Association for Information Systems
AIS Electronic Library (AISeL)

AMCIS 2005 Proceedings

Americas Conference on Information Systems (AMCIS)

2005

An XML Based Architecture for Sharing Heterogeneous Models in Web and Distributed Computing Environments

Omar El-Gayar
Dakota State University, omar.el-gayar@dsu.edu

Kanchana Tandekar
Dakota State University, tandek@pluto.dsu.edu

An XML-based architecture for sharing heterogeneous models in Web and Distributed Computing Environments

Omar El-Gayar
College of Business and Information Systems
Dakota State University
omar.el-gayar@dsu.edu

Kanchana Tandekar
College of Business and Information Systems
Dakota State University
tandek@pluto.dsu.edu

ABSTRACT

Model management emerged in the mid-seventies in the context of managing models in decision support systems (DSS). With the recent advances in computer and telecommunication technologies, organizations are ever increasingly dependent on management models for data analysis and decision support. Accordingly, the number and complexity of management models and of modeling platforms dramatically increased rendering such models a corporate (and national) resource. With the advent of the Web and distributed computing environments, there is also an increasing demand to share these often heterogeneous models over corporate intranets as well as the Web.

To this end, this paper presents an XML-based architecture for sharing heterogeneous models in Web and distributed computing environments. The architecture includes an XML schema for representing models. The schema is based on the structured modeling paradigm as a formal mathematical environment for conceiving, representing and manipulating a wide variety of models. The architecture allows different types of models, developed in a variety of modeling platform to be represented in a standardized format and shared over the Web. The paper demonstrates the proposed architecture through a case study.

Keywords

Model management, Structured modeling, Model sharing, XML, Distributed computing environments.

INTRODUCTION

With the recent advances in computer and telecommunication technologies, organizations are increasingly dependent on management models for data analysis and decision support. Accordingly, the number and complexity of management models and of modeling platforms dramatically increased rendering such models a corporate (and national) resource. With the advent of the Web and distributed computing environments, there is also an increasing demand to share these often heterogeneous models over corporate intranets as well as the Web.

Nevertheless, models are platform-dependent. Accordingly, it is usually not possible to openly exchange models. As stated by Krishnan and Chari (2000), there are problems with the DSS concept of managing models. Firstly, much time and effort is required to implement and interface models with various modeling platforms. Secondly, the model representation used by these platforms lack abstractions or details required for communicating models effectively. All these problems made it difficult to verify, validate, reuse, share, and maintain models.

In the quest for expressive model representation, structured modeling (SM) received a great deal of attention in the literature. SM as defined by Geoffrion (1987) is a "formal mathematical and computer-based environment for conceiving, representing and manipulating a wide variety of models". SM realizes many of the features desired from model management systems which make it a very useful tool for model representation. SM provides a coherent conceptual framework for modeling based on a single modeling system, irrespective of the underlying modeling paradigm.

Moreover, the recent development of eXtensible Markup Language (XML) emphasized the importance of content information by making it possible for designers to create and manage their own sets of tags (Bradley, 2002). Accordingly, XML facilitates searching for specific content-based information as well as moving documents across applications and systems, i.e., model exchange in a distributed environment. Modelers using different modeling tools or environments can

communicate using the common XML representation. While Geoffrion (1987) demonstrated different ways of rendering a structured model as a web document, not much research has been done to represent structured models using XML.

It is the objective of this research to leverage the inherent synergy between SM and XML to facilitate model sharing in a distributed environment. This is accomplished through the development of an XML-based architecture. At the core of the architecture is an XML schema for a new markup language referred to as the Structured Modeling Markup Language (SMML) for representing models. The schema is based on the structured modeling paradigm as a formalism for conceiving, representing and manipulating a wide variety of models. In effect, the proposed architecture allows for:

- Representing different types of models that are developed using in a variety of modeling platforms in a standardized format.
- Sharing and publishing models among model users irrespective of the modeling environment used by these users.
- Using heterogeneous models (developed in different modeling environments and for different solvers) without the need for re-writing models for each tool.
- Creating a lifetime repository (archive) of models in an environment and a platform independent format. Accordingly, the models are reusable, even after a particular environment is rendered obsolete.

The paper is organized as follows: the next section provides a brief review model management, structured modeling, and XML. Next, we present a description of the proposed architecture with particular emphasis on SMML followed by a case study demonstrating the applicability of the proposed system. The final section concludes the paper.

MODEL MANAGEMENT AND STRUCTURED MODELING

Model management emerged in the mid-seventies in the context of managing models in decision support systems (DSS) (Will, 1975; Sprague & Watson, 1975). While a comprehensive review of the model management (MM) literature can be found elsewhere (Krishnan and Chari, 2000; Blanning, 1993; and Chang et al., 1993), it is worth noting that much of the motivation behind MM focused around finding ways for developing, storing, manipulating, controlling, and effective utilization of models in an organization (Muhanna, 1993). Inherent in such functionality is the ability to represent models at a higher lever of abstraction, i.e., meta-modeling. In that regard, several frameworks are proposed in the literature including structured modeling (Geoffrion, 1987), logic-based modeling (Bhargava and Kimbrough, 1993), graph grammar (Jones, 1990), object-oriented modeling (Lenard 1993; Muhanna, 1993), and frame-based modeling (Binbasioglu and jarke, 1986) as well as modeling languages including SML (Geoffrion 1992a&b), GAMS (Brooke et al., 1988), AMPL (Fourer, Gay, and Kernighan, 1993), and LINGO (Katz, Risman, and Rodeh, 1980).

Among the aforementioned frameworks and associated languages, SM is particularly attractive. Specifically, SM has many features that are highly desirable from a MM perspective (Krishnan and Chari, 2000; Geoffrion, 1987), most notably: independence of model representation and model solution, sufficient generality to encompass a wide variety of modeling paradigms, and representational independence of general model structure and the detailed data needed to describe specific model instances (Geoffrion, 1987). Moreover, SM offers distinct advantage when it comes to model integration. Specifically, SM allows for testing if a given structure (model) is a valid structure and for assessing the impact of a change to a model as it is integrated with another model (Krishnan and Chari, 2000). Last, but not least, SM is compatible with two important developments in computing, namely, object-oriented programming (Lenard, 1993; Muhanna, 1993) and XML. All this makes SM an inevitably useful tool and formal foundation for any modeling environment.

In SM, a model is defined as a combination of a model schema and one or more model instances. A model schema describes the general structure of a model and is represented as a hierarchically, acyclic, and attributed graphs. A model schema may be associated with one or more model instances. Model instances correspond to the data part of the model. Detailed description of SM concepts can be found in (Geoffrion, 1987; Geoffrion, 1989; Geoffrion, 1992a&b).

XML AND MODELING

XML is a meta-language originally developed by the World Wide Web Consortium (W3C) as a simplified subset of SGML. In effect, XML seeks to facilitate the exchange of data by supporting non-proprietary data format. The tremendous growth of XML applications (XML-based markup languages) for representing and exchanging data in various problem domains (e.g., Chemical ML, MathML, Molecular Dynamics Markup Language) is a testimony to XML's success.

Moreover, XML success has also extended to representing models in various problem domains and for specific modeling paradigms. Hucka, Finney, Sauro, Bolouri, Doyle, and Kitano (2003) and Finney and Hucka (2003) describe a Systems

Biology Markup Language (SBML) for representing and exchanging biochemical network models between simulation/analysis tools. In data mining, the Predictive Model Markup Language (PMML) provides a tool independent mechanism for representing and sharing predictive models such as regression models, cluster models, trees, neural networks, and Bayesian models among compliant vendors (Data Mining Group, 2004). With respect to simulation, Canonico, Emma, and Ventre (2003) presents an XML application for describing generic network scenarios as well as the process for translating the scenarios into a simulation script for a network simulator. Wang and Lu (2002) develop an XML application to represent discrete event simulation models based on DEVS (Discrete Event System Specification) approach (Zeigler 1990), while Lu, Qiao, and Mclean (2003) and Qiao, Raddick, and McLean (2003) discuss cases utilizing an XML-based simulation interface specification being developed by the National Institute of Standards and Technology (NIST).

Nevertheless, only Kim (2001) provides an XML-based markup language that is based on a modeling formalism thereby potentially realizing many of the desirable MM features mentioned earlier. The proposed language OOSML (Object-Oriented Structured Markup Language) is a pioneering effort in this area with some significant drawbacks. Firstly, OOSML utilizes XML Document Type Definition (DTD) which is becoming obsolete and incapable of representing complex structures. Moreover, the DTD presented lacks support for representing mathematical equations and explicit indexing. The instance representation was also not generic in nature and it is not possible to validate an instance file and to enforce the rules for writing a good model instance document.

In this paper, we take the work done by Kim (2001) a step further. This research tries to overcome the shortcomings and provides a more general and complete view for integrating SM and XML. This research uses XML schemas for model representation thereby leveraging the richness of data types, extensive support for name spaces and other advantages of schemas over DTDs (Evans, Kamanna, and Mueller, 2002; Bradley, 2002), and MathML (W3C, 2003) for representing equations. It provides support for indexes and indexsets. There is a separate schema to validate a model instance and enforce rules that checks for consistency of data.

THE SYSTEM ARCHITECTURE

There are two components to the architecture of the system: The server component and the client component. Figure 1 shows a graphical representation of the architecture. The model database stores the models as model schemas and model instances. The database server gets the request from the Web server to access the models. As shown in Figure 1, model suppliers and consumers may be using heterogeneous modeling tools and environments (platforms) like LINDO, Stella and GAMS. The client component allows model suppliers to convert platform dependent models into SMML models shareable on the web and then upload them, similarly model consumers can download SMML models and convert them to platform dependent models and then solve the models using these platforms.

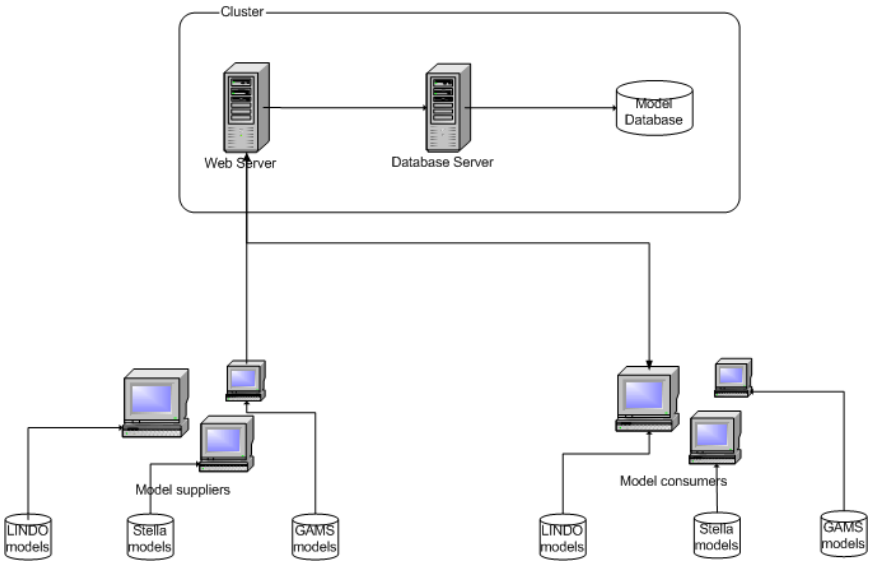

Figure 1. System architecture for the Web-based System

Model Representation using SMML

SM recognizes a set of Genus elements, their types, index, indexsets, interpretation, etc. as explained in (Geoffrion, 1987). Each SM model follows these guidelines for defining model elements. These guidelines are what make up a markup language. XML schemas define a markup language for validating model schemas and model instances against these guidelines. The proposed markup language used to facilitate defining models is referred to as Structured Modeling Markup Language (SMML). There are two separate XML schemas as part of SMML: ModelStructure.xsd is used to validate all model schemas and ModelInstance.xsd is used to validate model instances as shown in Figure 2. A model schema is the SM representation of a model, also called model structure and is an XML document. Similarly, a model instance is also an XML document that provides the data for a particular model schema. Corresponding to a model schema there can be one or more model instances/data.

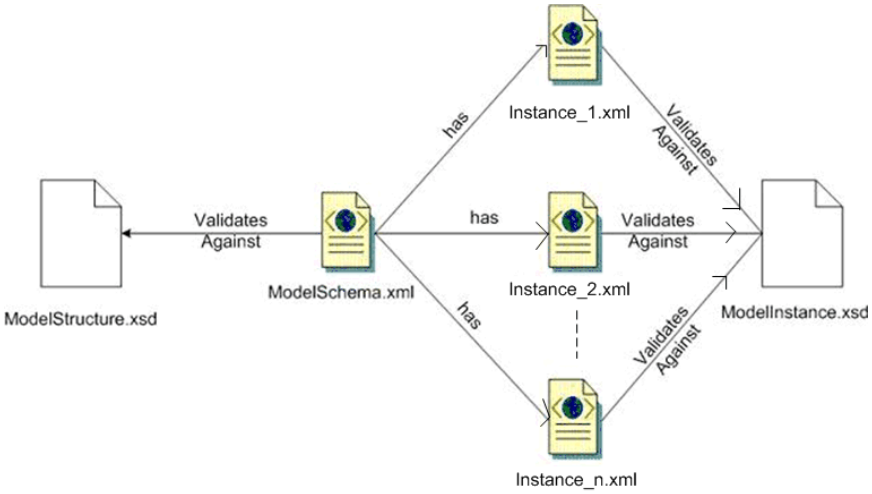

Figure 2. SMML model architecture

Figure 3 shows the model schema architecture in the SM approach. The root level represents the schema document as a whole. It can be further decomposed into any number of genus and module paragraphs. A module paragraph can consist of any number of genus and module paragraphs. Genus paragraphs are the lowest level in the hierarchy and define a single entity or element. A genus element can have attribute types. The solid ovals going out from the genus element denote the attributes. The attributes common to all Genus elements are the name, type and interpretation attributes. Type can take any one of the six values: pe, ce, a, va, f or t, corresponding to primitive entity, compound entity, attribute, variable attribute, function or test, respectively. A genus element can have exactly one of the attribute types. The dotted ovals below the type attribute are placeholders for type attribute values. In addition to the common attributes, each type introduces some special attributes for the genus element. So if the genus element is a "pe" type, it can only have one more additional attribute, "index", which is optional in case of non set-based models. Genus elements with a type value of "ce", "a" or "va", "f" and "t" have two more attributes in addition to the common ones, "calling sequence" and "indexset". "f" and "t" also introduce a "function description" which describes the equation used.

Writing a Model schema in SMML

When writing a model schema using SMML certain guidelines and rules have to be followed. These rules are enforced by the XML schema 'ModelStructure.xsd'. Figure 4 is a code snippet of ModelStructure.xsd. The last statement indicats that element "GENUS" is a complex type having elements and attributes. The name of the complex type is "GenusType". The definition of "GenusType" in the first statement indicates that it can have a child element called "TYPE" which is the type of the GENUS element and can take any one of the values "pe", "ce', "a", "va", "f" or "t" noted earlier. GENUS element can also have an attribute "name" which is unique for each GENUS element and identifies the name of the GENUS. There is a one-to-one correspondence with the SM model schema architecture described in Figure 3. Corresponding to a genus element, the XML schema defines a GENUS element. A module paragraph is defined using a MODULE element.

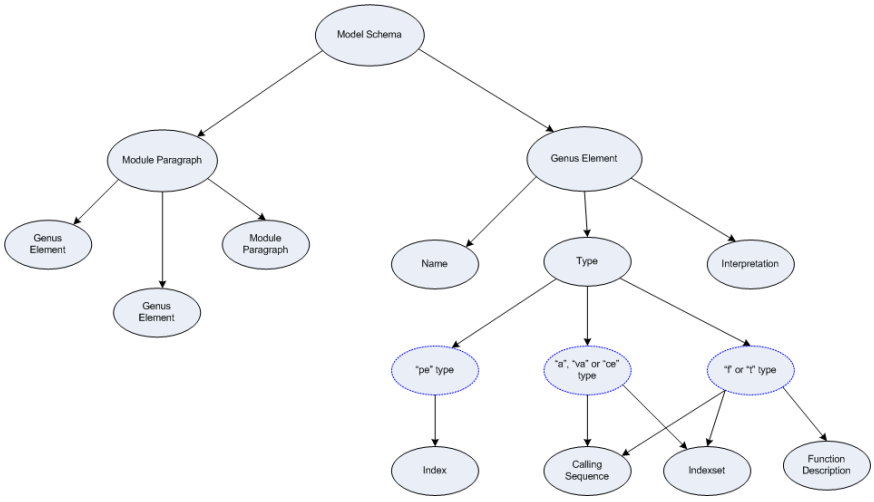

Figure 3. Model schema architecture in SM

```
<xsd:complexType name="GenusType">
        <xsd:sequence>
                <xsd:element name="TYPE">
                        <xsd:simpleType>
                                <xsd:restriction base="xsd:string">
                                        <xsd:pattern value="pe|ce|a|va|f|t"/>
                                </xsd:restriction>
                        </xsd:simpleType>
                </xsd:element>


        </xsd:sequence>
        <xsd:attribute name="name" type="xsd:string"/>
</xsd:complexType>
……
……
<xsd:element name="GENUS" type="GenusType"/>
```

Figure 4. Code snippet from ModelStructure.xsd

A model schema always starts with the tag <MODEL> with attributes such as name, level, and type. Level can have any one of the values 1, 2, 3 or 4 and specifies the complexity of the schema (Geoffrion, 1992a&b). The type specifies the category of modeling (also called modeling techniques) to which the model belongs, such as optimization, simulation, etc. The name attribute specifies a name for the model and can also be used to search for the model. Other attributes of element MODEL specify the location of XML schema file, i.e., ModelStructure.xsd. If the model intends to use MathML, it needs to declare that in the namespace along with the location of the XML schema used to validate MathML content. The MODEL element may optionally be followed by the element KEYWORDS which may be followed by a model DESCRIPTION.

Figure 5 shows a code snippet from the model schema representation of the transportation problem. The example shows the listing for two genera: PLANT and SUP. Since PLANT is a primitive entity it is assigned type as “pe”. SUP is an attribute type and assigned a value of “a” for the type. Primitive elements have an INDEX element associated with it. Other elements may have an INDEXSET associated that lists the set of primitive elements over which the genus is defined. Corresponding to each element inside an INDEXSET, a GENUSREF element is associated.

```
<GENUS name="PLANT">
        <TYPE>pe</TYPE>
        <INDEX>i</INDEX>
        <INTERPRETATION>There is a list of
                <KEY_PHRASE>PLANTS</KEY_PHRASE>
        </INTERPRETATION>
</GENUS>
<GENUS name="SUP">
        <TYPE>a</TYPE>
        <CALLING_SEQ>
                <GENUSREF refer="PLANT"/>
        </CALLING_SEQ>
        <INDEXSET>
                <GENUSREF refer="PLANT"/>
        </INDEXSET>
        <RANGE>R+</RANGE>
        <INTERPRETATION>Every PLANT has a
                <KEY_PHRASE>SUPPLY CAPACITY</KEY_PHRASE>measured in tons.
        </INTERPRETATION>
</GENUS>
```

Figure 5. Code snippet for the Transportation Problem in SMML

Figure 6 shows the a detailed view of the GENUS hierarchy which is adapted from Geoffrion (1987). Ovals denote an attribute and rectangles denote an element. Each GENUS element has a <TYPE> and </TYPE> tag that encloses any of the six values from the list "pe", "ce', "a', "va', "f" or "t". For a "pe" element, the TYPE element is immediately followed by an <INDEX> element, which encloses the index value, e.g., <INDEX>i</INDEX>, associates an index i with the genus. Indexes are useful in set-based arithmetic. Level 1 and Level 2 models do not involve sets and hence no INDEX elements.

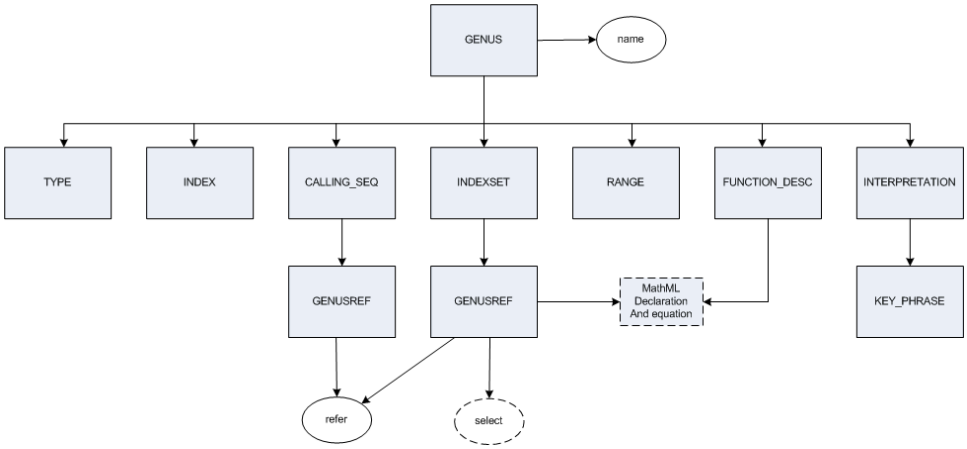

Figure 6. GENUS element hierarchy

The CALLING_SEQ element is used to include all the other elements that a genus refers to. INDEXSET is the set of elements that this genus is based on. With each attribute genus an element RANGE may be associated which defines the range of values for the attribute. Each "f" and "t" type genus contains a FUNCTION_DESC element. For representing equations, MathML is used which requires a namespace definition in the FUNCTION_DESC element. For example, a genus paragraph for element T:DEM (demand constraint for customers) in SM would be like:

```
T:DEM(FLOW.j,DEMj)  /t/ {CUST} ; SUMi(FLOWij) = DEMj
```

T:DEM is defined over the set of customers, i.e., CUST element. T:DEM is calculated for each CUST value. Below is a representation for the T:DEM genus in SMML and utilizing MathML:

Writing a Model Instance in SMML

A model instance is used to represent a particular set of data for a model. The XML schema file used to validate model instances is called 'ModelInstance.xsd'. A hierarchy of elements used in a model instance is given in Figure 7. A model instance always starts with the tag <ELEMENTAL_DETAIL>. It has such as 'name', 'refer', and 'namespace'. The 'name'

should be the name of the model schema to which it belongs. The 'refer' attribute may contain the physical filename or the location for the model schema. The way a model instance is represented in SM is through a table structure. The XML schema follows this convention so as to keep the data representation as generic in nature as possible. The namespace attribute should specify the address of ModelInstance.xsd.

The root element may be followed by any number of TABLE and PARAMETER elements in any order. The model instance document does not contain the function and test genus element values as they can be calculated from the data provided. Inclusion of decision variables is optional and if included, can provide a starting point for calculations and should provide a feasible solution to the problem.

Figure 8 shows a simplified code snippet for the model instance file for transportation problem. <TABLE> tag denotes the start of a table structure. Each elemental detail table translates to a TABLE element in a model instance. Each TABLE has a name to identify it uniquely. Each table has a record description (the <RECORD_DESC> tag) which lists the fields in the TABLE. Each field is denoted by a FIELD tag which can have three elements, NAME and TYPE are mandatory and define the name and data type of the field. The TYPE element can take any of the four values: integer, real, Boolean or string. In addition there is a third field, FOREIGN_KEY, which is optional and is used only when the field is related to a field in another table, i.e., is a foreign key. FOREIGN_KEY is an empty element, meaning it has no content, and has only one attribute, refer, which contains the name of an existing PRIMARY_KEY field in another table. The FIELD tags are followed by optional PRIMARY_KEY (see Figure 7) or UNIQUE_KEY. PRIMARY_KEY can contain only one FIELD element that contains a reference to one of the field names defined in the table.

```
<GENUS name="T:DEM">
        <TYPE>t</TYPE>
        <CALLING_SEQ>
                <GENUSREF refer="FLOW"/>
                <GENUSREF refer="DEM"/>
        </CALLING_SEQ>
        <INDEXSET>
                <GENUSREF refer="CUST"/>
        </INDEXSET>
        <FUNCTION_DESC xmlns:m="http://www.w3.org/1998/Math/MathML">
                <m:apply>
                        <m:eq/>
                        <m:apply>
                                <m:sum/>
                                <m:bvar>
                                        <m:apply>
                                                <m:selector/>
                                                <m:ci type="vector">PLANT</m:ci>
                                                <m:ci>i</m:ci>
                                        </m:apply>
                                </m:bvar>
                                <m:apply>
                                        <m:selector/>
                                        <m:ci type="matrix">FLOW</m:ci>
                                        <m:ci>i</m:ci>
                                        <m:ci>j</m:ci>
                                </m:apply>
                        </m:apply>
                        <m:apply>
                                <m:selector/>
                                <m:ci type="vector">DEM</m:ci>
                                <m:ci>j</m:ci>
                        </m:apply>
                </m:apply>
        </FUNCTION_DESC>
        <INTERPRETATION>Is the total FLOW arriving at a CUSTOMER exactly equal to its DEMAND?
        This is called the <KEY_PHRASE>DEMAND TEST </KEY_PHRASE>.
        </INTERPRETATION>
</GENUS>
```

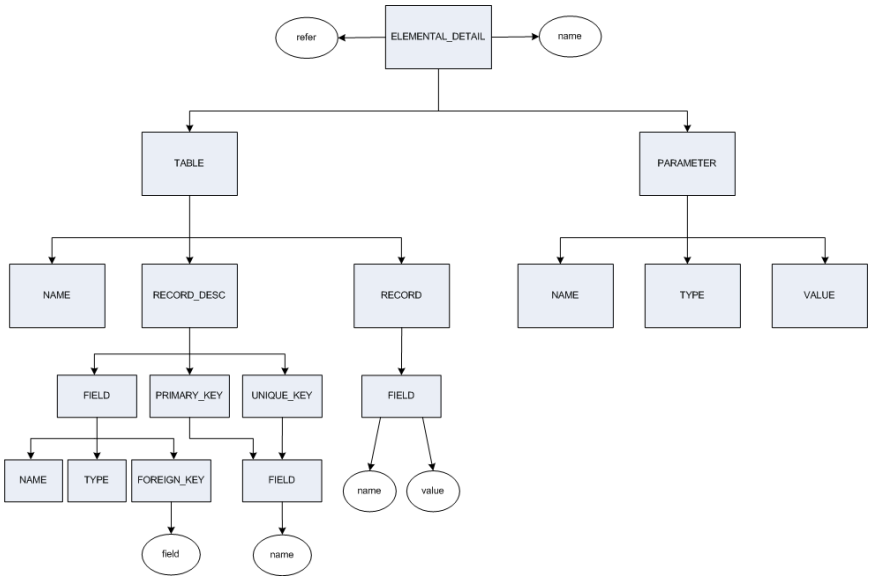

Figure 7. Model Instance element hierarchy

UNIQUE_KEY is used where the primary key is a composite key, composed of more than one field. Figure 9 is an example of a record description containing foreign field references and the use of FOREIGN_KEY tag. Corresponding to the number of records in the table, there are as many RECORD tags or elements. Each record element has the same number of FIELD elements as listed in the record description. Each FIELD element has a name and a value. Each RECORD is supposed to have a uniform structure within a table. This type of a structure closely resembles a table structure in a database. The table definition itself contains the meta data.

```
<TABLE>
        <NAME>PLANT</NAME>
        <RECORD_DESC>
                <FIELD>
                        <NAME>PLANT</NAME>
                        <TYPE>string</TYPE>
                </FIELD>
                <FIELD>
                        <NAME>SUP</NAME>
                        <TYPE>real</TYPE>
                </FIELD>
                <PRIMARY_KEY>
                        <FIELD name ="PLANT"/>
                </PRIMARY_KEY>
        </RECORD_DESC>
        <RECORD>
                <FIELD name="PLANT" value="DAL"/>
                <FIELD name="SUP" value="20000"/>
        </RECORD>
        <RECORD>
                <FIELD name="PLANT" value="CHI"/>
                <FIELD name="SUP" value="42000"/>
        </RECORD>
</TABLE>
```

Figure 8. Code snippet for a model instance for the transportation problem in SMML

```
<RECORD_DESC>
        <FIELD>
                <NAME>PLANT</NAME>
                <TYPE>string</TYPE>
                <FOREIGN_KEY field="PLANT"/>
        </FIELD>
        <FIELD>
                <NAME>CUST</NAME>
                <TYPE>string</TYPE>
                <FOREIGN_KEY field="CUST"/>
        </FIELD>
        <FIELD>
                <NAME>COST</NAME>
                <TYPE>real</TYPE>
        </FIELD>
        <UNIQUE_KEY>
                <FIELD name="PLANT"/>
                <FIELD name="CUST"/>
        </UNIQUE_KEY>
</RECORD_DESC>
```

Figure 9. Code snippet for a model instance demonstrating the use of Foreign keys

What ties the model schema and a model instance together are the FIELD names used. The FIELD names used should be same as already defined in the model or there is no way to relate it to the model schema. The fields in the table should have the same name as the ones used to define the genus in the model schema. So, for the transportation problem, the PLANT genus defines the list of plants. In the model instance too, the field name has a value PLANT to be able to assign values to the GENUS plant. There might be situations where an element that does not belong to a set, or is not part of a table as it is a simple entity with just a single value, in that case, it can be defined as a PARAMETER entity. Below is an example:

```
<PARAMETER>
        <NAME>LABOR_SUPPLY</NAME>
        <TYPE>integer</TYPE>
        <VALUE>160</VALUE>
</PARAMETER>
```

This statement translates as LABOR_SUPPLY = 160. Each PARAMETER element has a NAME, TYPE and VALUE element associated with it. The name is the GENUS name for which TYPE and a VALUE is provided.

CASE STUDY

There are two types of users, model suppliers and model consumers. Model suppliers upload the models to the model repository on the Web (Figure 10) and model consumers search and download these models to their local machines (Figure 11). These users may be using various modeling tools like GAMS, LINDO, Stella, etc. Since the models are written in SMML, the users need a system that allows for easy conversion of SMML models to platform-dependent models and vice versa. To demonstrate the ease of converting SMML models to a platform dependent model, the research builds a stand-alone windows application for converting LINGO models to and from SMML. The application demonstrates that an XML model schema together with a model instance for that schema can be integrated to generate a LINGO model. The model consumers using LINGO solver can download the SMML models and easily convert them to a LINGO model. The application also allows the user to solve the model by interfacing with the LINGO Dynamic Link Library (DLL).

Model suppliers, on the other hand, can convert their LINGO models to SMML models with the click of a mouse and upload them to the Web repository for others to benefit from the model written already. For this purpose, the application takes as input a LINGO model and generates a valid model schema and/or a model instance. The application has been successfully used to convert SMML models to LINGO and vice versa. The application has also been successfully used to solve a LINGO model once generated from SMML models. The case study uses a custom built equation parser in VB.NET and parses MathML equation written using content markup only. Figures 12 and 13 show a screen shot of the client-side application environment.

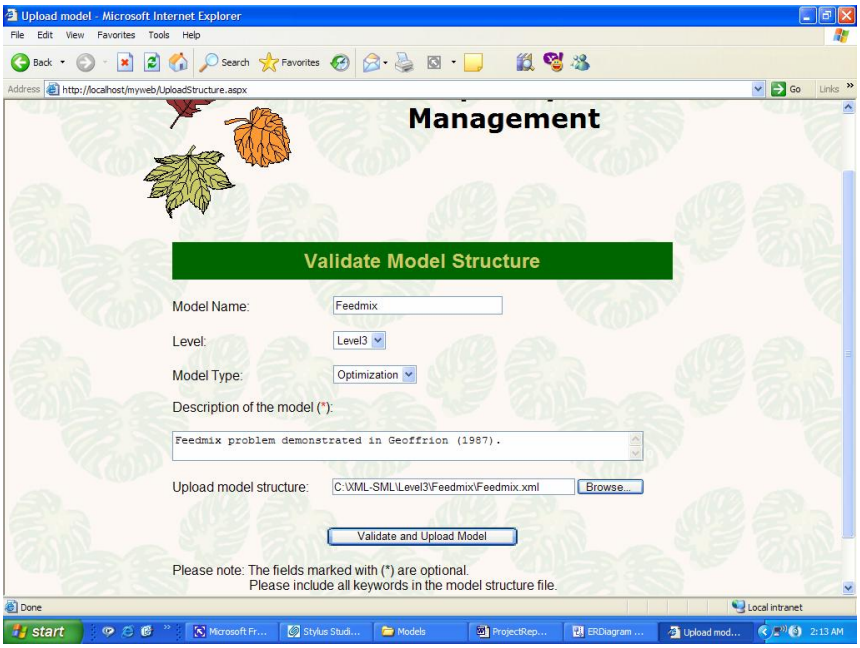

Figure 10. Model Schema Uploading through the Web server

RESULTS AND CONCLUSION

The research recognizes models as a corporate and national resource and lays the foundation for model sharing, integration and reuse in a distributed environment. The research proposes an XML-based architecture that leverages the synergy between SM and XML to develop the SMML for model representation. While XML provides the means for sharing and exchanging management models, SM provides the formalism and foundation for such markup language.

In essence, this paper proposes an XML-based architecture for sharing heterogeneous models on the Web using a standardized model representation. At the core of the architecture is SMML, which is based on structured modeling as a conceptual modeling framework. Such a language will prove helpful in bringing together modelers using disparate modeling tools and environment. The paper demonstrates the proposed architecture using a case study including server side components for maintaining a shared web-based model repository and a client-side components for converting models to and from SMML.

Recommendations for future research

With the inherent compatibility among SM, object-oriented (OO) modeling, and XML, future research needs to further capitalize on such synergy to develop modeling environments that can easily translate from SMML model representations into OO constructs executable in distributed environment. Moreover, since the research is conducted on a limited set of models mostly taken from Geoffrion (1987), Geoffrion (1992) and the LINGO User's Guide (2003), future work should expand upon these models to include a comprehensive list of models from several modeling paradigms such as simulation, simultaneous differential equations, stochastic models, etc.

Future work is also needed to standardize such a modeling language through practical reviews and modifications. Last but not least, research should also be done on the client component exploring different approaches to simplify converting SMML to and from various modeling paradigms, e.g., through the use of XML technologies such as Extensible Stylesheet Language (XSL) and XSL Transformation (XSLT).

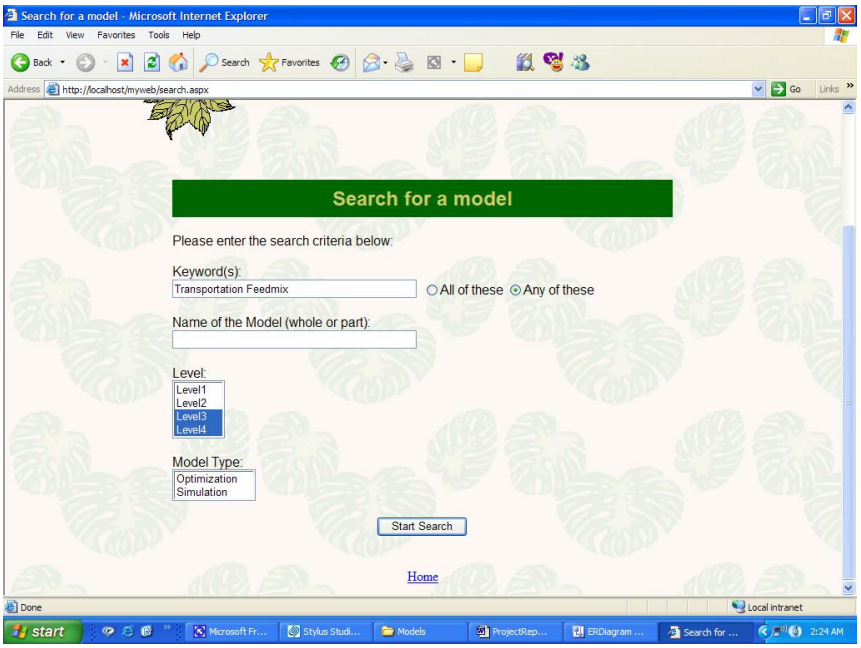

Figure 11. Specifying a search criteria for models

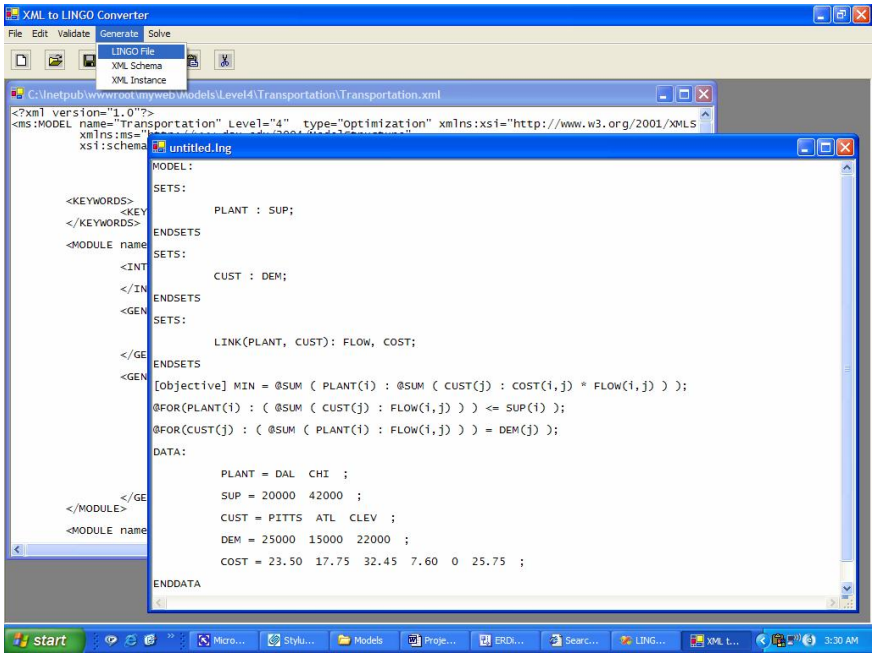

Figure 12. LINGO Solver Application – SMML Model to LINGO Model Conversion

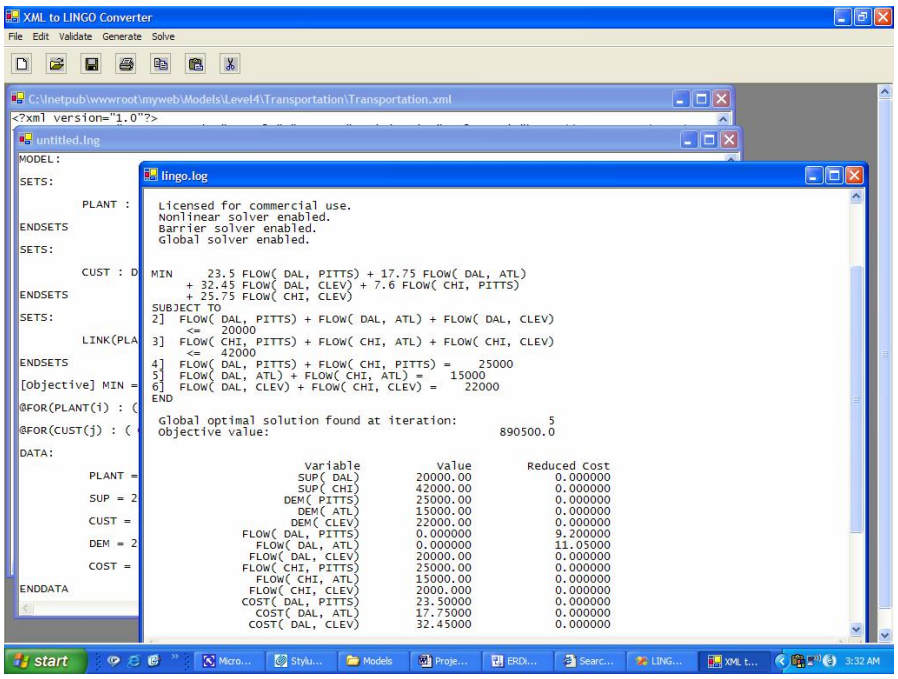

Figure 13. LINGO Solver Application – LINGO Solution generation using a LINGO model

ACKNOWLEDGMENTS

This material is based upon work supported by the National Science Foundation/EPSCoR Grant #EPS-0091948 and by the State of South Dakota. The usual disclaimer applies.

REFERENCES

1. Bhargava , H. K., & Kimbrough, S. O. (1993). Embedded Languages for Model Management. *Decision Support Systems, 10*(3), 277-299.
2. Binbasioglu, M., & Jarke, M. (1986). Domain specific DSS tools for knowledge-based model building. *Decision Support Systems, 2*(3), 213–223.
3. Blanning , R. W. (1993). Model Management Systems: An Overview. *Decision Support Systems, 9*, 9-18.
4. Bradley, N. (2002). *The XML companion* (3rd. Ed.). Boston, MA: Addison-Wesley.
5. Brooke, A., Kendrick , D., & Meeraus, E. (1988). *GAMS: A Users Guide*. Redwood City, CA: The Scientific Press.
6. Canonico, R., Emma, D., & Ventre, G. (2003). *An XML based network simulation description language.* Proceedings of the 7th International Symposium on Distributed Simulation and Real Time Applications (DS-RT 2003), Delft, Netherlands.
7. Chang, A.-M., Holsapple, C. W., & Whinston, A. B. (1993). Model management issues and directions. *Decision support system, 9*, 19-37.
8. Evans, K. A., Kamanna, K., & Mueller, J. (2002). *XML and ASP.NET*. Indianapolis, IN: New Riders Publishing.
9. Finney, A., & Hucka, M. (2003). Systems biology markup language: Level 2 and beyond. *Biochemical Society Transactions, 31*(6), 1472-1473.
10. Fourer, R., Gay, D. M., & Kernighan, B. W. (1993). APML: A modeling language for mathematical programming. In. Redwood City, CA: The Scientific Press.
11. Geoffrion, A. M. (1987). An introduction to structured modeling. *Management Science, 33*(5), 547-588.
12. Geoffrion, A. M. (1989). The formal aspects of structured modeling. *Operational Research, 37*(1), 30-51.

13. Geoffrion, A. M. (1992a). The SML language for structured modeling: Levels 1 and 2. *Operations Research, 40*(1), 38-57.

14. Geoffrion, A. M. (1992b). The SML language for structured modeling: Levels 3 and 4. *Operations Research, 40*(1), 58-75.

15. Data Mining Group. (2004). *Predictive Model Markup Language (PMML)*. Retrieved 1/6/2004, from http://www.dmg.org/

16. Hucka, M., Finney, A., Sauro, H., Bolouri, H., Doyle, J. C., & Kitano, H. (2003). The systems biology markup language (SBML): a medium for representation and exchange of biochemical network models. *Bioinformatics, 19*(4), 524-531.

17. Jones , C. V. (1990). An introduction to graph based modeling system: Part 1. Overview. *ORSA Journal of Computing, 2*(2), 180-206.

18. Katz, S., Risman, L. J., & Rodeh, M. (1980). A system for constructing linear programming models. *IBM Systems Journal, 19*(4), 505-520.

19. Kim, H. (2001). An XML based modeling language for open interchange of decision models. *Decision Support Systems, 31*, 429-445.

20. Krishnan, R., & Chari, K. (2000). *Model Management: Survey future research directions and a bibilography*. Retrieved 3/20/2003, from http://www.coba.usf.edu/departments/isds/faculty/chari/model/doc.html

21. Lenard, M. L. (1993). An object oriented approach to model management. *Decision Support Systems, 9*, 67-73.

22. LINDO Systems Inc. (2003). *LINGO User's guide*. Chicago, IL: LINDO Systems Inc.,.

23. Lu, R. F., Qiao, G., & Mclean, C. (2003). *NIST XML simulation interface specification at Boeing: A case study*. Proceedings of the 2003 Winter Simulation Conference, Piscataway, NJ.

24. Muhanna , W. A. (1993). An object oriented framework for model management and DSS development. *Decision Support Systems, 9*(1), 217-229.

25. Qiao, G., Raddick, F., & McLean, C. (2003). *Data Driven Design And Simulation System based on XML*. Proceedings of the 2003 Winter Simulation Conference, Piscataway, NJ.

26. Sprague, R. H., & Watson, H. J. (1975). *Model Management in MIS.* Proceedings of the Seventeenth National AIDS, Cincinnati, OH.

27. W3C Math working group. (2003). *Mathematical Markup Language (MathML)*. Retrieved 6/15/2004, from http://www.w3.org/tr/2003/REC-MathML2-20031021

28. Wang, Y., & Lu, Y. (2002). *An XML-based DEVS Modeling Tool to Enhance Simulation Interoperability*. Proceedings of the 14th European Simulation Symposium.

29. Will , H. J. (1975). Model Management Systems. In E. Grochla & N. Szyperski (Eds.), *Information systems and organization structure* (pp. 468-482). Berlin: Walter de Gruyter.

30. Zeigler, B. P. (1990). *Object-oriented simulation with hierarchical modular models: Intelligent agents and endomorphic systems*. Boston, MA: Academic Press.