements that can’t be satisfied by the
aggregator’s current data schema, then the aggregator needs to change its data schema. This in turn, forces the aggregator to change the conversion rules that map its data schema to those of the carriers. It is these manual changes that make this approach unscalable in the long run.

As a disclaimer, this is only a hypothetical example, and the data used does not represent actual data that AIG, Travelers, or Hartford use in their systems. The names AIG, Travelers, and Hartford are just being used to make the example more readable. However, the types of data heterogeneity problems that it reveals is realistic, and these problems do occur in the insurance industry.

Period 1

![Diagram](8.1. Aggregator to carrier mappings for period 1.)

In period 1, the aggregator’s data schema contains only two lines, and its mappings to the carriers’ data schemas are relatively simple.

Carrier: AIG
Partial data schema:  
<#domesticEmployees>  
<#foreignEmployees>

Carrier: Travelers
Partial data schema:  
<#employees>

Aggregator
Partial data schema:  
<#domesticEmployees>  
<#foreignEmployees>

Aggregator --> AIG mapping
<#domesticEmployees> --> <#domesticEmployees>  
<#foreignEmployees> --> <#foreignEmployees>

Aggregator --> Traveler mapping
<#domesticEmployees> + <#foreignEmployees> --> <#employees>
Period 2 (new carrier added)

In period 2, the new data requirements introduced by Hartford forces the aggregator to expand its data schema to six lines. Its mappings to the existing carriers’ data schemas also needed to be altered. The amount of code needed to map the aggregator’s data schema to the carriers’ data schemas also doubled in size.

Hartford’s data schema has the same structure as that of AIG, but their context are different. Hartford is only interested in fulltime employees while AIG is interested in both the number of fulltime and part-time employees.

![Diagram](image)

Figure 8.2. Aggregator to carrier mappings for period 2. After changing its data schema, the aggregator had to discard rule sets 1 and 2. It had to create new conversion rule sets 3, 4, and 5 to map its new data schema to those of AIG, Travelers, and Hartford.

Carrier: Hartford
Partial data schema: 
- `<#domesticEmployees>`
- `<#foreignEmployees>`
Context: full time employees only

Carrier: AIG
Partial data schema: 
- `<#domesticEmployees>`
- `<#foreignEmployees>`
Context: full time and part time employees

Carrier: Travelers
Partial data schema: 
- `<#employees>`
Context: full time and part time employees
Aggregator
Partial data schema:

<#domesticEmployees>
  <#fulltime>
  <#parttime>
  <#foreignEmployees>
  <#fulltime>
  <#parttime>

Aggregator --> Hartford mapping

<#domesticEmployees>
  <#fulltime>  -->  <#domesticEmployees>

<#foreignEmployees>
  <#fulltime>  -->  <#foreignEmployees>

Aggregator --> AIG mapping

<#domesticEmployees><#fulltime> + <#domesticEmployees><#parttime>
  --> <#domesticEmployees>

<#foreignEmployees><#fulltime> + <#foreignEmployees><#parttime>
  --> <#foreignEmployees>

Aggregator --> Traveler’s mapping

<#domesticEmployees><#fulltime> + <#domesticEmployees><#parttime>
  + <#foreignEmployees><#fulltime> + <#foreignEmployees><#parttime>
  --> <#employees>
Period 3 (changes in AIG’s data requirement)

In period 3, AIG changed its data schema to obtain data with a greater granularity of detail. It needed to further classify the domestic and foreign employees into union and nonunion employees. The aggregator is forced to expand its data schema to ten lines. Its mappings to the existing carriers’ data schemas also changed.

![Aggregator to carrier mappings for period 3. After changing its data schema, the aggregator had to discard rule set 3. It had to create new conversion rule set 6 to map its new data schema to that of AIG.](image)

**Carrier: AIG**
Partial data schema:  
<#domesticEmployees>  
    <#union>  
    <#nonunion>  
    <#foreignEmployees>  
    <#union>  
    <#nonunion>  

Context: full time and part time employees

**Carrier: Hartford**
Partial data schema:  
<#domesticEmployees>  
<#foreignEmployees>  

Context: full time employees only; includes both union and nonunion employees

**Carrier: Travelers**
Partial data schema:  
<#employees>  

Context: full time and part time employees; includes both union and nonunion employees
Partial data schema:

```xml
<#domesticEmployees>
  <#fulltime>
  <#parttime>
  <#union>
  <#nonunion>
<#foreignEmployees>
  <#fulltime>
  <#parttime>
  <#union>
  <#nonunion>
```

Aggregator --> Hartford mapping

```xml
<#domesticEmployees>
  <#fulltime> --> <#domesticEmployees>

<#foreignEmployees>
  <#fulltime> --> <#foreignEmployees>
```

Aggregator --> AIG mapping

```xml
<#domesticEmployees><#union> --> <#domesticEmployees><#union>
<#domesticEmployees><#nonunion> --> <#domesticEmployees><#nonunion>
<#foreignEmployees><#union> --> <#foreignEmployees><#union>
<#foreignEmployees><#nonunion> --> <#foreignEmployees><#nonunion>
```

Aggregator --> Traveler’s mapping

```xml
<#domesticEmployees><#fulltime> + <#domesticEmployees><#parttime>
  + <#foreignEmployees><#fulltime> + <#foreignEmployees><#parttime>
  --> <#employees>
```
Period 4 (changes in Hartford’s data requirement)

In period four, Hartford changed its data schema to categorize domestic and foreign employees into union and nonunion employees. Hartford, however, is still only interested in fulltime employees while the other two carriers needed information for both fulltime and part-time employees. The aggregator’s data schema increased in size to fourteen lines, and it once again needed to change its mappings to the carriers’ data schema. The size of the code needed to map the aggregator’s data schema to Traveler’s data schema has increased by a factor of eight.

Figure 8.4. Aggregator to carrier mappings for period 4. After changing its data schema, the aggregator had to discard rule sets 4, 5, and 6. It had to create new conversion rule sets 7, 8, and 9 to map its new data schema to those of AIG, Travelers, and Hartford.

Carrier: Hartford
Partial data schema:  
<#domesticEmployees>
  <#union>
  <#nonunion>
<#foreignEmployees>
  <#union>
  <#nonunion>
Context: full time employees only; includes both union and nonunion employees

Carrier: AIG
Partial data schema:  
<#domesticEmployees>
  <#union>
Context: full time and part time employees
Carrier: Travelers
Partial data schema: part time employees; includes both union and nonunion employees

Aggregator
Partial data schema:

Aggregator --> Hartford mapping

Aggregator --> AIG mapping
This example illustrated that while XML is much easier to work with comparing to traditional EDI data standards, which require proprietary software for encoding and decoding, XML by itself does not solve the scalability problems that are inherent in most aggregation efforts. This example only involved three carriers, one data element, and three changes. Imagine the amount of changes and maintenance work that would be required when there are many carriers, thousands of data elements, and many changes for each data element. It is quite obvious that XML alone will not solve the problem of scalability that plagued traditional EDI.

Chapter 10 of this thesis provides a design that attempts to solve the problem of scalability. This design allows the data schema of the aggregator and the carrier’s system to be changed without forcing the other carriers’ systems to change their mappings. This design is based on concepts from the Context Interchange (COIN) Framework, which will be discussed in the next chapter.
9.0. Context Interchange (COIN) Framework

The context Interchange (COIN) framework is a novel approach for solving data heterogeneity problems in the aggregation of data from multiple disparate data sources. The COIN framework, which was created by Stuart Madnick and Michael Siegel at MIT’s Sloan School of Management, takes the context of data into account during aggregation, and it provides context mediation for data sources that have different contexts for their data.

While most data aggregation approaches take either a tight-coupling or a loose-coupling methodology, COIN seeks to provide a solution that offers the benefits of both worlds. Tight-coupling approaches offer faster development time and greater simplicity, but they are inflexible to changes in data requirement. An example of a tight coupling approach is a client-server system in which the implementation of the client is completely dependent on the implementation of the server. If the implementation of the server changes, then the client will no longer work, and it needs to be re-implemented. Loose-coupling approaches offer much greater flexibility to handle changes in data requirement, but they often have performance issues and are often too complex to implement for practical purposes [BFGJ 1997]. An example of a loose-coupling approach is a client-server system in which the implementation of the client is not strictly dependent on the implementation of the server. Thus when the implementation of the server components change, the client system will still work, and the client does not need to be re-implemented. The COIN approach offers both the flexibility of loose-coupling and the practicality of tight-coupling. The architectural diagram on the next page is from [BFGM 1997] and shows the COIN model.

9.1. Architectural overview

At the heart of the COIN framework is the context mediator, which resolves context differences between data from disparate sources. Built around the context mediator are server components, client components, and wrappers for data sources. The client components can be incorporated into any application on any platform. They take data query requests and their desired context from users or applications. These requests are converted to logical queries by the SQL Compiler component in the server. At the other end of the framework are data source wrappers, which enable COIN to aggregate data from databases, HTML web pages, and semi-structured documents. For each type of data source, a data wrapper can be built. Thus the types of data sources that COIN can work with are unrestricted. In addition, because COIN communicates to data sources and clients through the HTTP protocol, platform independence is achieved.

In addition to the Context Mediator, COIN also have several other important server components. They are the COIN repository, the optimizer, and the executioner. The COIN repository contains context axioms for data, domain specific knowledge for context mediation, and conversion axioms for converting data from one context to another. The optimizer is responsible for transforming client queries into a form that minimizes the amount of computations needed and also the amount of potential network delay experienced [FK 1997]. The Executioner carries out the optimized queries, and it serves as the interface between the data sources, the client, and the other server components. Thus it provides a layer of abstraction between these components. This allows changes to be made in the implementation of different parts of COIN without requiring changes in the other parts.
Figure 9.1. An architectural diagram of the COIN framework.

9.2. The Process of Context Mediation

COIN allows the disparate data sources to have data schemas with different syntactical representations and contexts. During context mediation, elevation axioms are used to map the data from the different data sources and their associated contexts to a common domain model. The client queries, which have been converted to logical form, are performed on this intermediate data. 
representation. The result of the queries are converted to the desired context of the client using conversion axioms.

The use of the common domain model and elevation axioms allows the data sources to have different data schemas, and it eliminates the need for data standardization. The use of elevation axioms, context axioms, and conversion axioms provides flexibility for the framework, and this is superior to tight-coupling approaches that hard-wire the conversion rules into the application code. The use of these axioms is also simple, and it avoids the complexity of most loose-coupling methodologies.

Below is a sample domain model from [PT 1997].

![Diagram](image)

Figure 9.2. An architectural diagram of a sample domain model.

### 9.3. The Domain Model

The domain model captures knowledge in a particular domain or area of specialty such as finance. It is consisted of a network of entities in the domain and the relationships and constraints among these entities. There are two major types of entities: simple and derived. Simple entities are primitive data types such as numbers and strings. The derived entities are created from the simple entities or other derived entities. “ExchangeRate”, for example is
derived from the simple entity number, and it has extra attributes such as fromCurrency and toCurrency.

There are three major types of relationships in the domain model. They are inheritance, attribute, and modifier. Inheritance refers to sub-classing. “ExchangeRate”, for example is a sub-class of number. Attribute refers to properties of an entity. “Company name”, for example is an attribute of “company financials”. Modifier refers to a different representation of an entity. Date, for example, is a type of string with a specific format and representation.

9.4. COIN and the Semantic Web

The semantic web is a vision of creating a network where data on the Internet can be used by machines not just for displaying purposes, but for automation, integration, and reuse of data across various applications [W3C 2000]. The implementation of the semantic web is currently being lead by members of the World Wide Web Consortium (W3C). Under the leadership of Tim-Bernes Lee, the inventor of the World Wide Web, the W3C is trying to transform data on the World Wide Web, which is mostly human-readable only, into data that can be understood and processed by software agents and systems. Once the semantics of data can be understood by software systems across the semantic web, then it will be much easier for these systems to interoperate and share data.

At the core of the semantic web implementation is XML, which provides the syntax for adding meta-data to data. Building on top of XML is Resource Description Framework (RDF), which is a language that uses XML syntax to describe resources and the relationships between these resources on the World Wide Web. One of RDF’s strengths is its ability to handle source attribution. When a web entity is expressed in RDF format, it becomes easy to identify the source of this entity. RDF also allows constraints and relationships between web entities to be easily expressed.

The most basic RDF model is a triple, which is consisted of three resources. The first is the subject; the second is the object; and the third is the predicate, which is an arc that links the subject and the object. Another way of describing this model is that the <subject> has a property <predicate> valued by <object>. [ACP 2000]

![Figure 9.3. A sample RDF triple graph.](http://ama.cpe.fr/index.html http://publ.org/DC/cez8z mailto:champin@cpe.fr)

The triple in figure 9.3. states that “Champin is the creator of index.html.”.

Building on top of RDF are ontology languages such as DARPA Agent Markup Language + Ontology Interchange Language (DAML + OIL), which use RDF syntax for building ontologies and domain models. The objective of DAML + OIL is to help systems model their data
according to an ontology, where the semantics of the data are exposed and logical inferencing can be applied to the data [DO 2000]. A complete analysis of DAML + OIL, however, is beyond the scope of this thesis. Interested readers can go to http://www.daml.org/2001/03/daml-oil-index for more information.

Even though COIN and the semantic web are independent research efforts that have started separately, the two efforts have common goals, and their works are complementary. While the semantic web group is focusing on enabling systems to express the semantics of their data, the COIN group is focusing on helping systems reconcile the contextual and semantic differences between data of different sources. Technologies for the semantic web, such as XML, RDF, and DAML + OIL, can be incorporated into the implementation of the COIN framework for representing context data and domain models. The extension of the COIN framework proposed in chapter ten, for example, uses XML syntax for the representation of meta-data and the representation of the domain model. Once the semantics of the data on the web is exposed, then the COIN framework can be used to help systems aggregate data from all over the semantic web and reconcile their semantic differences.
10.0. COIN – Insurance Extension (IE)

This chapter proposes a scalable solution for online insurance aggregation that enables the aggregator and the carriers to change their data schema without forcing all other carriers in the interchange to change their software and mappings. This solution, which I shall name COIN-IE (Insurance Extension), uses many concepts from the COIN Framework. For example, the aggregator and the carrier’s systems all map their data schema to a common domain model using elevation axioms. Context differences between the data schema of the different systems are mediated using conversion axioms. The difference between COIN and COIN-IE is that COIN-IE assumes that the domain model is incomplete and not perfect. In COIN-IE, the domain model is allowed to change and evolve over time to satisfy new data requirements that can’t be mapped to the previous version of the domain model. The value added by COIN-IE is that it allows the domain model to be changed without forcing the existing carriers to change the mapping of their data schema to the domain model. Thus this framework provides a scalable solution for dealing with changes in data requirements over time.

10.1. Design Criteria

1. Scalability - The design must be able to work with a large number of quoting services. The cost and maintenance efforts required must not increase substantially when a large number of quoting services are added to the aggregation. In addition, adding new quoting services should not require existing quoting services to alter their software implementation.

2. Flexibility - The design must be able to handle frequent changes in the data requirements of existing quoting services over time and the new data requirements of new quoting services.

3. Simplicity - The design must be simple to implement. Designs that are too complex are frequently good on paper but not get implemented successfully in practice due to their complexity.

4. Fault Tolerance - The failure of any quoting service should not affect the performance of other quoting services. The aggregator must be able to function properly even when some of the quoting services are offline.

5. Correctness - The information produced by the aggregator must be correct, and the aggregator must not send incorrect data to the quoting services as a result of its inability to solve data heterogeneity problems.

10.2. Classification of new data requirements

To solve the data heterogeneity problems introduced by new data requirements or changes in data requirements over time, we first need to classify the types of new data requirements that can cause the domain model to change.
New data requirements can be classified into several categories with regard to elements in the existing domain model. The <Partial incomplete hybrid domain model> tag is used to indicate that the domain model is incomplete and only a portion of that incomplete domain model is shown. In addition, it is called a hybrid because it is structurally different from the domain model used in the original COIN framework.

1. The new data element is independent of all existing elements in the domain model.

Example:
<Partial incomplete hybrid domain model>
  <address>
    <phoneNum>
    <companyName>
  </Partial incomplete hybrid domain model>

New data requirement:
<#employees>, which is unrelated to any of the elements in the existing domain model.

2. The new data element has the same structure and semantics as one of the existing data elements even though this element has a different name.

Example:
<Partial incomplete hybrid domain model>
  <address>
    <phoneNum>
    <companyName>
  </Partial incomplete hybrid domain model>

New data requirement:
<companyAddress>, which is equivalent to <address>.

3. The new data element is a sub-class of one of the existing data elements. This element has extra attributes that provide a richer set of context information.

Example:
<Partial incomplete hybrid domain model>
  <address>
    <phoneNum>
    <companyName>
  </Partial incomplete hybrid domain model>

New data requirement:
<address>
  <streetAddress>
  <city>
This element extends the existing element <address>. It should be easy to map the new <address> element to the existing <address> element. However, it will be difficult to convert data in the existing <address> element to the new <address> element format.

4. The new data element is a super-class of several existing data elements. This element has a list of attributes that are shared by all the sub-class elements.

Example:
<Partial incomplete hybrid domain model>
  <#employee>
    <#fulltime>
    <#union>
    <#nonunion>
  <#parttime>
    <#union>
    <#nonunion>
  <#employee>
    <#fulltime>
    <#foreign>
    <#domestic>
  <#parttime>
    <#foreign>
    <#domestic>
</Partial incomplete hybrid domain model>

New data requirement:
  <#employees>
    <#fulltime>
    <#parttime>

The new element is a super-class of the two existing elements in the hybrid domain model because its sub-elements <#fulltime> and <#parttime> are present in both existing elements.

5. The new data element is a sibling of one of the data elements. Two data elements are defined to be siblings if they refer to the same entity, but they have different attributes that provide different sets of context information.

Example:
<Partial incomplete hybrid domain model>
New data requirement:

```
<#employees>
  <#union>
  <#nonunion>
  <#foreign>
  <#domestic>
</partial incomplete hybrid domain model>
```

10.3. Discussion of strategies for accommodating new data requirements

The solution to the first type of new data requirements is quite simple; just add the new data element needed to satisfy the new data requirement into the existing domain model. No existing quoting services will be affected since they don’t interact with this new data element, and this new data element has no effect on the other data elements.

The solution to the second type of new data requirements is even more trivial; no changes are needed for the domain model. The new quoting service just maps its data element to the corresponding data element in the existing domain model. Even though the two elements have different names, they are the same structurally and semantically.

Handling the third type of new data requirements is not so easy. Consider the following scenario.

```
<partial domain model>
  <#employees>
  new data element
  <#employees>
    <#union>
    <#nonunion>
```
The design decision here is whether to modify the existing `<#employees>` element by adding attributes to it or by adding a new data element with new attributes to the domain model. If the existing element is modified, then mappings of external data schemas to this element in the domain model may need to be modified. If a new element is added, then the domain model increases in size and becomes more difficult to manage. Considering that a new data element is added, whether the new element uses the same name or a different name is another design decision.

The solution to the fourth type of new data requirements is easier than the last one. The domain model can be left unchanged and the new quoting service can be requested to map its data element to any one of the existing elements in the domain model that satisfies the data requirements. However, the aggregator and the new quoting service must coordinate so that they are mapping to the same element. Otherwise, if the aggregator maps to element A while the quoting service maps to element B, then the quoting service will not get the data it needs.

An alternative approach is to add a new data element with the desired attributes. Both the aggregator and the new quoting service will map to this new element. The disadvantage here is that there is an extra element to deal with in the domain model.

The fifth type of new data requirements is the hardest to handle. One solution is to merge the two sibling elements. For example, if the two elements are:

```xml
<#employees>
  <#union>
  <#nonunion>
</#employees>

and

<#employees>
  <#domestic>
  <#foreign>
</#employees>
```

then the following data element can be created by merging the attributes.

Case 1:
```
<#employees>
  <#union>
  <#nonunion>
  <#domestic>
  <#foreign>
</#employees>
```

However, it is also possible for the new element to have the following two forms.
Case 2:
<employees>
  <domestic>
    <union>35</union>
    <nonunion>20</nonunion>
  </domestic>
  <foreign>
    <union>5</union>
    <nonunion>10</nonunion>
  </foreign>
</employees>

Case 3:
<employees>
  <union>35</union>
  <foreign>5</foreign>
</employees>

To illustrate the problems that can occur, data is added to these three elements.
Comparing case 1 to case 2, not only is the structure of the data elements different, the context and the granularity of the context information are also different. Case 1 allows the carrier to immediately extract the number of domestic, foreign, union, and nonunion employees directly from the data. However, there is no way of finding out how many domestic employees are also union employees or how many foreign employees are nonunion employees. In case 2, it is not possible to directly extract from the data tree the total number of domestic employees or union employees. Computations are needed to derive these values. The data structure in case 2 provides context information in a higher level of granularity, but extra computations are required to extract summary values such as the total number of domestic employees. Thus deciding on which data structure to use can be a difficult decision.

Another problem is that if the existing sibling object is merged with the new sibling object, then all the mappings to the existing sibling object will need to be changed. If these changes are undesirable, then perhaps it will be a good idea to have both the existing sibling object and the new sibling object coexist in the domain model.
10.4.0. COIN – IE (Insurance Extension) Framework

The following sections provide a new design that solves the scalability problem described in chapter eight. Under this design, changes in the domain model do not require all the quoting services to change their mappings to the domain model. In addition, the aggregator is not forced to maintain a separate mapping for each quoting service; it only has to map to the common domain model. Thus scalability is achieved.

Under the COIN-IE framework, the aggregator maps its data to the hybrid domain model. Then it sends the data, which is in the hybrid domain model representation, to the carriers. Each of the carriers have its own mappings to the hybrid domain model, and they use these mappings to extract the relevant data from the aggregator’s message. The irrelevant data in the message is ignored.

In some sense, this approach has some resemblance to a global schema approach. However, the difference here is that when the global schema changes, all the carriers would need to change their mappings to the global schema, which can be quite a bit of work. Under the COIN-IE framework, when the hybrid domain model changes, the carriers don’t have to change their mappings to the hybrid domain model at all. The following sections will describe the components and mechanism that allows the carriers to keep their old mappings to the hybrid domain model when the hybrid domain model changes.

10.4.1. The Family Container

This design introduces a new type of entity for the domain model. The new entity is called the family container. This container is used to hold objects and the relationships among them. Three types of relationships can exist between objects in the family container; they are “parent”, “child”, and “sibling”. When an object is a parent of other objects, this object encapsulates the features that are common to all these objects. An object is a child of another object when this object extends the parent object with a richer set of attributes. The sibling relationship is used to link objects that have a subset of their properties in common. In figure 10.1. on the next page, the parent object is <#employees>, which have no attributes. There are four children objects that extend this parent object with new attributes. These four children objects are also siblings of each other.

In some sense, the relationships between objects in the family container have strong resemblance to the concept of inheritance in object-oriented programming languages. Just like inheritance in C++, the child objects in COIN-IE extend the functionality of their parent objects by providing a richer set of information. However, the similarities stop there. The COIN-IE objects are data objects only; they don’t have methods or functions within them.
10.4.2. The Head Sibling Object

Inside each family container, there must be a “head sibling” object. The “head sibling” object is defined to be the object that can be used to derive the values of all the other objects in the family container. The “head sibling”, therefore, contains data of the greatest granularity of detail. It is also the object that the aggregator must map its data element to. If the data schema of the aggregator changes and it maps to a new “head sibling” object, then the new “head sibling” object will be assigned a number greater than the previous one. Thus the “head sibling” object with the highest number is the one that the aggregator maps to.
Figure 10.2 and Figure 10.3 above illustrate how the aggregator and the carriers map their data schema to the hybrid domain model. In period N, the aggregator maps to “head sibling” 1. The conversion axioms are used to derive values for sibling 1 and sibling 2 from “head sibling” 1. Quoting services 1 and 2 extract their data from sibling 1 while quoting services 3 and 4 extract their data from sibling 2. In period N+1, quoting services 6 enters the interchange with new data requirements, and “head sibling” 2 is created to satisfy these requirements. Now the aggregator maps to “head sibling” 2, and conversion axioms are added so that “head sibling” 1 can derive its values from “head sibling” 2. Quoting service 6 extracts its data from “head sibling” 2, but everything else is unchanged. Quoting services 1, 2, 3, and 4 keep their existing mappings to the hybrid domain model even when the hybrid domain model has changed.

10.6.3. Family Conversion Axioms

Associated with each family container is a set of conversion axioms. These axioms use the values from the “head sibling” object to derive the values for all the other objects in the family container.

10.6.4. Modifying the Domain Model

This section describes the rules for modifying the domain model.

1. If the new data element is unrelated to the existing data elements, then simply add the new data element to the domain model.

2. If the new data element has the same structure and semantics as one of the existing elements in the domain model, but their names are different, then make no changes to the domain model.
Simply asking the new quoting service to map to the existing element in the existing domain model is sufficient.

3. If the new data element is a sub-class of an existing data element, then create a new data element in the domain model, and use a sub-class link to join the sub-class to the super-class.

4. If the new data element is a super-class of several data elements, then create a new data element in the domain model, and add a super-class link from the super-class element to all the sub-class elements.

5. If the new data element is a sibling of an existing data element, then create a new data element in the domain model, and add a sibling link between the data elements.

![Insurance Hybrid Domain Model](image)

Figure 10.4. Example of a partial hybrid domain model for the insurance industry.
10.6.5. Aggregator’s Mappings to the Domain Model

When mapping to the domain model, the aggregator must make sure that if an element has a value, its siblings, sub-class, and super-class must also have a value. This is necessary because when a sibling, sub-class, or super-class object exists, it must have been introduced by some data requirement of some quoting service. By satisfying this rule, the aggregator does not have to keep track of which data element is specifically needed by which quoting service.

The aggregator maps each of its data elements to the “head sibling” object in the corresponding family container, and then it invokes the conversion axioms for each family container to derive the values for all the other objects in the family containers.

When the aggregator changes its mappings to the domain model and maps to a new “head sibling” object, it must provide conversion axioms that map the new “head sibling” object to the former “head sibling” object. By providing these axioms, the aggregator does not have to create mappings from the new “head sibling” object to all the other objects in the same family container.

10.6.6. Quoting Services’ Mappings to the Domain Model

For the quoting services, each of them only has to care about its own mappings to the domain model. One of the main challenges for the quoting services is what to do when they see multiple data elements with the same name in the data stream sent from the aggregator. To solve this problem, they must look at the attributes and the structure of each element. This in some sense is the signature of the data element; this is like overloading functions with the same name but different number or types of arguments in languages like C++. The quoting services will only extract the element with the correct signature.

10.6.7. Analysis of Design

In this framework, the aggregator maps to the member of the “#employee” family container that matches its data schema. This member of the family container should be the one with the most context information and the finest level of granularity in detail. This member is defined so that all other family members can be derived from it, and that is why this member is called the “head sibling” object. The set of conversion axioms for this family container is used to derive the values for all other members of the family container from the “head sibling” object.

After all the family member objects have been given a value, then this family container, along with other objects, are send to all the quoting services. When dealing with a family container, a quoting service first need to identify the family member that it maps to by looking at each member’s attribute and structure signature. This quoting service will only extract values from the member of the family container that has the right signature.
One concern that may arise is that what if there are two member objects with the same signature. However, this is an impossible scenario because if two member objects have the same attributes and the same structure, then they are by definition the same member of the family container.

Another serious concern with this design is that what if the new data requirements force the aggregator to map to a new data element. This would require new rules for mapping the new aggregator member to all the other members of the family container, and scalability is not achieved. Fortunately, there is a simple solution. Just add conversion axioms that map the new aggregator member to the old aggregator member. This will eliminate the need for the aggregator to change all the existing conversion axioms for this family container.

The advantage of this design is that it adequately satisfies the design criteria, and it solves the problem of the evolving domain models. When new data requirements force the domain model to change, either a new data element is added outside of the family containers or an element is added to a family container. In either case, the existing quoting services do not need to change their domain model because the part of the domain model that they map to has not changed. For the aggregator, the problem of scalability is significantly alleviated because it only has to map to one of the members of the family container. The rest of the members are derived from the member that the aggregator maps to.

**Disadvantages**

The disadvantage of this design is that the name space of the domain model can get large very quickly because new data requirements add new data elements to the domain model.

Another plausible argument against the current design is why not collapse the family container and merge all the members into a single data element that can be used to derive all the other member elements. The problem with this approach is that when the data requirements change, this single aggregate element will change. Because all the quoting services map to this aggregate element, they would need to change their mappings to this aggregate element. Thus scalability is not achieved.

**10.6.8. Alternative Design**

One feasible alternative design to the COIN-IE framework is to use a mediator service between the aggregator and the quoting services. The mediator will use conversion axioms to map the aggregator’s data schema to those of the quoting services. The aggregator and the quoting services don’t have to know anything about each other’s data schema because the mediator service will handle the schematic differences between them. This design appears to be much simpler and better.

The problem with this design is that when there are new data requirements forcing the aggregator’s data schema to change, then the mapping from the aggregator’s data schema to all the quoting services would need to be changed as well. Thus scalability is not achieved. One might argue that the mediator may use the approach used by the COIN-IE design and simply add rules that map the aggregator’s new schema to its old schema, which in turn is mapped to the
data schema of all the existing quoting services. This is a good argument, but the problem is who will perform all the system maintenance work for the mediator. Since the mediator needs to know the mappings from the aggregator’s data schema to all the quoting services’ data schemas, the only party that is capable of handling this task is the aggregator. Thus the concept of the mediator is just another name, and all the work is still being done by the aggregator. In addition, this approach is worse than the previous one because it requires a pair-wise solution between the aggregator and each of the quoting services. The aggregator needs to keep track of which quoting service uses which data schema, and it must prepare a separate and different data format for each quoting service. Thus this approach suffers from the scalability problem of traditional EDI. With the COIN-IE design, the aggregator only has to map to the “head sibling” object in the family container in the domain model, and it does not have to care about which quoting service is using which data schema.

![Diagram](image)

Figure 10.5. Schematic of an alternative design where a mediator service does all the conversions for the aggregator and the quoting services.
10.6.9. Domain Model Management Concerns

If a quoting service no longer uses a data element in the domain model, should this element be dropped from the domain model? If it is not dropped, then there could be a problem similar to a memory leak in C++ programs, which is a common problem in programming languages that use memory pointers; the name space of the domain model can explode in size over time and get too complex for practical use.

If maintaining a small name space is indeed important, then the aggregator must keep track of which data element is being used by which quoting services. This may be too much work for the aggregator and introduces scalability problems.

10.6.10. Managing the Message Size and the Name Space

Under the current framework, a new sibling object is added to a family container in the domain model every time a carrier changes its data schema. This approach has two negative side effects. The first is that the size of the message that the aggregator broadcasts to all the carriers will increase in size every time there is a change. The second is that the name space of objects in the domain model also increases in size every time there is a change.

The increase in message size and name space must be managed for this framework to be scalable. When a carrier changes one of its objects, the old version of that object and its corresponding element in the domain model may no longer be used. Thus the unused element can be removed from the domain model to limit growth in the message size and the name space.

Figure 10.6. on the next page illustrates a mechanism for controlling the growth of the message size and the name space. Under this mechanism, a conversion rule container is associated with each family container. The conversion rule container has rule sets that map the head sibling object to the other sibling objects in the family container. Inside each sibling object are three parameters that are being used to determine whether this sibling objects should be removed from the current system. The parameters are: the active? flag, the usage counter, and the last-modified-date. When a carrier changes its data schema, it tells the aggregator which sibling object it is no longer using. The aggregator decreases the usage counter of that sibling object by one. When the usage counter of a sibling object reaches zero, then the aggregator will set the active flag of that object to be false. At this point, the aggregator has the option of eliminating this sibling object from the system. The aggregator also has the flexibility to keep this sibling object in the system for a specific amount of time after the last-modified-date in the event that another carrier might want to use this object in the near future.

Similarly, the active? flag, the usage counter, and the last-modified-date parameters are also used to limit the rule set space. Each time a sibling object that a rule set points to is eliminated, that rule set’s usage counter is decreased by one. When the usage counter for a rule set reaches zero, the aggregator has the choice of eliminating that rule set immediately or keep it for a short period of time.
Optimization can also be used to limit the size of the domain model. For example, if there are two carriers and one of the carriers went through 99 changes in data schema so that carrier 1 maps to “head sibling 1” while carrier 2 maps to “head sibling” 100, then the aggregator can create a new set of conversion rules to map “head sibling” 100 to “head sibling” 1. This allows the aggregator to eliminate “head sibling” 2 to 99 and conversion rule sets 1 to 99 from the hybrid domain model. This type of optimization can be done by the aggregator periodically or whenever it detects that the domain model is getting too big.

### 10.6.11. Trade-off Analysis

The above mechanism provides an effective solution for limiting the message size and the name space growth in the aggregation framework. Sibling objects and rule sets that are not being used are eliminated. In addition, the aggregator has the flexibility to keep an unused sibling object or rule set for a specific amount of time in the event that a new carrier may want to use that object or rule set in the near future.

![Figure 10.6](image_url)

Figure 10.6. This diagram illustrates the meta-data that is being used for managing the size of the hybrid domain model. Rule set 1 maps the “head sibling” object to sibling 1. Rule set 2 maps the “head sibling” object to sibling 2 and sibling 3. Rule set 3 is no longer being used and can be eliminated.

The disadvantage of this approach is that the aggregator has to maintain three parameters for each sibling object and rule set. This adds extra complexity to the framework. Nevertheless, the aggregator only has to change the parameters for a small number of objects and rule sets each time there is a change in the data requirement. Thus this process is still manageable, and the benefits outweigh the disadvantages.
10.6.12. Application of COIN – IE to Web Service Aggregation

The following example illustrates how the COIN - IE framework can be applied to the web service aggregation. It is the same example as the one described in the chapter eight (scalability problem example).

There are four periods in this example, and a change in data requirement takes place between each period. For each period, the following information is provided:

- Diagram of the mappings to the hybrid domain model
- Partial data schema of all the parties
- The sibling objects in the hybrid domain model
- The conversion axioms for mapping the head sibling object to the other sibling objects in the domain model
- The elevation axioms that map each party to the domain model
- An application of the current mappings to sample data

**Period 1**

![Diagram of the mappings to the hybrid domain model for period 1. The aggregator maps its data to the hybrid domain model and sends the data in the hybrid domain model representation. Both AIG and Travelers use their mappings to the hybrid domain model to extract relevant data that is needed. Irrelevant data is ignored.]

**Partial data schemas**

Carrier: AIG
Partial data schema:

```xml
<#domesticEmployees>
<#foreignEmployees>
```
Carrier: Travelers
Partial data schema: <#employees>

Aggregator
Partial data schema: <#domesticEmployees>
<#foreignEmployees>

Incomplete Hybrid Domain Model:

<head Sibling 1>
<meta active = yes, usage_cnt = 2, last_mod_date=4/1/2001>
  <#employees>
    <#domestic>
    <#foreign>

Elevation axioms

Aggregator --> Domain Model elevation axioms
<#domesticEmployees> --> <head sibling 1><#employee><#domestic>
<#foreignEmployees> --> <head sibling 1><#employee><#foreign>

AIG --> Domain Model elevation axioms
<#domesticEmployees> --> <head sibling 1><#employee><#domestic>
<#foreignEmployees> --> <head sibling 1><#employee><#foreign>

Travelers --> Domain Model elevation axioms
<#employees> --> <head sibling 1><#employee><#domestic>
  + <head sibling><#employee><#foreign>

Application to sample data

Sample data from Aggregator:
<#domesticEmployees> 1000 </#domesticEmployees>
<#foreignEmployees> 500 </#foreignEmployees>

Sample data mapped to domain model:
<employee family container>
  <head sibling>
    <#employees>
      <#domestic> 1000 </#domestic>
Sample data extracted by AIG from domain model:
<#domesticEmployees> 1000 </#domesticEmployees>
<#foreignEmployees> 500 </#foreignEmployees>

Sample data extracted by Travelers from domain model:
<#employees> 1500 </#employees>

Period 2

Figure 10.8. Mappings to the hybrid domain model for period 2.

Partial data schemas

Carrier: Hartford
Partial data schema: 
<#domesticEmployees>
<#foreignEmployees>

Context: full time employees only

Carrier: AIG
Partial data schema: 
<#domesticEmployees>
<#foreignEmployees>

Context: full time and part time employees
Carrier: Travelers
Partial data schema: <#employees>
Context: full time and part time employees

Aggregator
Partial data schema: <#domesticEmployees>
                  <#fulltime>
                  <#parttime>
                  <#foreignEmployees>
                  <#fulltime>
                  <#parttime>

Incomplete Hybrid Domain Model:

<head Sibling 1>
<meta active = yes, usage_cnt = 2, last_mod_date=4/1/2001>
   <#employees>
      <#domestic>
      <#foreign>
<head sibling 2>
<meta active = yes, usage_cnt = 2, last_mod_date=4/7/2001>
<#employee>
   <#domestic>
      <#fulltime>
      <#parttime>
   <#foreign>
      <#fulltime>
      <#parttime>

Conversion Axioms within the domain model:

<rule set 1>
<meta active = yes, usage_cnt = 1, last_mod_date=4/7/2001>
<head sibling 1><#employees><#domestic> =
    <head sibling 2><#employees><#domestic><fulltime>
    + <head sibling 2><#employees><#domestic><parttime>
<head sibling 1><#employees><#foreign> =
    <head sibling 2><#employees><#foreign><fulltime>
    + <head sibling 2><#employees><#foreign><parttime>
Elevation axioms

Aggregator --> Domain Model elevation axioms
<#domesticEmployees><fulltime> --> <head sibling 2> <#employees><#domestic><fulltime>
<#domesticEmployees><parttime> --> <head sibling 2> <#employees><#domestic><parttime>
<#foreignEmployees><fulltime> --> <head sibling 2> <#employees><#foreign><fulltime>
<#foreignEmployees><parttime> --> <head sibling 2> <#employees><#foreign><parttime>

Hartford --> Domain Model elevation axioms
<#domesticEmployees> --> <head sibling 2><#employee><#domestic>
<#foreignEmployees> --> <head sibling 2><#employee><#foreign>

AIG --> Domain Model elevation axioms
<#domesticEmployees> --> <head sibling 1><#employee><#domestic>
<#foreignEmployees> --> <head sibling 1><#employee><#foreign>

Travelers --> Domain Model elevation axioms
<#employees> --> <head sibling 1><#employee><#domestic>
+ <head sibling><#employee><#foreign>

Application to sample data

Sample data from Aggregator:
<#domesticEmployees>
  <#fulltime> 800 </#fulltime>
  <#parttime> 200 </#parttime>
</#domesticEmployees>
<#foreignEmployees>
  <#fulltime> 400 </#fulltime>
  <#parttime> 100 </#parttime>
</#foreignEmployees>

Sample data mapped to domain model:
<employee family container>
  <head sibling 2>
  <#employees>
    <#domestic>
      <#fulltime>800</#fulltime>
      <#parttime> 200 </#parttime>
    </#domestic>
    <#foreign>
      <#fulltime> 400 </#fulltime>
    </#foreign>
  </#employees>
</employee family container>
Sample data extracted by AIG from domain model:
<#domesticEmployees> 1000 </#domesticEmployees>
<#foreignEmployees> 500 </#foreignEmployees>

Sample data extracted by Travelers from domain model:
<#employees> 1500 </#employees>

Sample data extracted by Hartford from domain model:
<#domesticEmployees> 800 </#domesticEmployees>
<#foreignEmployees> 400 </#foreignEmployees>

**Period 3 (changes in AIG’s data requirement)**

Figure 10.9. Mappings to the hybrid domain model for period 3.
Partial data schema

Carrier: AIG
Partial data schema:  <#domesticEmployees>
    <#union>
    <#nonunion>
    <#foreignEmployees>
    <#union>
    <#nonunion>
Context: full time and part time employees

Carrier: Hartford
Partial data schema:  <#domesticEmployees>
    <#foreignEmployees>
Context: full time employees only; includes both union and nonunion employees

Carrier: Travelers
Partial data schema:  <#employees>
Context: full time and part time employees; includes both union and nonunion employees

Aggregator
Partial data schema:  <#domesticEmployees>
    <#fulltime>
    <#parttime>
    <#union>
    <#nonunion>
    <#foreignEmployees>
    <#fulltime>
    <#parttime>
    <#union>
    <#nonunion>

Incomplete Hybrid Domain Model:

<head Sibling 1>
<meta active = yes, usage_cnt = 1, last_mod_date=4/1/2001>
    <#employees>
        <#domestic>
        <#foreign>
</head>

<head sibling 2>
<meta active = yes, usage_cnt = 2, last_mod_date=4/7/2001>
    <#employee>
        <#domestic>
Conversion Axioms within the domain model:

<rule set 1>
<meta active = yes, usage_cnt = 1, last_mod_date=4/7/2001>
<head sibling 1><#employees><#domestic> =
  <head sibling 2> <#employees><#domestic><fulltime>
  + <head sibling 2> <#employees><#domestic><parttime>
<head sibling 1><#employees><#foreign> =
  <head sibling 2> <#employees><#foreign><fulltime>
  + <head sibling 2> <#employees><#foreign><parttime>

<rule set 2>
<meta active = yes, usage_cnt = 1, last_mod_date=4/14/2001>
<head sibling 2> <#employees><#domestic><fulltime>
  = <head sibling 3> <#employees><#domestic><fulltime>
<head sibling 2> <#employees><#domestic><parttime>
  = <head sibling 3> <#employees><#domestic><parttime>
<head sibling 2> <#employees><#foreign><fulltime>
  = <head sibling 3> <#employees><#foreign><fulltime>
<head sibling 2> <#employees><#foreign><parttime>
  = <head sibling 3> <#employees><#foreign><parttime>

Elevation axioms

Aggregator --> Domain Model elevation axioms
<#domesticEmployees><#domestic><fulltime> --> <head sibling 3> <#employees><#domestic><fulltime>
Hartford --> Domain Model elevation axioms
<#domesticEmployees> --> <head sibling 3><#employee><#domestic>
<#foreignEmployees> --> <head sibling 2><#employee><#foreign>

AIG --> Domain Model elevation axioms
<#domesticEmployees><union> --> <head sibling 3><#employee><#domestic><union>
<#domesticEmployees><nonunion> --> <head sibling 3><#employee><#domestic><nonunion>
<#foreignEmployees><union> --> <head sibling 3><#employee><#foreign><union>
<#foreignEmployees><nonunion> --> <head sibling 3><#employee><#foreign><nonunion>

Travelers --> Domain Model elevation axioms
<#employees> --> <head sibling 1><#employee><#domestic>
+ <head sibling><#employee><#foreign>

Application to sample data

Sample data from Aggregator:
<#domesticEmployees>
  <#fulltime> 800 </#fulltime>
  <#parttime> 200 </#parttime>
  <#union> 700 </#union>
  <#nonunion> 300 </#nonunion>
</#domesticEmployees>
<#foreignEmployees>
  <#fulltime> 400 </#fulltime>
  <#parttime> 100 </#parttime>
  <#union> 300 </#union>
  <#nonunion> 200 </#nonunion>
</#foreignEmployees>

Sample data mapped to domain model:
<employee family container>
  <head sibling 3>
  <employees>
    <#domestic>
      <#fulltime>800</#fulltime>
Sample data extracted by AIG from domain model:
<#domesticEmployees>
  <#union> 700 </#union>
  <#nonunion> 300 </#nonunion>
</#domesticEmployees>
<#foreignEmployees>
  <#union> 300 </#union>
  <#nonunion> 200 </#nonunion>
</#foreignEmployees>

Sample data extracted by Travelers from domain model:
<#employees> 1500 </#employees>

Sample data extracted by Hartford from domain model:
<#domesticEmployees> 800 </#domesticEmployees>
<#foreignEmployees> 400 </#foreignEmployees>
Period 4 (changes in Hartford’s data requirement)

Figure 10.10. Mappings to the hybrid domain model for period 4. “Head sibling 2” is in a dotted square because it is no longer being directly used by any carrier. The aggregator has the option of mapping “head sibling 3” to “head sibling 1” and eliminate “head sibling 2”.

Partial data schemas

Carrier: Hartford
Partial data schema:  
<#domesticEmployees>
   <#union>
   <#nonunion>
<#foreignEmployees>
   <#union>
   <#nonunion>
Context: full time employees only; includes both union and nonunion employees

Carrier: AIG
Partial data schema:  
<#domesticEmployees>
   <#union>
   <#nonunion>
<#foreignEmployees>
   <#union>
   <#nonunion>
Context: full time and part time employees
Carrier: Travelers
Partial data schema: 
Context: full time and part time employees; includes both union and nonunion employees

Aggregator
Partial data schema:

<#employees>
  <#domesticEmployees>
    <#fulltime>
      <#union>
      <#nonunion>
    <#parttime>
      <#union>
      <#nonunion>
  <#foreignEmployees>
    <#fulltime>
      <#union>
      <#nonunion>
    <#parttime>
      <#union>
      <#nonunion>

Incomplete Hybrid Domain Model:

<head Sibling 1>
<meta active = yes, usage_cnt = 1, last_mod_date=4/1/2001>
  <#employees>
    <#domestic>
    <#foreign>

<head sibling 2>
<meta active = yes, usage_cnt = 1, last_mod_date=4/7/2001>
  <#employee>
    <#domestic>
    <#fulltime>
    <#parttime>
    <#foreign>
    <#fulltime>
    <#parttime>

<head sibling 3>
<meta active = yes, usage_cnt = 2, last_mod_date=4/14/2001>
  <#employee>
    <#domestic>
    <#fulltime>
    <#parttime>
    <#union>
    <#nonunion>
Conversion Axioms within the domain model:

<rule set 1>
<meta active = yes, usage_cnt = 1, last_mod_date=4/7/2001>
<head sibling 1><employees><domestic> =
<head sibling 2> <employees><domestic><fulltime>
+ <head sibling 2> <employees><domestic><parttime>
<head sibling 1><employees><foreign> =
<head sibling 2> <employees><foreign><fulltime>
+ <head sibling 2> <employees><foreign><parttime>

<rule set 2>
<meta active = yes, usage_cnt = 1, last_mod_date=4/14/2001>
<head sibling 2> <employees><domestic><fulltime>
= <head sibling 3> <employees><domestic><fulltime>
<head sibling 2> <employees><domestic><parttime>
= <head sibling 3> <employees><domestic><parttime>
<head sibling 2> <employees><foreign><fulltime>
= <head sibling 3> <employees><foreign><fulltime>
<head sibling 2> <employees><foreign><parttime>
= <head sibling 3> <employees><foreign><parttime>
<rule set 3>
<meta active = yes, usage_cnt = 1, last_mod_date=4/28/2001>
<head sibling 3> <#employees><#domestic><fulltime>
  = <head sibling 4> <#employees><#domestic><fulltime><union>
  + <head sibling 4> <#employees><#domestic><fulltime><nonunion>
<head sibling 3> <#employees><#domestic><parttime>
  = <head sibling 4> <#employees><#domestic><parttime><union>
  + <head sibling 4> <#employees><#domestic><parttime><nonunion>
<head sibling 3> <#employees><#foreign><fulltime>
  = <head sibling 4> <#employees><#foreign><fulltime><union>
  + <head sibling 4> <#employees><#foreign><fulltime><nonunion>
<head sibling 3> <#employees><#foreign><parttime>
  = <head sibling 4> <#employees><#foreign><parttime><union>
  + <head sibling 4> <#employees><#foreign><parttime><nonunion>

Aggregator --> Domain Model elevation axioms
</#domesticEmployees><fulltime><union>
  -- > <head sibling 4> <#employees><#domestic><fulltime><union>
</#domesticEmployees><fulltime><nonunion>
  -- > <head sibling 4> <#employees><#domestic><fulltime><nonunion>
</#domesticEmployees><parttime><union>
  -- > <head sibling 4> <#employees><#domestic><parttime><union>
</#domesticEmployees><parttime><nonunion>
  -- > <head sibling 4> <#employees><#domestic><parttime><nonunion>
</#foreignEmployees><fulltime><union>
  -- > <head sibling 4> <#employees><#foreign><fulltime><union>
</#foreignEmployees><fulltime><nonunion>
  -- > <head sibling 4> <#employees><#foreign><fulltime><nonunion>
</#foreignEmployees><parttime><union>
  -- > <head sibling 4> <#employees><#foreign><parttime><union>
</#foreignEmployees><parttime><nonunion>
  -- > <head sibling 4> <#employees><#foreign><parttime><nonunion>

Hartford --> Domain Model elevation axioms
</#domesticEmployees><union>
  -- > <head sibling 4> <#employees><#domestic><fulltime><union>
</#domesticEmployees><nonunion>
  -- > <head sibling 4> <#employees><#domestic><fulltime><nonunion>
</#foreignEmployees><union>--> <head sibling 4><#employee><#foreign><fulltime><union>
</#foreignEmployees><nonunion>
  -- > <head sibling 4><#employee><#foreign><fulltime><nonunion>

81
AIG --> Domain Model elevation axioms
<#domesticEmployees><union> --> <head sibling 3><#employee><#domestic><union>
<#domesticEmployees><nonunion> --> <head sibling 3><#employee><#domestic><nonunion>
<#foreignEmployees><union> --> <head sibling 3><#employee><#foreign><union>
<#foreignEmployees><nonunion> --> <head sibling 3><#employee><#foreign><nonunion>

Travelers --> Domain Model elevation axioms
<#employees> --> <head sibling 1><#employee><#domestic> + <head sibling><#employee><#foreign>

Application to sample data

Sample data from Aggregator:
<#domesticEmployees>
  <#fulltime>
    <#union> 600 </#union>
    <#nonunion> 200 </#nonunion>
  </#fulltime>
  <#parttime>
    <#union> 100 </#union>
    <#nonunion> 100 </#nonunion>
  </#parttime>
</#domesticEmployees>
<#foreignEmployees>
  <#fulltime>
    <#union> 350 </#union>
    <#nonunion> 50 </#nonunion>
  </#fulltime>
  <#parttime>
    <#union> 50 </#union>
    <#nonunion> 50 </#nonunion>
  </#parttime>
</#foreignEmployees>

Sample data mapped to domain model:
<employee family container>
  <head sibling 3>
  <#employees>
    <#domestic>
      <#fulltime>
        <#union> 600 </#union>
        <#nonunion> 200 </#nonunion>
      </#fulltime>
      <#parttime>
        <#union> 100 </#union>
        <#nonunion> 100 </#nonunion>
      </#parttime>
    </#domestic>
Sample data extracted by AIG from domain model:

```xml
<#domesticEmployees>
  <#union> 700 </#union>
  <#nonunion> 300 </#nonunion>
</#domesticEmployees>
<#foreignEmployees>
  <#union> 300 </#union>
  <#nonunion> 200 </#nonunion>
</#foreignEmployees>
```

Sample data extracted by Travelers from domain model:

```xml
<#employees> 1500 </#employees>
```

Sample data extracted by Hartford from domain model:

```xml
<#domesticEmployees>
  <#union> 600 </#union>
  <#nonunion> 200 </#nonunion>
</#domesticEmployees>
<#foreignEmployees>
  <#union> 350 </#union>
  <#nonunion> 50 </#nonunion>
</#foreignEmployees>
```

By using the COIN-IE approach, the amount of work required for the aggregator and the carriers are reduced by an order of N. When new data requirements force the domain model to change, the carriers that have not changed their data schema do not have to change their mappings to the domain model at all. The aggregator is not required to create new mappings for these carriers.
either. It only has to map to the new “head sibling” in the domain model and add conversion rules that map the new “head sibling” to the former “head sibling”. Thus scalability is achieved.
11.0. Conclusions

Internet technologies have created disruptive changes to the insurance brokerage industry by enabling insurance carriers to sell directly to customers online without any help from the brokerage firms. To counter this threat, many insurance brokerages are implementing online aggregators that improve the customers’ insurance shopping experience online. The implementation of these aggregators, however, is hindered by system incompatibility and data heterogeneity issues.

The effectiveness of popular system integration and interoperability solutions, which include XML, XSLT, SOAP, Biztalk, and data standardization, was evaluated. XML and XSLT, which are excellent candidates for pair-wise solutions, do not scale when there are many web services and frequent changes to their data schemas. SOAP and Biztalk place heavy emphasis on document delivery, but avoid addressing data heterogeneity issues. Data standardization, which can be effective for applications where changes in data schema are rare, is not flexible enough to handle the frequent changes to the web services’ data schemas.

While many of the syntactic and even semantic data incompatibility problems can be manually identified and manually reconciled, fixing these problems manually is not a scalable or long term solution. The scalability example in chapter eight clearly illustrated that manually changing a system’s software code to solve data heterogeneity problems caused by changes in the data requirements over time is not a scalable solution.

A better approach for dealing with changes in data requirements over time, which is called COIN-IE, is proposed. COIN-IE uses concepts from COIN for managing the mappings of the aggregator’s data schema to the carriers’ data schemas. The value added by the COIN-IE framework is that it allows the domain model to be incomplete, and it allows the domain model to change over time without forcing the carriers to change their mappings to the domain model. Thus scalability is achieved.
12.0 Bibliography


