December 2006

Modeling XBRL-based Applications with UML: Developing Balanced-Scorecard Management Appraisal Systems

Joseph Callaghan
Oakland University

Robert Nehmer
Oakland University

Vijayan Sugumaran
Oakland University

Modeling XBRL-based Applications with UML: Developing Balanced-Scorecard Management Appraisal Systems

Joseph H. Callaghan
Department of Accounting and Finance
School of Business Administration
Oakland University
Rochester, MI 48309
callaghan@oakland.edu

Robert Nehmer
Department of Accounting and Finance
School of Business Administration
Oakland University
Rochester, MI 48309
nehmer@oakland.edu

Vijayan Sugumaran
Department of Decision and Information Sciences
School of Business Administration
Oakland University
Rochester, MI 48309
sugumara@oakland.edu

ABSTRACT

This paper focuses on object modeling of eXtensible Business Reporting Language (XBRL) applications using Unified Modeling Language (UML) diagrams in the context of developing management appraisal systems based on balanced scorecard concepts. An overview of theses technologies and the benefits associated with their integration is explored. The potential offered by reverse-engineered XBRL financial reporting (FR) and general ledger (GL) taxonomies and schemas into UML diagrams is demonstrated. The class attributes and implied relationships reflecting XBRL elements and relationships is exploited in the development of the methods necessary to implement a management appraisal application using balanced scorecard concepts. In so doing, both hierarchical and process-oriented structures implied by object modeling are utilized. The usefulness of these models and techniques is demonstrated in using financial, customer, internal process and learning aspects of the balanced scorecard system.

Keywords

XBRL, UML, OO-Modeling, Balanced Scorecard, Application Development

INTRODUCTION

The eXstensible Business Reporting Language (XBRL) is an application of the eXstensible Markup Language (XML). As such it provides the XML features of operating system independence, semantically meaningful tag sets, with structures, attributes and elements that facilitate human understanding through its self-describing nature (Fox et al., 2001; Chang and Jarvenpaa, 2005). Further, through its well-formedness and possible validation against schemas, control over data is enhanced. Finally, the underlying data of instance documents can be of higher integrity and be processed effectively by querying technologies and software applications. Many of the benefits associated with this technology will be realized through the developments of applications and methods that avail themselves of the language features (Fisher and Fisher, 2001).

As suggested by its name, XBRL applies to the domain of business reporting. It consists of two subsets: XBRL Financial Reporting (FR) and XBRL General Ledger (GL). The former taxonomy covers the external financial reporting area, including the production of financial statements and their disclosures under many possible systems of Generally Accepted Accounting Principles (GAAP) (Deshmukh, 2004). The external users of financial statements are its intended consumers, including current and prospective creditors and owners, financial analysts, and governmental regulators. The latter taxonomy covers the area from financial-based transactions to aggregations constituting a possibly virtual General Ledger, a summary of certain business activities (Doolin and Troshani, 2004).

XBRL FR users may develop software applications that analyze financial statement data both cross-sectionally, across a set of companies, at a given point in time and temporally for a given company, or in combination (Naumann, 2004). These types of financial statement analyses will be greatly facilitated by regulatory mandates requiring the use of XBRL FR, such as a SEC mandate for 10-Q and 10-K filings for the registered securities of companies within its jurisdiction (Malhotra and Garritt, 2004). Web-querying of publicly available XBRL documents could form the inputs necessary for the development of sophisticated applications availing themselves of the inherent structure and validity of the underlying documents. Similarly, application development mining the vast amount of documents tagged by XBRL GL is possible now for companies deciding to tag its documents according to this or its inherently extensible schema. One can imagine, for instance, the development of sophisticated, real-time, and continuously running internal audit software (Hannon, 2005).

XBRL provides a common format for sharing financial information across applications. This impacts how business data is collected, controlled, analyzed and reported to external entities. Particularly, it facilitates the process link between internal business reporting and external financial reporting. Traditionally, when financial data is moved from its origin to other applications, its context in terms of definition, currency, period, entity etc. is lost (Willis and Hannon, 2005). Business rules that are used to process this data are applied on an ad hoc basis. This is primarily due to the fact that these applications are data centric and the operational and the semantic knowledge is not included along with the data. However, if the process knowledge is encapsulated with it, target applications can utilize the financial data in a more meaningful way and also provide adequate control and audit trail. Hence, we contend that the data sharing facilitated by XBRL should be augmented with sharing of the corresponding semantic and process knowledge.

As XBRL gets widely adopted, organizations will generate lots of XBRL instance documents that need to be processed by various applications. In order to make effective use of these instance documents, sharing of contextual information and processing knowledge has to take place in a coherent manner (Rezaee and Turner, 2002). One approach that holds potential is to use a standard modeling technique to combine the financial data along with the process knowledge and make that available to the application. In other words, the instance document can be augmented with the process knowledge by abstracting the main elements and creating objects and embedding process knowledge into these objects. This object oriented model can then be used by an existing application in performing a particular task. More interestingly, this model can also be used to develop new XBRL applications. The modeling approach that naturally lends itself in accomplishing this goal is the UML modeling technique. Hence, *the objective of this research is to develop a methodology for modeling XBRL-based applications with UML and use this model to develop target applications.* The contribution of this research is a framework that can be used to reverse engineer XBRL instance documents and add process knowledge to improve their use.

PROPOSED METHDOLOGY

For the development of target applications, we propose a methodology that uses an object-oriented approach. Figure 1 depicts our framework for XBRL application development using Uniform Modeling Language (UML). The proposed approach consists of the following three main phases: a) reverse engineer XBRL instance documents to create UML model, b) augment the UML model with additional process knowledge, and c) application generation from augmented UML models. Each phase is briefly described below.

Reverse Engineering XBRL Documents

Among other things, financial statement element analysis can be properly conducted when additional semantics provided by disclosures are incorporated into the process (Sugumaran et al., 2002). UML provides a conceptual framework to capture more meaningful semantics, particularly among related, collaborating objects. For example, there is a close interaction between XBRL FR and XBRL GL components. UML diagrams permit the modeling of these natural interactions. These diagrams in turn can provide a schema for augmenting XBRL documents that then can be better processed by downstream users, including intelligent agents, and applications.

UML can be used to not only represent the structure and behavior of systems, but also to model various aspects of domain artifacts at a high level (Carlson, 2001). For example, in the financial reporting domain, there are several types of reporting documents that exist with well-defined components and structures. These components are also interrelated, with specific processing implications. It is essential that these relationships are understood and captured to facilitate the automated processing of financial data contained in these documents. UML naturally lends itself in modeling the structural components of financial reporting documents, the relationships between them, and how these components are utilized in financial analysis and how they impact one another in various contexts. Thus, modeling XBRL instance documents using UML would help enhance the semantic content of the document. This semantic augmentation can be exploited by the applications that use these documents for better financial analysis.

We propose that XBRL instance documents and schemas be modeled using UML constructs. In particular we propose that both XBRL FR and XBRL GL be reverse-engineered and initially represented by UML Class diagrams, reflecting the structures, attributes, elements and relationships inherent in these schemas. XBRL instance documents typically have elements such as item, context, tuple and group. A class is created for each of these element types and the relationships that exist between them are captured in the class diagram. These elements also contain one or more attributes. For example, the context element contains the following attributes: id, period, unit, precision, cwa attribute (closed world assumption attribute), entity sub-element, and scenario sub-element. The attributes are represented as instance variables of the class.

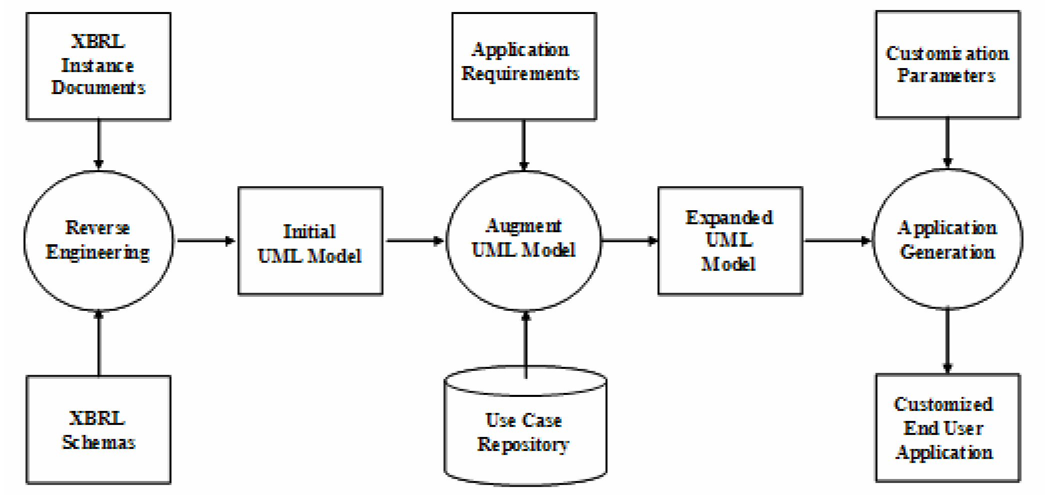

Figure 1. Framework for XBRL Application Development using UML Model

Augment UML Model

In the initial phase, classes are created corresponding to elements of the XBRL FR and XBRL GL instance documents. These classes contain only instance variables and the actual values. However, they do not contain any process knowledge, which is essential for performing various operations on these data elements. A UML class is a container of structural and behavioral features, but only the structural features have been modeled so far corresponding to the XBRL vocabulary. The dynamic behavior and processing knowledge needs to be added at this point. Thus, the initial UML model is augmented with additional information. To start with, the processing knowledge is added by defining appropriate methods for the classes. Once so represented, the methods implied by the target application are discovered and represented by various Use-Case diagrams stored in a repository. Based on the target application domain requirements, the initial UML model is also augmented with Activity Diagrams, Communication Diagrams and Sequence Diagrams to reflect the dynamic behavior of applications. Activity diagrams are typically used for business process modeling, for modeling the logic captured by a single use case or usage scenario, or for modeling the detailed logic of a business rule. In many ways UML activity diagrams are the object-oriented equivalent of flow charts and data flow diagrams (DFDs) from structured development. The communication diagram shows instances of classes, their interrelationships, and the message flow between them. Communication diagrams typically focus on the structural organization of objects that send and receive messages. Sequence diagrams demonstrate the behavior of objects in a use case by describing the objects and the messages they pass. It captures the sequential logic, i.e., the time ordering of messages between objects. Thus, the initial UML model derived from the XBRL instance documents is augmented with behavioral information. From this augmented UML model, a specific application could be generated.

Application Generation

XBRL instance documents are rendered using stylesheets and several tools exist that create reports in a WYSIWYG format. Similarly, tools have been developed to simplify the process of creating XBRL documents and taxonomies. These tools provide the opportunity for even small organizations to develop XBRL-compliance. However, one of the major challenges has been the lack of software applications that build XBRL into the core fabric of the program so that users can create data streams of marked-up XBRL directly into internal business reporting systems such as digital dashboards or scorecard systems. In other words, the XBRL instance documents can be tightly integrated into various applications and provide seamless data as well as process interoperability. In our approach, the augmented UML models capture the additional process knowledge as well as dynamic behavior so that a customized application can be generated from these models.

The expanded UML models, coupled with customization parameters, would form the logical constructs needed to generate (through forward-engineering, code-generation) customized applications. Such applications would be documented and maintained within this framework. If schemas were changed or extended, or user requirements evolved, or customization parameter redefined, then application maintenance and regeneration would be fairly straightforward.

EXAMPLE OF ADDING METHODS

XBRL FR and XBRL GL provide tags for marking up the content of financial reports and their supporting detail respectively. While there is a good deal of structure in the specifications, neither was explicitly designed to deal with performance measurement applications in mind. To show how object oriented modeling can be used to create generic FR and GL applications, we develop a prototype performance measurement system. The application is developed with two common performance objectives in mind: tracing responsibility hierarchically through the organizational structure and tracing process flows across the organizational structure. In order to do this, and to keep the notation at a reasonable level for the purposes of this paper, we will use a psuedo-tagset of both FR and GL. For example, operating profit is represented in a FR instance document as:

```
<usfr-pt:OperatingProfit numericContext="P3M1QFY06">48000000</usfr-pt:OperatingProfit>
```

We will use the simplified version <OperatingProfit>48000000</OperatingProfit> or even <OperatingProfit> to refer to the tag pair in the method notations when they are introduced. The full tag details can be substituted back into an implemented system with no loss of generality. Further, we do not specify the FR or GL unique identifiers, FR ID and GL ID, in order to keep the examples as simple and clear as possible. In the discussion that follows, we assume that instance documents have already been created. These are the same instance documents which were used to produce the class diagrams discussed in the previous section.

Our first example is the hierarchical tracing of responsibility from operating profit and revenues in the income statement, through divisional and departmental revenues. The relevant FR tags are:

```
<OperatingProfit>
<SalesRevenueNetGoods>
<SalesRevenueGrossGoods>
```

Since FR does not provide a way to follow the responsibility for accounts, we propose a method which will allow us to trace this through the FR tags. The general form of the method is:

```
[FR ID:tag, FR:tag]
```

where, FR ID is a unique reference to the FR instance document and tag is the FR tag. So there would be two simple methods in this example:

```
[FR ID:<OperatingProfit>, FR ID:<SalesRevenueNetGoods>]
[FR ID:<SalesRevenueNetGoods>, FR ID:<SalesRevenueGrossGoods>]
```

This completes the hierarchy in FR. In order to follow responsibility into the detailed accounts, we reference an instance document which summarizes divisional gross sales revenue for goods into the FR tag SalesRevenueGrossGoods. Each division has an entry detail section in this instance document similar to the following for the East Shore division:

```
<entryDetail>
    <account>
        <accountMainID >10200</accountMainID>
        <accountMainDescription>Acme, East Shore Revenues </accountMainDescription>
    </account>
    <debitCreditCode>C</debitCreditCode>
    <amount>345000</amount>
</entryDetail>
```

Currently in FR and GL, there is no method to tie the GL instance document with the entry detail for the FR SalesRevenueGrossGoods tag. In order to allow the application to follow the responsibility hierarchy through the instance documents, we create methods which have the following structure:

[FR ID:tag, GL ID:tagset]

where, FR ID is a unique reference to the FR instance document, tag is the FR tag, GL ID is a unique reference to the GL instance document, and tagset is the GL instance detail tagset. So the method would look like:

[FR ID:<SalesRevenueGrossGoods>, GL ID:entryDetail>]

In order to follow responsibility down one more level, we reference an instance document at a further level of detail. Each subdivision has an entryDetail tag section and amount, similar to the following for the Maine subdivision:

```
<entryDetail>
    <account>
        <parentSubaccountCode >10207</parentSubaccountCode >
        <accountMainDescription>East Shore, Maine Revenues </accountMainDescription>
    </account>
    <debitCreditCode>C</debitCreditCode>
    <amount>64000</amount>
</entryDetail>
```

We need a method to trace responsibility down to this level of detail. Its general structure is:

[GL ID:<accountMainID >, GL ID:<entryDetail>]

where, GL ID is a unique reference to the GL instance document, <accountMainID> is the GL tag in the main instance, and <entryDetail> is the GL instance detail tagset.

This completes the construction of the methods for tracing responsibility through FR and GL instance documents in a hierarchical manner. We now consider tracing responsibility in a balanced scorecard environment using GL. Our example begins where the hierarchical example left off, with the revenue of a subdivision. In order to keep the example tractable, we consider only the output side of the balanced scorecard and ignore outcomes. We note however, that outcome measures are available in the GL tagset and can also be created as extensions to that tagset. The methods used in either outputs or outcomes will be very similar.

What we need to do is to show output measures for customer, financial, internal business processes and learning and growth specifically traceable to the revenue of a subdivision. We take the customer and financial components together since they are highly related within the structure of GL. We consider one possible structure from an instance document which details customer invoices and payments received from a particular customer over a period. This structure would look similar to the following:

```
<entryDetail>
    <account>
        <accountMainID >31005</accountMainID>
        <accountMainDescription>Hampton Company, Sales </accountMainDescription>
    </account>
    <debitCreditCode>C</debitCreditCode>
    <amount>15000</amount>
    <account>
        <accountMainID >31005</accountMainID>
        <accountMainDescription>Hampton Company, Sales </accountMainDescription>
    </account>
    <debitCreditCode>C</debitCreditCode>
    <amount>8000</amount>
    <account>
        <accountMainID >31005</accountMainID>
        <accountMainDescription>Hampton Company, Payment </accountMainDescription>
    </account>
```

```
          <debitCreditCode>D</debitCreditCode>
          <amount>22500</amount>
     </entryDetail>
```

We only need to create a method to trace the subdivision revenue down to this level of detail. Its general structure is again:

[GL ID:tag, GL ID:tagset]

where GL ID is a unique reference to the GL instance document, tag is the GL tag, and tagset is the GL instance detail tagset.

For the internal business process, we already have a tagset for the subdivision revenues. What a performance system might need to do is tie this with the relevant cost of goods sold or finished goods inventory. Here we will create a linkage back to the goods before they were sold, while they were still in finished goods inventory. Consider an instance document or documents which details this inventory listing. We can again link into an entry detail structure in this instance. Consider the tagset:

```
<entryDetail>
     <account>
          <accountMainID >10340</accountMainID>
          <accountMainDescription>Winter Coats</accountMainDescription>
     </account>
     <debitCreditCode>D</debitCreditCode>
     <amount>29000</amount>
     <account>
          <accountMainID >10340</accountMainID>
          <accountMainDescription>Winter Coats</accountMainDescription>
     </account>
     <debitCreditCode>D</debitCreditCode>
     <amount>7000</amount>
     <account>
          <accountMainID >10340</accountMainID>
          <accountMainDescription>Winter Coats</accountMainDescription>
     </account>
     <debitCreditCode>D</debitCreditCode>
     <amount>16000</amount>
</entryDetail>
```

We can create the familiar method [GL ID:tag, GL ID:tagset] from the revenue of this subdivision to the finished goods detail. In a similar way, we can drill back to production, raw materials, and vendors. All this assumes that the necessary GL instance documents are available.

Finally, consider creating methods to trace a learning and growth output measure through GL instance documents. One possibility is to track performance over time, using revenue for a product line. GL instance documents have tags which can be used to uniquely identify the documents they are derived from. As an example, consider the following tagset:

```
<documentNumber>2072</documentNumber>
<documentReference>Sales Report#: 3905 </documentReference>
<documentType >Sales Report</documentType>
<documentDate>2006-3-31</documentDate>
<amount>23500</amount>
```

This could represent a monthly sales report total. We can construct a method to create a quarterly report, showing sales trends, as follows:

[GL ID:2070, GL ID:2071, GL ID:2072]

where, GL ID is a reference to the instance document and 2070, 2071, and 2072 are the sales reports for January, February, and March, respectively.

CONCLUSION

This paper has considered using XBRL instance documents to create class diagrams. Further, it has developed an object oriented design approach to implementing applications based on XBRL by constructing methods of implementing linkages between XBRL instance documents. These links are useful in creating both hierarchical and non-hierarchical performance measuring applications. Further work with this technique could produce methods to trace transaction execution through a series of applications. This would be of tremendous value in auditing applications and tracing compliance to internal controls, such as is now required under Sarbanes Oxley. Our future work also involves exploring XBRL application development using Service Oriented Architecture (SOA). Web Services can push company information using XBRL format, which can be immediately consumed by analytical applications or viewing tools. Thus, web services enabled reporting is an enabler for the concepts reported in the balanced scorecard and value reporting models.

REFERENCES

1. Carlson, D. (2001) Modeling XML Applications with UML: Practical e-Business Applications, Addison-Wesley Professional, Boston, MA.
2. Chang, C., Jarvenpaa, S. (2005) "Pace of Information Systems Standards Development and Implementation: The Case of XBRL," Electronic Markets, Vol. 15, No. 4; pp. 365 – 377.
3. Deshmukh, A. (2004) "XBRL," Communications of the AIS, Vol. 13, Article 16, pp. 196 – 219.
4. Doolin, B., Troshani, I. (2004) "XBRL: A Research Note," Qualitative Research in Accounting and Management, Vol. 1, No. 2, pp. 93-104.
5. Fisher, D., Fisher, S. (2001) "Integrating Financial Reporting Systems with XBRL," *Proceedings of the 7th Americas Conference on Information Systems*, Boston, Massachusetts, August 3-5, pp. 636 – 541.
6. Fox, T., Durler, M.G., Swanson, Z., Hindi, N., Remington, W. (2001) "Examining a New Approach: XBRL Applications – A Case Study," *Proceedings of the 7th Americas Conference on Information Systems*, Boston, Massachusetts, August 3-5, pp. 507 – 509.
7. Hannon, N. (2005) "Post Sarbanes-Oxley: Does XBRL Hold the Key?" *Strategic Finance*, Vol. 86, No. 7, pp. 57 – 59.
8. Malhotra, R. and Garritt, F. (2004) "Extensible Business Reporting Language: The Future of E-Commerce-driven Accounting," *International Journal of Business*, Vol. 9, No. 1. Available at SSRN: http://ssrn.com/abstract=490444
9. Naumann, J.W. (2004) "Tap Into XBRL's Power the Easy Way," *Journal of Accountancy*, Vol. 197, No. 5, pp. 32 – 39.
10. Rezaee, Z., Turner, J.L. (2002) "XBRL-based financial reporting: Challenges and opportunities for government accountants," *The Journal of Government Financial Management*, Vol. 51, No. 2, pp. 16 - 21.
11. Sugumaran, V., Callaghan, J., Savage, A. (2002) "Modeling XBRL Using UML: Improving Semantics for Financial Analysis," *Proceedings of the 8th Americas Conference on Information Systems*, Dallas, TX, August 9 – 11, pp. 7 – 12.
12. Willis, M., Hannon, N.J. (2005) "Combating Everyday Data Problems with XBRL," *Strategic Finance*, Vol. 87, No. 1, pp. 57 – 59.