4-2007

An XML-based schema definition for model sharing and reuse in a distributed environment

Omar F. El-Gayar
Dakota State University

Kanchana Tandekar
Dakota State University

Recommended Citation

El-Gayar, O., & Tandekar, K. (2007). An XML-based schema definition for model sharing and reuse in a distributed environment. Decision Support Systems, 43(3), 791-808.

Available online at www.sciencedirect.com
ScienceDirect
Decision Support Systems 43 (2007) 791–808
Decision Support Systems
www.elsevier.com/locate/dss

An XML-based schema definition for model sharing and reuse in a distributed environment ☆

Omar El-Gayar *, Kanchana Tandekar

Dakota State University, United States

Received 21 September 2005; received in revised form 31 July 2006; accepted 17 December 2006
Available online 23 December 2006

Abstract

This research leverages the inherent synergy between structured modeling and the eXtensible Markup Language (XML) to facilitate model sharing and reuse in a distributed environment. This is accomplished by providing an XML-based schema definition and two alternative supporting architectures. The XML schema defines 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. Overall, SMML and supporting architectures allow different types of models, developed in a variety of modeling platforms to be represented in a standardized format and shared in a distributed environment. The paper demonstrates the proposed SMML through two case studies.

Keywords: Model management; Model sharing; Structured modeling; eXtensible Markup Language (XML); Distributed computing environments

1. 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. "Modeling in the large" as denoted by Muhanna and Pick [51] explicitly recognizes models as a resource and modeling as an ongoing process that should be supported. The focus is on the management of large shared model bases. Such a view of modeling and the emphasis on reuse is growing within the decision support community [41]. Model reuse can take several forms including the ability to use the same model with a different data set or with a different solver, and to use the same model in a different modeling environment. The notion of reuse can also be extended to include composing and integrating models from existing models. Supporting the modeling process in general, and "modeling in the large" in particular is the ability to represent models at a higher level of abstraction, i.e., meta-modeling.

However, in practice, models use a myriad of languages and task specific representations that include textual descriptions of problem statements, modeling languages, and graphical notation. While some model

☆ An earlier version of this paper was presented at the AMCIS 2005 conference, Omaha, NE. This paper extends and enhances those ideas and describes them more fully in a coherent framework.

* Corresponding author.
E-mail address: omar.el-gayar@dsu.edu (O. El-Gayar).

0167-9236/$ - see front matter © 2007 Elsevier B.V. All rights reserved.
doi:10.1016/j.dss.2006.12.010

representations offer distinct advantages such as model-data independence, others have data intertwined with the model structure. Moreover, several representations (and modeling environments) can be used within the same organization for addressing the same type of model. To share and reuse models in such environments, individual translators will need to be developed for each pair of model representation schemes. This solution is not scalable, particularly in the context of distributed inter-organizational setting. Moreover, such representations are not directly amenable to architectures supporting distributed environments. Last, but not least, such representation schemes are often paradigm dependent. In effect, without a unified scheme for representing the structure and semantics of models that preserves model-data, model-solver, and model-paradigm independence, efforts to support "modeling in the large" and to leverage existing investments in models through sharing and reuse are seriously curtailed.

Two important developments offer promising results, namely, structured modeling (SM) and eXtensible Markup Language (XML). In the quest for expressive model representation, structured modeling received a great deal of attention in the literature. SM as defined by Geoffrion [26] is a "formal mathematical and computer-based environment for conceiving, representing and manipulating a wide variety of models". SM has many of the features desired in model management systems, which makes 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.

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 [6]. 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 [26] demonstrated different ways of rendering a structured model as a web document, not much research has been done to represent structured models using XML.

This research leverages the inherent synergy between SM and XML to facilitate model sharing and reuse in a distributed environment. This is accomplished through the development of an XML-based schema definition and supporting architectures. The XML schema defines 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 language allows for:

- Representing different types of models that are developed using a variety of modeling platforms in a standardized format.
- Sharing and publishing models among model users regardless of their modeling environments.
- Reusing models (developed for different data sets, for different solvers, and in different modeling environments) 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 two sections provide a brief review of model representation and XML. Next, we highlight the advantages of using structured modeling and XML for model representation followed by a description of the schema definition for SMML. We then present two alternative supporting architectures followed by case studies demonstrating the applicability of the proposed representation. The final section concludes the paper.

2. Model representation

Model management (MM) emerged in the mid-seventies in the context of managing models in decision support systems (DSS) [54,57]. While a comprehensive review of the model management (MM) literature can be found elsewhere [3,12,41], it is worth noting that much of the motivation behind MM focused around finding ways for developing, storing, manipulating, controlling, and effectively utilizing models in an organization [50]. Inherent in such functionality is the ability to represent models at a higher level of abstraction, i.e., meta-modeling.

2.1. Requirements for model representation

Requirements for model representation evolved over the years to accommodate the increasing demands by analysts and users. Initial interest focused on abstracting away from low-level input formats and moved on to emphasize representing models at a higher level of abstraction to facilitate model management functions.

Design requirements supporting model management functions include:

- Paradigm independence: Ability to represent models from different paradigms.
- Model-data independence: Ability to use the model with different data sets.
- Model-solver independence: Ability to use the model with different solvers.
- Meta-level representation and reasoning: Ability to represent information about models.
- Single representation format: Amenable to multiple purposes such as computer execution, mathematical use, and managerial communication.

The advent of the Internet and related technologies put further demands with respect to model sharing. Additional requirements supporting model sharing in a distributed environment include:

- Portability: Compatible with various development environments and systems.
- Vendor independence: Is not locked by the support of a particular vendor.
- Extensibility: Amenable to extensions and continuous improvement efforts.
- Leverage of the Internet: Compatibility with Internet technologies such as XML.

2.2. Model representation supporting model management functions

A number of model representation approaches and languages are proposed in the literature, most notably are algebraic modeling languages, logic-based systems, relational-based approaches, graph-based languages, and structured modeling (which is also the basis for other approaches). Algebraic modeling languages such as GAMS [8], AMPL [22], and LINGO [39] attempts to simplify modeling by abstracting away from detailed model representations using ad hoc programs or matrix generator programs. In effect, these languages allow for a symbolic, general, concise, and understandable means to represent models [21] and tend to be specific to a particular modeling paradigm [41]. Moreover, these languages vary in their support to model representation requirements. For example, while GAMS supports model-solver independence, AMPL supports model-data and model-solver independence and both languages do not support meta-level representation.

In that regard, Bhargava and Kimbrough [1] propose a logic-based approach to model representation that augments existing modeling languages in representing formally and computationally information about expressions that is not amenable for representation. The objective is to leverage the strengths in existing modeling languages while being able to capture meta-model information that can then support MM functions. Nevertheless, in a distributed environment, a user is still confronted with handling models in a variety of representational formats representing the embedded language.

Relational-based approaches attempts to view and thus represent models as data. Proposals include [2,14,45]. A primary motivation of such approaches is to capture the developments in relational database technology for model representation and management. However, such approaches are particularly limited when it comes to handling rich model semantics and are often paradigm dependent.

With the variety of models that can be represented by graphs [9,37], graph-based modeling systems (GBMS) [37,38] attempts to facilitate the creation of graphs representing models that can be represented as attributed graphs. Inherent in these systems is the use of graph grammars to represent the structural constraints imposed on various types of graphs. Examples of such graphs include vehicle routing, neural networks, and structured modeling Genus graphs. While graph-based modeling has proven useful for model formulation and communication, graph-grammars can be difficult to manipulate and the applicability of graph-grammars to a wide variety of graph-based models need further exploration [37].

Geoffrion [26] proposed structured modeling (SM) as a means of model representation. Geoffrion [29] defines SM as "a way of thinking about analytic models and the systems which support them". SM views models and model-based systems as having computationally active definitional dependency diagrams as their central focus, with manipulation by solvers and other tools to achieve desired purposes. SM provides a conceptual framework for conceiving, representing, and manipulating a wide variety of models. It should be noted that model representation in SM does not specify the objective function and whether its value should be maximized or minimized. These are solver issues and can be defined at the time of solving the model. Thus a user may decide to use some or all of the constraints from the model and may decide whether to minimize or maximize a function. If there is more than one objective function, the user can choose the desired objective or may even wish to solve for all of them using different techniques like multi-objective programming, weighted goal programming, compromise programming, etc. The model representations are independent of the model solution and the way the model is

solved. Moreover, the model schema does not contain any actual data and this provides data independence. The same model may be invoked using different set of data. The data, in SM terminology known as an instance is maintained as a separate file. Accordingly, 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 hierarchical, acyclic, and attributed graph. 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 [26,27,30,31].

2.3. Limitations of existing approaches

Significant amount of model management research occurred in the early eighties and continued to the mid-nineties. Most of the research focused on addressing requirements of model management systems such as model-data and model-solver independence, paradigm independence, meta-model representation, and facilitating model management functions. With the exception of Muhanna [51] and few others, not much attention was given to managing large shared model bases. With the advent of the Internet and supporting communication infrastructure, it became feasible to easily share models within and across organizations resulting on additional requirements for portability, vendor independence, and compatibility with the new infrastructure. Accordingly, a major limitation of the aforementioned approaches is their limited support to the requirements for model sharing in a distributed environment.

3. XML and model representation

XML is a meta-language originally developed by the World Wide Web Consortium (W3C) as a simplified subset of SGML. In contrast to Hyper Text Markup Language (HTML) where the emphasis is on visual display and user interface definition, XML-based languages provide the means to represent the content, semantic, and schemata of data. XML allows for the definition of a set of domain specific tags which can then be used to markup data in various applications domains. By facilitating the exchange of data using a non-proprietary data format, XML is particularly valuable in supporting interoperability in distributed heterogeneous environments such as the Internet. 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.

3.1. XML-based languages for model representation

XML success has also extended to representing models in various problem domains and for specific modeling paradigms. Hucka et al. [35] and Finney and Hucka [20] describe a Systems Biology Markup Language (SBML) for representing and exchanging biochemical network models between simulation/analysis tools. The language is supported by a package for manipulating the SBML-based models [53]. 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 [15].

With respect to simulation, Canonico et al. [10] present 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 [55] develop an XML application to represent discrete event simulation models based on the DEVS (Discrete Event System Specification) approach [58], while Lu et al. [49] and Qiao et al. [52] discuss cases utilizing an XML-based simulation interface specification being developed by the National Institute of Standards and Technology (NIST).

Several XML-based languages are available to represent graph models. Examples include GraphML [7], GXL [33], and NaGML [5]. In general, these languages share a common purpose of providing an easy-to-use format for representing graphs that facilitates the sharing and exchange of such models. However, the languages differ with respect to problem domain focus, implementation specifics, and features. For example, GXL evolved from the software re-engineering community with an emphasis on representing and exchanging software engineering artifacts as graphs while GraphML was initiated during the 2000 Graph Drawing Symposium in Williamsburg, Virginia as a mean to represent graphs. To accommodate problem domain specifics, NaGML allows users to specify the name, data type, and restrictions for each node and arc property thus defining a problem domain schema within the document.

In the context of optimization models, a number of XML-based languages have been proposed [4]. Fourer et al. [24] propose Optimization Service Instance Language (OSiL) an XML-based computer language for representing instances of large-scale optimization problems including linear programs, mixed-integer programs, quadratic programs, and very general nonlinear programs. Fourer et al. [23] propose LPFML as

an XML schema for representing linear programming (LP) models and a set of C++ classes for translating models to and from LPFML. The language facilitates interoperability among modeling languages and solvers by reducing the number of drivers needed. While LPFML allows for a standard representation of LP models, the focus is on representing model (problem) instances as opposed to model structure (schema). In effect, model data and solution are stored with the model definition. Moreover, and contrary to mathematical programming languages such as GAMS and LINGO, LPFML does not make use of sets to represent models at a higher level of abstraction. Other related work include OptML [42] and SNOML [48]. Both languages focus on representing instances of linear and mixed integer programming models at the matrix level. In the context of a comprehensive framework for optimization, Ezechukwu and Maros [19] propose an architecture supporting distributed optimization over the Internet. At the core of the architecture are two XML-based languages, namely, Algebraic Modeling Language (AML) for representing models, and Optimization Reporting Markup Language (ORML) for representing model solutions, and a collection of Java programs referred to as Optimization Service Connectivity Protocol (OSCP) that are responsible for converting AML models to a target system for execution and converting model results to ORML.

Nevertheless, only Kim [40] 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 is 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.

3.2. Limitations of existing XML-based languages for model representation

In contrary to earlier model management and representation efforts, a primary motivation for XML-based languages for representing models is facilitating model sharing in distributed environments. Towards this end, and with the exception of Kim [40], these approaches tend to violate model management and representation requirements such as paradigm independence and model-data independence. This can be attributed to the problem-focus of the proposed approaches and the absence of a conceptual foundation for representing models at a higher level of abstraction.

4. Structured modeling and XML

Among the aforementioned approaches and associated languages, structured modeling (SM) is particularly attractive. Specifically, SM has many features that are highly desirable from a MM perspective [26,41], most notably are the independence of model representation and model solution, the sufficient generality to encompass a wide variety of modeling paradigms, and the representational independence of general model structure from the detailed data needed to describe specific model instances [26]. Having SM as a conceptual foundation for SMML facilitates model management tasks such model formulation [46] and model composition [32] where they used variants of SM as an underlying representation scheme. Moreover, SM offers distinct advantages 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 [41]. It is also the basis for a number of proposals for model representation including object oriented approaches [25,36,44,50], relational-based approaches such as [16,43], and graphic-based approaches such as GBMS/SM [13]. Finally, SM is compatible with XML. All this makes SM a useful tool and formal foundation for any modeling environment.

On the other hand, XML offer distinct advantages with respect to sharing information in general and models in particular in a heterogeneous distributed environment. Specifically, XML representations are vendor independent thereby supporting interoperability, extensible thereby amenable to continuous enhancements, and compatible with the latest Internet technologies and architectures such as web services, Simple Object Access Protocol (SOAP), and service oriented architectures (SOA). Moreover, the proliferation of XML made available numerous development tools and resources further simplifying the process of producing and consuming XML-based models.

In light of the aforementioned discussion, our proposal seeks to leverage structured modeling, XML,

and the synergy that exist as means for representing models to facilitate model sharing in distributed environments. In effect, we can summarize the contributions of the work as follows:

- It leverages SM to provide a conceptual framework for representing models and thereby meeting design requirements for model management such as the ability to reuse existing models with different data sets and different solvers.
- It is an XML application and is thereby compatible with the Internet, facilitates model sharing in a distributed environment by providing a standard and portable format for model representation, and leverages existing and latest developments in Internet and web technologies and architectures.
- It extends [40] by providing a more general and complete view for integrating SM and XML. Specifically, this research uses XML schemas for model representation thereby leveraging the richness of data types, the extensive support for name spaces and the other advantages of schemas over DTDs [6,18]. For example, by using a schema we can define and restrict data values in models [23]. There is a separate schema to validate a model instance and enforce rules that checks for consistency of data. The use of schemas ensures the stability of the standard. Last but not least, SMML utilizes MathML [56] as a standardized means for representing equations.

5. Model representation using SMML

SM recognizes a set of genus elements, their types, index, indexsets, interpretation, etc. as explained in [26]. Each SM model follows these guidelines for defining model elements. XML schemas define a markup language (a vocabulary) for validating model schemas and model instances against these guidelines. The proposed markup language that is used to facilitate the defining of 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. 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. For any model schema there can be one or more model instances/data.

In representing XML schemas (vocabularies), several authors advocate the use of the Unified Modeling Language (UML) [11,34]. In effect, using UML to represent XML schemas:

- Clearly depicts the semantics and structure of the underlying vocabulary, information that is often obscured in the lengthy text-based documents.
- Provides a visual notation that is implementation neutral.
- Facilitates reading and understanding by non-computer scientists.
- Relies on a de facto standard with readily available documentation in the form of books, web resources and articles.

Fig. 1 depicts a UML class diagram representing the SMML vocabulary. A model schema always starts with the <MODEL> element (denoted using a UML class) with attributes such as name, level, description, keywords, and type. Level can have any one of the values 1, 2, 3 or 4 and specifies the complexity of the schema [30,31]. 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 the XML schema file, i.e., ModelStructure.xsd. The MODEL element may optionally be followed by the element KEYWORDS which may be followed by a model DESCRIPTION.

A model can be further decomposed into any number of genus and module paragraphs as noted by the composition relationships between the model and module and genus classes. In turn, a module element (paragraph in SM terminology) can consist of any number of genus and module elements. Genus elements are the lowest level in the hierarchy. A genus element can have attribute types (represented as XML elements). The attributes common to all genus elements are the name, type, index, 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. In addition to the common attributes, each type introduces some special attributes for the genus element. For example, the genus element of types “ce”, “a”, “va”, “f” or “t”, has two more attributes, namely, “calling sequence”, and “indexset”. Furthermore, “f” and “t” also introduce a “rule” which describes the equation used (function description) and is

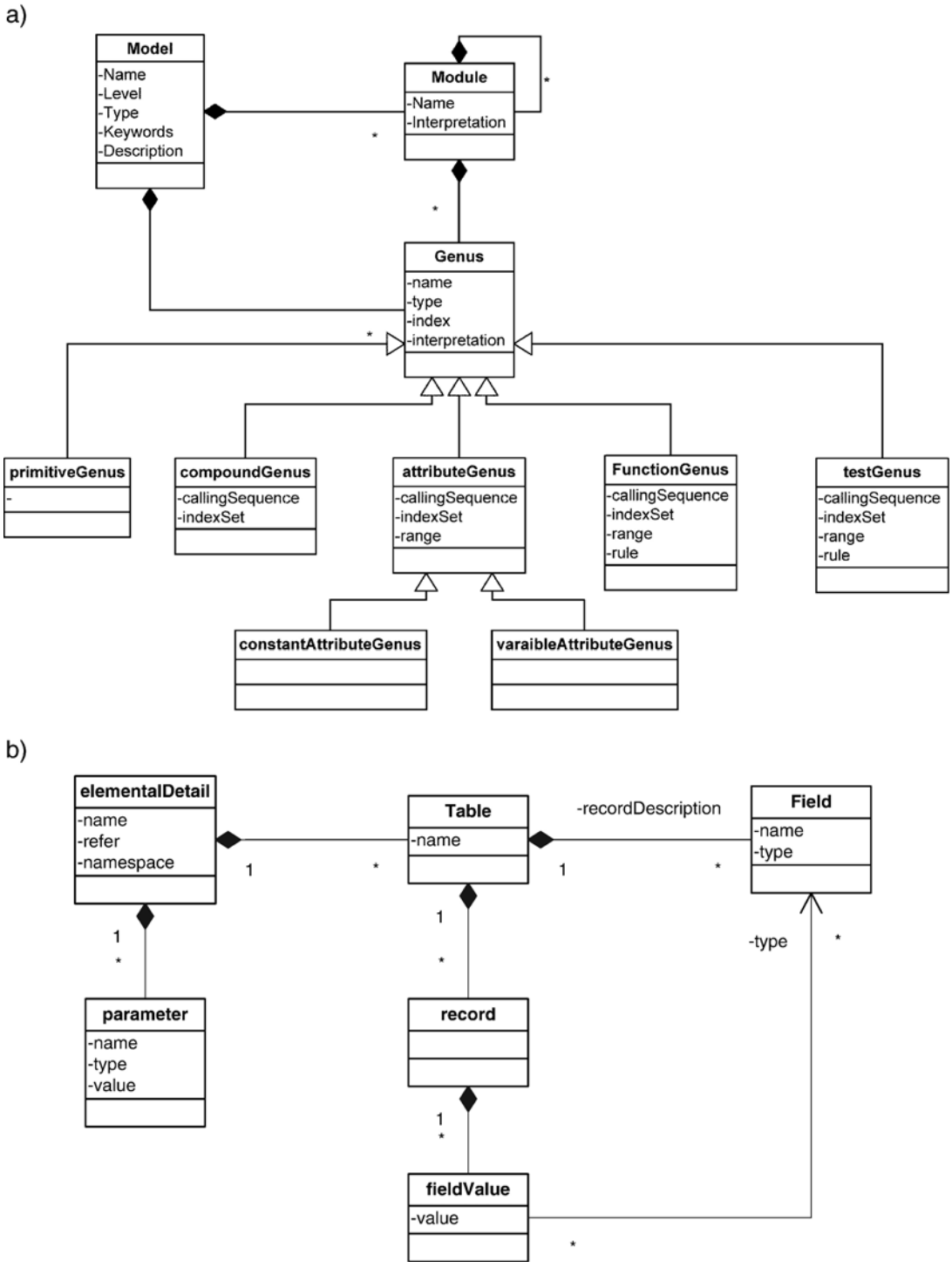

Fig. 1. Model schema and instance architecture in SM. a) Model schema architecture. b) Model instance architecture.

represented using MathML. In effect, the genus type specializes a typical genus by specifying which attributes are relevant. This is denoted by the inheritance relationship in the UML model.

5.1. 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'. The following is a code snippet of ModelStructure.xsd depicting genus definition:

```
<xsd: complexTypename="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"/>
```

The last statement indicates that element "GENUS" is a complex type having elements and attributes. The name of the complex type is "GenusType" which corresponds to the "Genus" class in the UML model. 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 Fig. 1. Corresponding to a genus element, the XML schema defines a GENUS element. A module paragraph is defined using a MODULE element. The following SMML is a code snippet from the model schema representation of the transportation problem:

```
<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>
```

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>, and associates an index i with the genus. Indices are useful in set-based arithmetic.

The example shows the listing for two genera: PLANT and SUP. PLANT is a primitive entity and is assigned type as "pe" and has an INDEX element associated with it indicating that there can be a set of plants. SUP is an attribute type and assigned a value of "a" for the type and has an INDEXSET associated with it identifying the set of primitive elements over which the genus is defined. In this example, each SUP is associated with a PLANT through the INDEXSET and GENUSREF elements.

The CALLING_SEQ element is used to include all the other elements to which a genus refers. 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:

```
T: DEM(FLOW. j, DEMj) /t/; 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:

```
<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>
```

5.2. 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'. Using UML, we can represent the SMML vocabulary as shown in Fig. 1b. A model instance always starts with the tag <ELEMENTAL_DETAIL> denoted by the "elementalDetail" class. It has attributes 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 to allow validating parsers to identify the schema to which to validate against.

The root element may be followed by any number of TABLE and PARAMETER elements in any order as noted by the composition relationship between the "elementalDetail" class and the "parameter" and the "table" class. Each table is comprised of a set of field type definitions defining a record description and a set of records. Each record is comprised of a set of field values corresponding to the field definition of the record description. The following is a code snippet for the model instance file for transportation problem:

```
<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>
```

As shown in this example, <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 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.

UNIQUE_KEY is used where the primary key is a composite key, composed of more than one field as show in the following example of a record description

containing foreign field references and the use of FOREIGN_KEY tag:

```
<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>
```

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 structure closely resembles a table structure in a database. The table definition itself contains the metadata. Notice that 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.

What ties the model schema and a model instance together are the FIELD names used. The FIELD names used should be the 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 must have a value PLANT to be able to assign values to the GENUS plant. There might be situations where there is 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.

6. Supporting architectures

SMML is a model representation language that is platform and vendor independent, provide for model-instance independence, and provide a standardized format for exchanging models. The following subsections provide two alternative architectures supporting model sharing and exchange in a distributed environment where SMML represent the primary mechanism for representing models.

6.1. A centralized architecture

There are three tiers to the architecture of the system: a server, middle and a client tier as depicted in Fig. 2a. The server tier is comprised of a model management system (MMS) responsible for storing, retrieving, and querying the database of model schema and associated model instances (data). All model schema and instances are stored as XML text files using the Structured Modeling Markup Language (SMML). To facilitate model management functions [41], the MMS maintains information about models in a relational database. Moreover, by using XML technologies such as XPath, the MMS can directly query the repository of SMML models.

The middle tier is comprised of a Parser/translator manager that manages (adds, updates, retrieves, deletes, and searches) a library of parsers/translators. The purpose of the parsers/translators is to translate platform dependent models into SMML. For example, GAMS and Excel translators can translate GAMS and spreadsheet models, respectively into SMML. By the same token, parsers are needed to parse SMML model schemas and instances and generate platform specific models. Such parsers can generate alternate model representation such as GAMS (for which a solver needs to be invoked) or can directly generate model files for a specific solver.

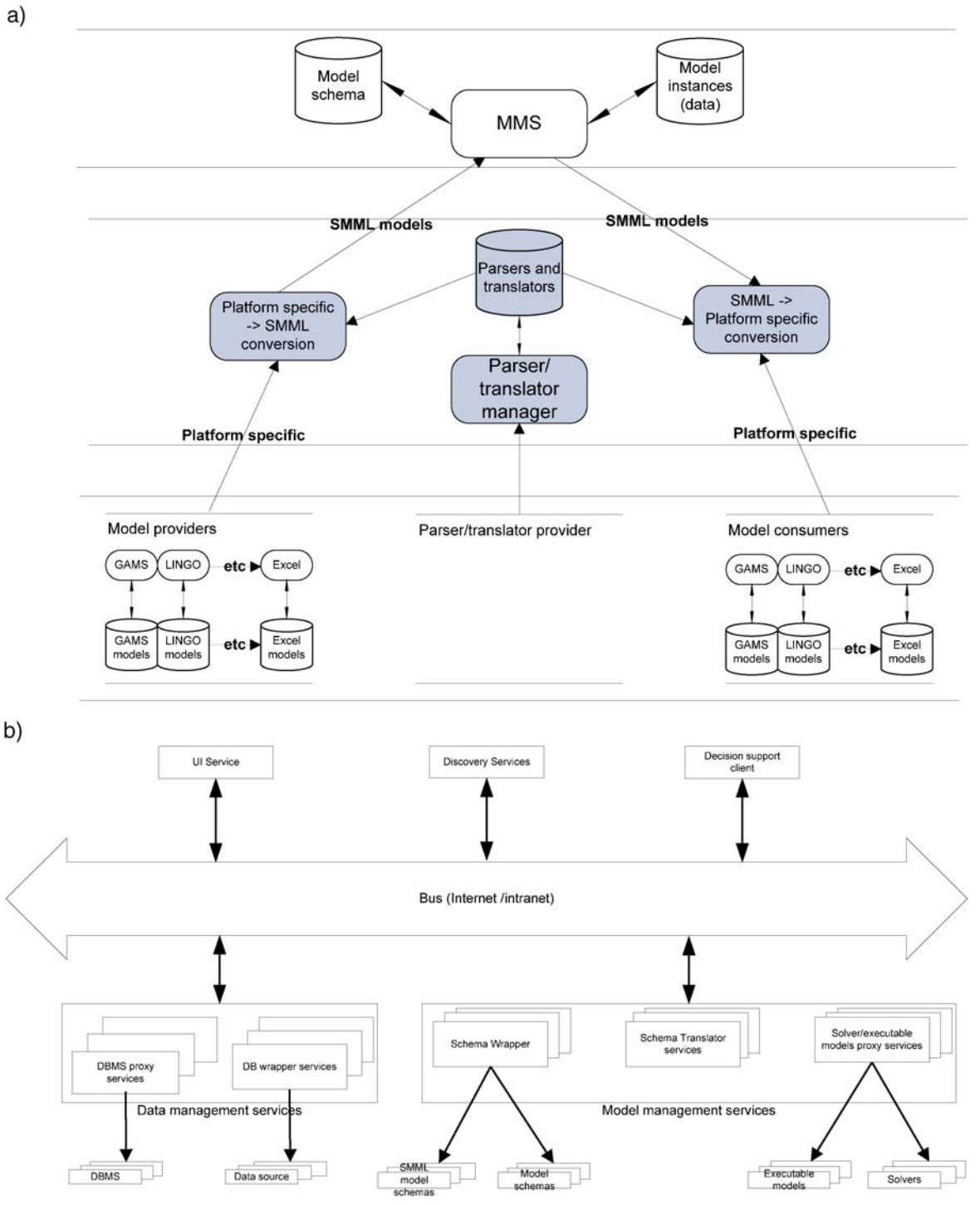

Fig. 2. System architecture for the Web-based System. a) A centralized architecture. b) A decentralized web-services-based architecture.

The client tier is comprised of model providers, model consumers, and parser/translator providers. Model providers and consumers may be using heterogeneous modeling tools and environments (platforms) like LINDO, Excel and GAMS. Using translators available through the middle tier, model providers can convert platform dependent models into SMML models shareable on the web and then upload them. Similarly model consumers can download SMML

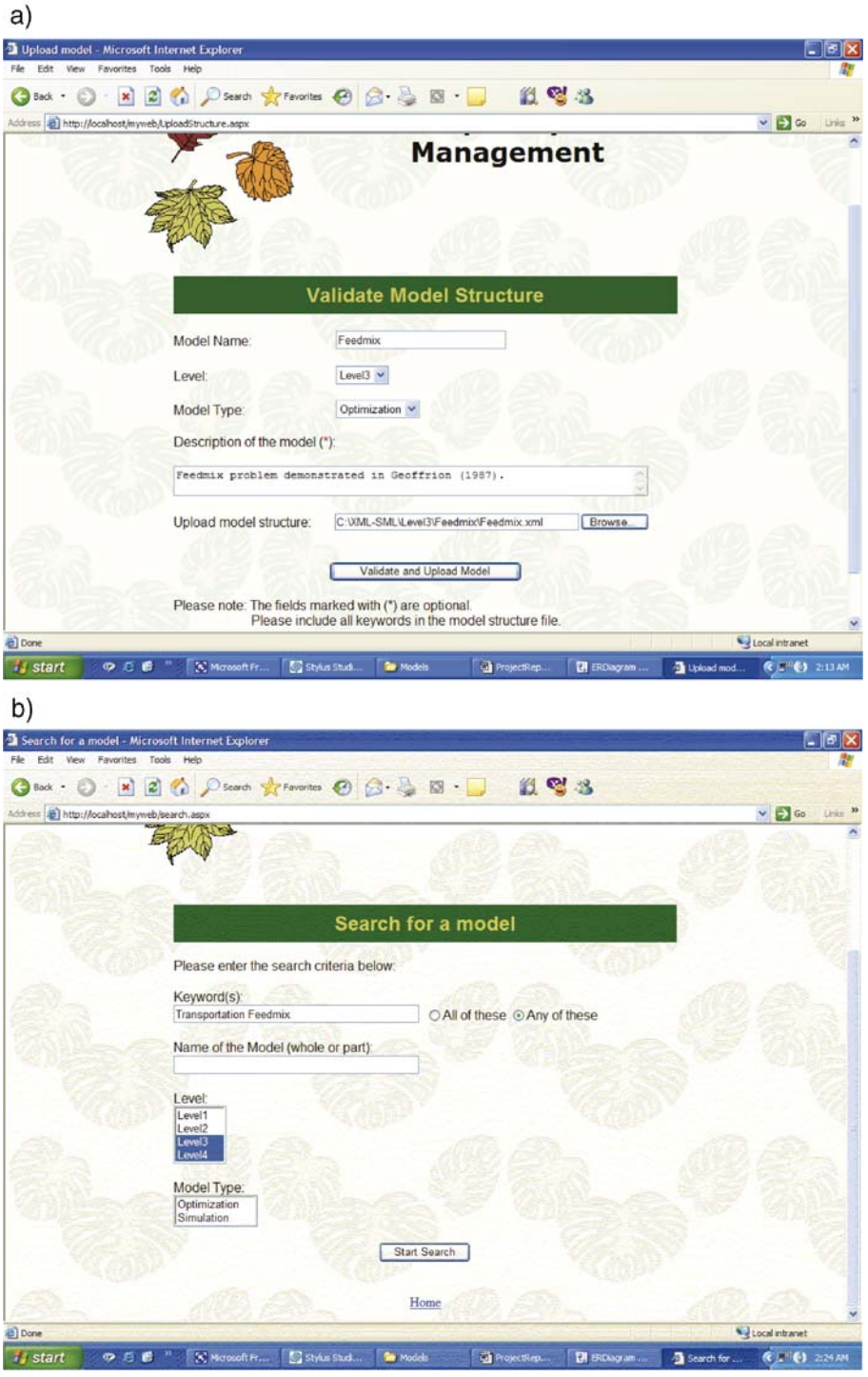

Fig. 3. Sample screens for web-based system. a) Model Schema Uploading through the Web server. b) Specifying a search criteria for models.

models and convert them to platform dependent models and use the models on these platforms.

It should be noted, however, that the components comprising the middle tier can be considered part of the server tier. Likewise, translators for individual platforms may be developed and maintained by model providers, while the development of parsers may be the responsibility of model consumers, i.e., migrating the functionality

of the middle tier to the client tier. Regardless, explicitly representing a middle tier highlights the importance of parsers and translators in the proposed architecture as well as the need to manage these parsers/translators as a resource to maximize reuse and sharing.

6.2. A decentralized web services-based architecture

Alternatively, SMML can be used to represent models in a decentralized web services-based architecture [17]. At the core of the architecture is a service bus providing the underlying communication infrastructure for the various services (Fig. 2b). The bus supports intra and inter-organization communication among services by implementing web services standards such as SOAP over HTTP. Connected to the bus is a collection of decision support services such as user interface services, database management services, and model management services. These services provide access to a variety of decision support resources such as data, models, and solvers. A decision support component acting as a client, can locate (by using discovery services) and access any of the services connected to the bus irrespective of the physical location of the service. To facilitate intra and inter-organizational communication among services, the architecture adopts XML web services in which all services communicate via Internet protocols and all data messages are sent and received as XML documents.

Model management services provide access to and management of a variety of modeling resources. These resources include specialized solvers, model schemas represented as text files, and models as executable components. Different types of services are needed to utilize the various resources. For executable models and solvers, proxy services are used to encapsulate the functionality of existing modules as web services. For model schemas represented as stand-alone non-executable SMML models, schema wrapper services encapsulate the functionality and purpose of the underlying models and coordinate the interface with other services to successfully execute these models.

In this architecture, model parsers/translators and solvers are distributed as web services. Adopting XML web services, discovery services are implemented as Universal Description, Discovery, and Integration (UDDI) server managing information about all registered services. Discovery services can be used by model producers to register their models and solvers as web services and by model consumers to locate relevant models and solvers. Similar to the centralized model, once a parser/translator is created and registered for a particular modeling environment, it is available for use by other model consumers.

7. Case studies

A number of models representing levels 1, 2, 3, and 4 of structured modeling as outlined by [28,30,31] have been successfully represented in SMML including a DOS glossary, a tournament graph, a beam deflection model from structural mechanics, an income statement spreadsheet, a contract as an example of propositional calculus modeling, a Russian roulette, a blending model, a capital planning model, an economic order quantity (EOQ) model, a feed mix model, an instructor's grade book, a Markov chain model, a Product mix model, a set covering model as an example of predicate calculus modeling, a staff scheduling model, a transportation planning model, a PERT model, a production planning model, a relational data modeling example, and a semantic data modeling example. For illustration purposes, this section describes two cases studies. Case study 1 demonstrates the use of SMML via a small, yet complete example, while case study 2 demonstrates the use of SMML in a centralized architecture through the design and implementation of a prototype system for sharing models on the Internet.

7.1. Case study 1: An SMML representation of SATELLITE model

An earth's satellite's orbit can be disturbed by gravitational forces directed toward other objects that it encounters. Based on their mass and proximity of such objects, we can calculate whether or not they pose a threat to the satellite's orbit. The force which an object exerts on a satellite is proportional to the product of the masses of the satellite and the object, inversely proportional to the product of the masses of the satellite and the object, and inversely proportional to the square of the distance between them. The proportionality factor is the gravitational constant. It is assumed that the object is a threat if it exerts enough force to accelerate one gram by one millimeter per second, i.e., if it exerts a force of at least 10^{-6}. A model schema for the Satellite problem just discussed is presented next:

```
<?xml version="1.0"?>
   <ms:MODEL name="Satellite" Level="2" type=
"Optimization"
   xmlns:xsi="http://www.w3.org/2001/XML
Schema-instance"
   xmlns:ms="http://www.dsu.edu/2004/Model
Structure"
   xsi:schemaLocation="http://www.dsu.edu/2004/
ModelStructure
   http://localhost/myweb/models/ModelStructure.
xsd
   http://www.w3.org/1998/Math/MathML
   http://localhost/myweb/models/MathML.xsd">
```

```
<KEYWORDS>
   <KEYWORD>Satellite</KEYWORD>
</KEYWORDS>
<GENUS name="SATELLITE">
   <TYPE>pe</TYPE>
   <INTERPRETATION>There is a
   <KEY_PHRASE>SATELLITE</KEY_PHRASE>
   in space.</INTERPRETATION>
</GENUS>
<GENUS name="OBJECT">
   <TYPE>pe</TYPE>
   <INTERPRETATION>There is an
   <KEY_PHRASE>OBJECT</KEY_PHRASE>
   in space.</INTERPRETATION>
</GENUS>
<GENUS name="S_MASS">
   <TYPE>a</TYPE>
   <CALLING_SEQ>
      <GENUSREF refer="SATELLITE"/>
   </CALLING_SEQ>
   <RANGE>Real+</RANGE>
   <INTERPRETATION>The SATELLITE
   has a certain<KEY_PHRASE>
   SATELLITE MASS</KEY_PHRASE>in
   kg.</INTERPRETATION>
</GENUS>
<GENUS name="O_MASS">
   <TYPE>va</TYPE>
   <CALLING_SEQ>
      <GENUSREF refer="OBJECT"/>
   </CALLING_SEQ>
   <RANGE>Real+</RANGE>
   <INTERPRETATION>The OBJECT
   has a certain<KEY_PHRASE>
   OBJECT MASS</KEY_PHRASE>in kg.
   </INTERPRETATION>
</GENUS>
<GENUS name="D">
   <TYPE>va</TYPE>
   <CALLING_SEQ>
      <GENUSREF refer="SATELLITE"/>
      <GENUSREF refer="OBJECT"/>
   </CALLING_SEQ>
   <RANGE>Real+</RANGE>
   <INTERPRETATION>The SATELLITE
   and OBJECT are a certain
   <KEY_PHRASE>DISTANCE</KEY_PHRASE>
   apart in meters.</INTERPRETATION>
</GENUS>
<GENUS name="FORCE">
   <TYPE>f</TYPE>
   <CALLING_SEQ>
      <GENUSREF refer="S_MASS"/>
      <GENUSRE refer="O_MASS"/>
      <GENUSREF refer="D"/>
   </CALLING_SEQ>
   <FUNCTION_DESC xmlns:m="http://www.w3.org/
   1998/Math/MathML">
      :
      :function description in MathML
      :
   </FUNCTION_DESC>
   <INTERPRETATION>The OBJECT exerts a certain
   <KEY_PHRASE>FORCE</KEY_PHRASE>on the
   SATELLITE in newtons, according
   to the Newton's Law of
   Gravitation.</INTERPRETATION>
</GENUS>
<GENUS name="THREAT">
   <TYPE>t</TYPE>
   <CALLING_SEQ>
      <GENUSREF refer="FORCE"/>
   </CALLING_SEQ>
   <FUNCTION_DESC xmlns:m="http://www.w3.org/
   1998/Math/MathML">
      :
      :function description in MathML
      :
   </FUNCTION_DESC>
   <INTERPRETATION>The OBJECT
   one is a<KEY_PHRASE>THREAT</KEY_PHRASE>
   to the SATELLITE if and only
   if it exerts a FORCE of greater than
   one millionth of a newton.</INTERPRETATION>
</GENUS>
</ms:MODEL>
```

Since none of the element values appear more than once, we can combine all the elements and values into a single TABLE element with a single record. A model instance is presented next:

```
<?xml version="1.0"?>
<ELEMENTAL_DETAIL name="Satellite" refer=
"Satellite.xml" xmlns:xsi="http://www.w3.org/
2001/XMLSchema-instance" xsi:noNamespaceSchema
Location="http://localhost/myweb/models/Model
Instance.xsd">
   <TABLE>
      <NAME>SATELLITE</NAME>
      <RECORD_DESC>
         <FIELD>
            <NAME>S_MASS</NAME>
            <TYPE>real</TYPE>
         </FIELD>
         <FIELD>
            <NAME>O_MASS</NAME>
            <TYPE>real</TYPE>
         </FIELD>
         <FIELD>
            <NAME>D</NAME>
            <TYPE>real</TYPE>
         </FIELD>
      </RECORD_DESC>
      <RECORD>
         <FIELD name="S_MASS" value="100"/>
         <FIELD name="O_MASS" value="2000"/>
         <FIELD name="D" value="200"/>
      </RECORD>
   </TABLE>
</ELEMENTAL_DETAIL>
```

7.2. Case study 2

There are two types of users, model suppliers and model consumers. Model suppliers upload the models to the model repository on the Web (Fig. 3a) and model consumers search and download these models to their local machines (Fig. 3b). These users may be using various modeling tools like GAMS, LINDO, Stella, etc.

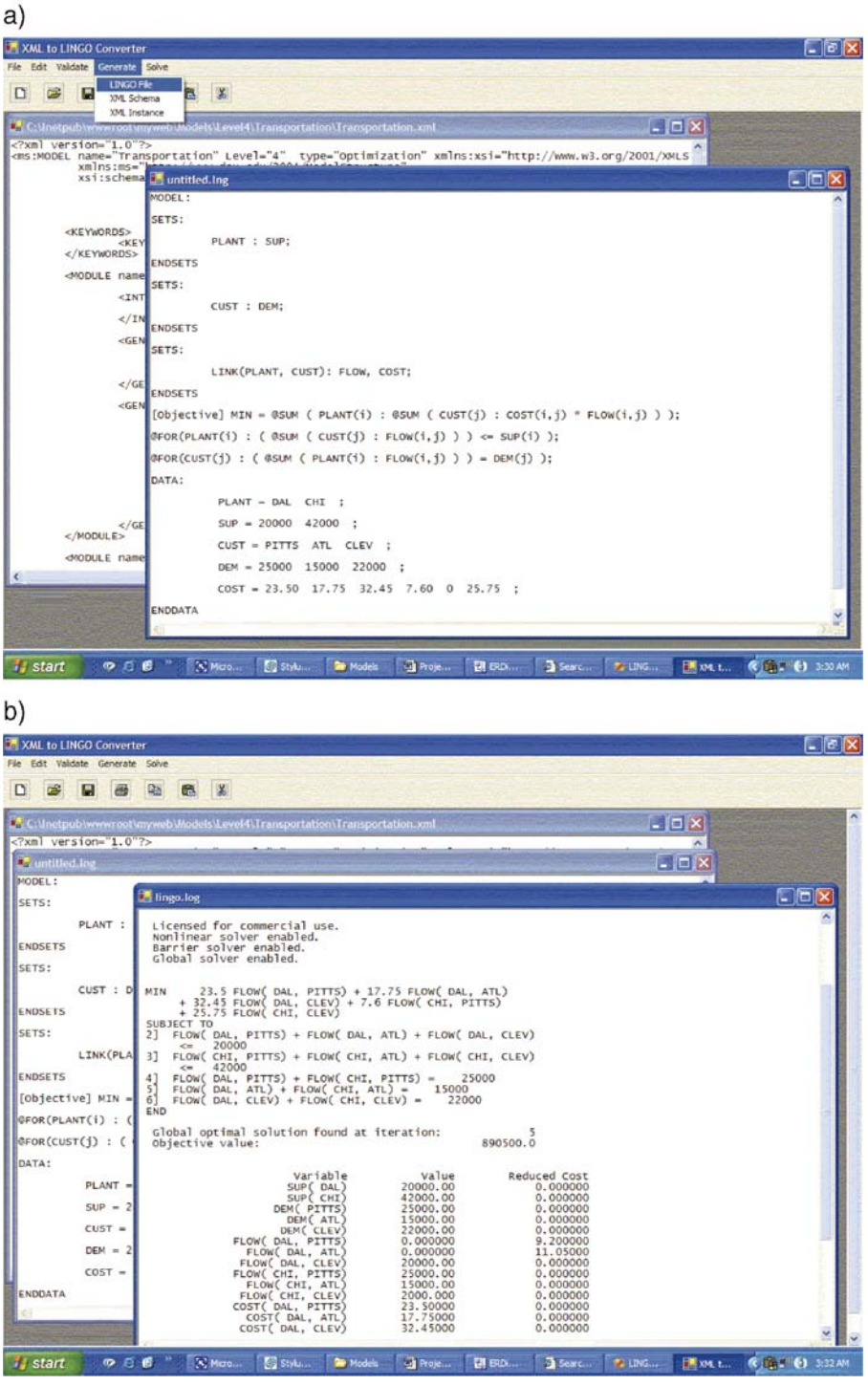

Fig. 4. LINGO Solver Application. a) SMML Model to LINGO Model Conversion. b) LINGO Solution generation using a LINGO model.

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 equations written using content markup only. Fig. 4 shows screen shots of the client-side application environment.

8. Results and conclusion

In summary, this paper proposes an XML-based schema and supporting architectures for sharing heterogeneous models in distributed environments using a standardized model representation. SMML 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 schema using a case study including server side components for maintaining a shared web-based model repository and client-side components for converting models to and from SMML.

The XML-based schema leverages the synergy between SM and XML to facilitate model representation and sharing in a distributed environment. While XML provides the means for sharing and exchanging management models, SM provides the formalism and foundation for such markup language. In effect, SMML as an XML-based language facilitates model sharing by acting as an intermediate representational mechanism for documenting the semantics and use of models. Instead of the need for creating translators between pairs of different model representation schemes, it is sufficient to create translators between each model representation scheme and SMML. Moreover, by grounding SMML in SM, we leverage desirable model representation features such as paradigm independence, model-data independence, and model-solver independence thereby facilitating model reuse. Aside from facilitating the creation of a lifetime repository of models, an XML representation of models that is grounded in theory creates unprecedented opportunities for leveraging web technologies such as web services and recent developments in enterprise development such as service-oriented architecture (SOA).

8.1. Limitations and directions for future research

From a model management perspective, the proposed schema provides a foundation for model sharing and representation. However, issues relating to leveraging the proposed developments beyond model representation and into model composition, and integration are warranted. Revisiting and extending the literature on model composition in ways that leverage XML and distributed environments is a natural next step.

Founding SMML in SM contributes to its soundness and completeness as it leverages the results of similar research on SM while having SMML as an XML-based language leverages the extensibility of XML in supporting continuous improvements of the language. However, since the research is conducted on a set of models mostly taken from [26,30,31] and the LINGO User's Guide [47], 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.

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. Future work is also needed to standardize such a modeling language through practical reviews and modifications.

On the technical front, XML supporting technologies such as Simple API for XML (SAX) and Document Object Model (DOM) provide programmatic interfaces that facilitate the creation of programs to process and produce XML document such as model translators. eXtensible Stylesheet Language Transformation (XSLT) supports the creation of translators by providing

an XML-based programming language for translating XML documents into other text formats. While such technologies facilitates the creation of model translators, issues related to the performance requirements as well as the complexity of developing parsers and translators warrants further investigation. Complexity relates to cost and can seriously affect the adoption and diffusion of such technology at the intra- and inter-organizational level. Related organizational issues relate to who (e.g., vendors, government, and non-government organizations) would create such translators and why? What are the critical success factors for adopting and implementing the proposed infrastructure? Future work is also needed to capitalize on the recent technological developments in native XML databases (NXD).

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] H.K. Bhargava, S.O. Kimbrough, Embedded languages for model management, Decision Support Systems 10 (3) (1993) 277–299.
[2] R.W. Blanning, A relational framework for join implementation in model management, Decision Support Systems 1 (1985) 69–85.
[3] R.W. Blanning, Model management systems: an overview, Decision Support Systems 9 (1993) 9–18.
[4] G.H. Bradley, Introduction to eXtensible Markup Language (XML) with Operations Research Examples, INFORMS Computer Society Newsletter, 2003.
[5] G.H. Bradley, Network and Graph Markup Language (NaGML) — Data File Formats, presented at Ninth INFORMS Computing Society Conference, Maryland, 2005.
[6] N. Bradley, The XML Companion, 3rd. ed. Addison-Wesley, Boston, MA, 2002.
[7] U. Brandes, M. Eiglsperger, I. Herman, M. Himsolt, M. Marshall, GraphML, Retrieved from http://graphml.graphdrawing.org/ on (6/12/2006).
[8] A. Brooke, D. Kendrick, E. Meeraus, GAMS: A Users Guide, The Scientific Press, Redwood City, CA, 1988.
[9] H. Bunke, Attributed programmed graph-grammars and their applications to schematic diagram interpretation, IEEE Transaction on Pattern Analysis and Machine Intelligence 4 (1982) 574–582.
[10] R. Canonico, D. Emma, G. Ventre, "An XML Based Network Simulation Description Language," presented at 7th International Symposium on Distributed Simulation and Real Time Applications (DS-RT 2003), Delft, Netherlands, 2003.
[11] D. Carlson, Modeling XML Applications with UML: Practical E-Business Applications, Addison-Wesley, Boston, 2001.
[12] A.-M. Chang, C.W. Holsapple, A.B. Whinston, Model management issues and directions, Decision Support Systems 9 (1993) 19–37.
[13] K. Chari, T.K. Sen, An implementation of a Graph-Based Modeling System for Structured Modeling (GBMS/SM), Decision Support Systems (1998) 103–120.
[14] J. Choobineh, SQLMP: a data sublanguage representation and formulation of linear mathematical models, ORSA Journal of Computing 3 (1991).
[15] Data Mining Group, Predictive Model Markup Language (PMML), Retrieved from http://www.dmg.org/ on (1/6/2004).
[16] D.R. Dolk, Model management and structured modeling: the role of an information resource dictionary system, Communications of the ACM 31 (6) (1988) 704–718.
[17] T. Erl, Service-Oriented Architecture: A Field Guide to Integrating XML and Web Services, Prentice Hall, 2004.
[18] K.A. Evans, K. Kamanna, J. Mueller, XML and ASP.NET, New Riders Publishing, Indianapolis, IN, 2002.
[19] O.C. Ezechukwu, I. Maros, OOF: Open Optimization Framework, Retrieved from http://www.doc.ic.ac.uk/old-doc/deptechrep/DTR03-7.pdf on 2006(March 9).
[20] A. Finney, M. Hucka, Systems biology markup language: level 2 and beyond, Biochemical Society Transactions 31 (6) (2003) 1472–1473.
[21] R. Fourer, Modeling languages versus matrix generators for linear programming, ACM Transactions on Mathematical Software (1983) 143–183.
[22] R. Fourer, D.M. Gay, B.W. Kernighan, AMPL: A Modeling Language for Mathematical Programming, The Scientific Press, Redwood City, CA, 1993.
[23] R. Fourer, L. Lopes, K. Martin, LPFML: A W3C XML Schema for Linear and Integer Programming, INFORMS Journal on Computing 17 (2) (2005) 139–158.
[24] R. Fourer, J. Ma, K. Martin, OSiL: An Instance Language for Optimization, Department of Industrial Engineering and Management Sciences, Northwestern University, Chicago, IL, 2006.
[25] M. Gagliardi, C. Spera, BLOOMS: a prototype modeling language with object oriented features, Decision Support Systems 16 (1997) 1–21.
[26] A.M. Geoffrion, An introduction to structured modeling, Management Science 33 (5) (1987) 547–588.
[27] A.M. Geoffrion, The formal aspects of structured modeling, Operational Research 37 (1) (1989) 30–51.
[28] A.M. Geoffrion, A Library of Structured Models, Informal note, 1990.
[29] A.M. Geoffrion, FW/SM: a prototype structured modeling environment, Management Science 37 (12) (1991) 1513–1538.
[30] A.M. Geoffrion, The SML language for structured modeling: levels 1 and 2, Operations Research 40 (1) (1992) 38–57.
[31] A.M. Geoffrion, The SML language for structured modeling: levels 3 and 4, Operations Research 40 (1) (1992) 58–75.
[32] M. Holocher, R. Michalski, D. Solte, F. Vicuña, MIDA: an open systems architecture for model-oriented integration of data and algorithms, Decision Support Systems 20 (2) (1997) 135–147.
[33] R. Holt, A. Schurr, S. Elliott, A. Winter, Graph Exchange Language, Retrieved from http://www.gupro.de/GXL/ on (6/12/2006).
[34] M. Hucka, SCHUCS: A UML-Based Approach for Describing Data Representations Intended for XML Encoding, Retrieved from http://sbml.org/specifications/notation/notation.pdf on (5/30/2004).
[35] M. Hucka, A. Finney, H. Sauro, H. Bolouri, J. Doyle, H. Kitano, A. Arkin, B. Bornstein, The Systems Biology Markup Language (SBML): a medium for representation and exchange of biochemical network models, Bioinformatics 19 (4) (2003) 524–531.

[36] S.Y. Huh, Model based construction with object oriented constructs, Decision Sciences 24 (2) (1993) 409–434.
[37] C.V. Jones, An introduction to graph based modeling system: Part 1. Overview, ORSA Journal of Computing 2 (2) (1990) 180–206.
[38] C.V. Jones, An introduction to graph based modeling systems, Part 2. Graph grammars and their implementation, ORSA Journal of Computing (1991) 180–206.
[39] S. Katz, L.J. Risman, M. Rodeh, A system for constructing linear programming models, IBM Systems Journal 19 (4) (1980) 505–520.
[40] H. Kim, An XML-based modeling language for open interchange of decision models, Decision Support Systems 31 (2001) 429–445.
[41] R. Krishnan, K. Chari, Model Management: Survey Future Research Directions and a Bibliography, Retrieved from http://www.coba.usf.edu/departments/isds/faculty/chari/model/doc.html on (3/20/2003).
[42] B. Kristjansson, Optimization Modeling in Distributed Applications: How New Technologies Such as XML and SOAP Allow or to Provide Web Services, Retrieved from http://www.maximal-usa.com/slides/Montrl02/index.htm on (February 20, 2006).
[43] M.L. Lenard, Representing models as data, Journal of Management Information Systems 2 (4) (1986) 36–48.
[44] M.L. Lenard, An object oriented approach to model management, Decision Support Systems 9 (1993) 67–73.
[45] T.P. Liang, Integrating model management with data management in decision support systems, Decision Support Systems 1 (3) (1985) 221–232.
[46] T.P. Liang, B.R. Konsynski, Modeling by analogy: use of analogical reasoning in model management systems, Decision Support Systems 9 (1993) 113–125.
[47] LINDO Systems Inc., LINGO User's Guide, LINDO Systems Inc, Chicago, IL, 2003.
[48] L. Lopes, R. Fourer, XML-Based Proposals for Optimization, Retrieved from senna.iems.nwu.edu/xml/ on (June 17, 2005).
[49] R.F. Lu, G. Qiao, C. Mclean, NIST XML Simulation Interface Specification at Boeing: A Case Study, presented at 2003 Winter Simulation Conference, Piscataway, NJ, 2003.
[50] W.A. Muhanna, An object oriented framework for model management and DSS development, Decision Support Systems 9 (1) (1993) 217–229.
[51] W.A. Muhanna, R.A. Pick, Meta-modeling concepts and tools for model management: a systems approach, Management Science 40 (9) (1994) 1093–1123.
[52] G. Qiao, F. Raddick, C. McLean, Data Driven Design and Simulation System Based on XML, presented at 2003 Winter Simulation Conference, Piscataway, NJ, 2003.
[53] B.E. Shapiro, M. Hucka, A. Finney, A. Doyle, MathSBML: a package for manipulating SBML-based biological models, Bioinformatics 20 (16) (2004) 2829–2831.
[54] R.H. Sprague, H.J. Watson, Model Management in MIS, presented at Seventeenth National AIDS, Cincinnati, OH, 1975.
[55] Y.-H. Wang, Y.-C. Lu, An XML-Based DEVS Modeling Tool to Enhance Simulation Interoperability, presented at 14th European Simulation Symposium, 2002.
[56] H.J. Will, Model management systems, in: E. Grochla, N. Szyperski (Eds.), Information Systems and Organization Structure, Walter de Gruyter, Berlin, 1975, pp. 468–482.
[57] W3C Math working group, Mathmatical Markup Language (MathML), Retrieved from http://www.w3.org/tr/2003/REC-MathML2-20031021 on (6/15/2004).
[58] B.P. Zeigler, Object-Oriented Simulation with Hierarchical Modular Models: Intelligent Agents and Endomorphic Systems, Academic Press, Boston, MA, 1990.

Omar El-Gayar Dr. El-Gayar is an Associate Professor of Information Systems at the College of Business and Information Systems, and the Dean of Graduate Studies and Research, Dakota State University. His research interests include: decision support systems, multiple criteria decision making, and the application of decision technologies in security planning and management, healthcare, and environmental management. He has an inter-disciplinary educational background and training in information technology, computer science, economics, and operations research. In addition to his academic credentials, Dr. El-Gayar has industry experience as an analyst, modeler, and programmer. He has numerous publications in various information technology related fields. He is a member of AIS, ACM, INFORMS, and DSI.

Kanchana Tandekar Kanchana Tandekar holds a Bachelor of Engineering Degree in Computer Science from Rajiv Gandhi Technical University, Bhopal, India. She also holds a Master of Science Degree in Information Systems from Dakota State University, Madison, South Dakota. She's currently working as a programmer/systems analyst at Factor 360, a web design and development company in Pierre, SD. Her research interest includes decision support systems, artificial intelligence, model management systems, data mining and warehousing.