International Institute for Applied Systems Analysis
IIASA www.iiasa.ac.at

Documentation Support for Structured Modeling Technology

Moltchanov, V.

IIASA Interim Report
December 2004

International Institute for
Applied Systems Analysis
Schlossplatz 1
A-2361 Laxenburg, Austria

Tel: +43 2236 807 342
Fax: +43 2236 71313
E-mail: publications@iiasa.ac.at
Web: www.iiasa.ac.at

Interim Report **IR-04-052**

Documentation Support for Structured Modeling Technology

Vladimir Moltchanov, vmoltcha@cc.hut.fi

Approved by

Marek Makowski
Senior Research Scholar, Risk, Modeling and Society Program

December, 2004

Foreword

This paper presents the results the author achieved during his participation in the Young Scientists Summer Program (YSSP) 2004.
However, the impact of these results is wider than it can be seen from this paper. This is because of the synergy resulting from team work.

Three participants of the YSSP 2004: Bartłomiej Prędki, Cezary Chudzian, and Vladimir Molchanov were members of the team working on the development of the Structured Modeling Technology (SMT). The other two members of the team were Michał Majdan (who spent five months at IIASA on leave from the National Institute of Telecommunications, Warsaw, Poland) and myself.

The development of SMT is a long-term challenging undertaking that requires collaborative work of researchers that have experience not only in methods and tools for advanced modeling but also knowledge and skills in DBMSs (Data Base Management Systems), XML (Extensible Markup Language), and object-oriented programming of Web-based applications.

Michał Majdan has designed the user interface to, and basic data structures of SMT. He had been coordinating the design of elements developed by other colleagues in order to be able to smoothly combine all elements into one system. This work has not been documented yet.

The contributions of the other three members of the team have been described in three Interim Reports (IRs), which constitute a kind of virtual set describing the collaborative work. I briefly summarize the scope of each IR encouraging the reader to become familiar with all of them:

Bartłomiej Prędki (IR-04-050) has implemented an extension of SMT (originally designed for algebraic models) by implementing a prototype handling of decision rule models; he has adapted a suite of software supporting applications of decision rules for analysis of qualitative data to work with SMT. Moreover, he tested the concept using a medical case study developed in collaboration with the Ottawa University.
Cezary Chudzian (IR-04-051) has developed the key elements of SMT that support a part of the modeling process composed of instance definition, specification of preferential structure for various types of model analysis, and efficient handling of underlying complex and large data structures (e.g., for parametric optimization, and diversified sets of results, both composed of huge amounts of data).
Vladimir Molchanov (IR-04-052) has explored possibilities of using XML for automatic documentation of the modeling process, and implemented a prototype of automatic documentation of model specification, which is the most difficult element of the documentation due to the complexity of the structure of the symbolic specification and the requirement for supporting gradual modifications of the descriptive part of the

documentation (which is added to the part resulting from the interactive model specification).

Finally, I would like to stress that it has been a pleasure to be the leader of the SMT team during the Summer of 2004. Each member of the team not only has very good professional skills but also abilities necessary for team work, strong dedication to achieve good results, and to have fun during the short periods spent away from the keyboard.

We plan to make the SMT publicly available in 2005. Therefore, I invite the readers to not only become familiar with the IRs mentioned above, but also to visit http://www.iiasa.ac.at/~marek in Spring 2005 to check on the further developments of SMT.

Marek Makowski

Contents

Abstract

This report describes the background and a prototype implementation of the documentation module for the modeling system, which operates according to the structured modeling technology. The presented implementation deals with the model specification stage of the modeling process. It is implemented with DBMS, OOP, Java and XML technologies. However, the proposed architecture was developed for the whole modeling process and might become fully implemented in the future.

Acknowledgments

The research described in this report has been done while the author participated in the Young Scientists Summer Program (YSSP) 2004 at IIASA under the supervision of Dr Marek Makowski.

Work has been conducted in cooperation with the members of the RMS programt who have been involved in the development of the SMT system: Michal Majdan, Cezary Chudzian and Bartolomiej Predki.

Participating in YSSP 2004 was an interesting and rewarding experience. Besides being able to gain competence in different technologies in my field of science I also had a chance to know what problems are being solved by scientists in other fields and to see new possibilities for applying Computer Science. Above all, the great working atmosphere at IIASA, which combines the internationality, team spirit and healthy competition is worth mentioning**.**

Special thanks to Karolina Werner for editing this paper.

About the Author

Vladimir Molchanov received a B.Sc. in Software Engineering from Espoo-Vantaa Institute of Technology and currently is working on his Masters degree in computer science at Helsinki University of Technology. He is also presently working as a research engineer at the Nokia Research Center (Communications laboratory).

His scientific interests include software engineering, operating systems, embedded programming and system design. He is also interested in different applications for software technologies. During the YSSP 2004 he was working on SMT implementation within the RMS project.

Documentation Support for Structured Modeling Technology

Vladimir Moltchanov*

1. INTRODUCTION

Structured Modeling Technology (SMT) organizes the modeling process into the following groups of activities; analysis of the problem, model specification, data collection and verification, model instance creation, and model analysis. Any modeling environment should adequately support the needs for documentation of each of these activities as well as the modeling process in general. All documents should be well structured, have capabilities of cross-linking from-to other elements and be portable to other representation formats, which could be used out of the scope of a given modeling environment. Documents should be either stored in DBMS or created on demand. Special attention will be given to activities that can manipulate a large amount of data. The documentation subsystem is a part of the modeling environment so it will be closely integrated with the modeling environment implementation. Technologies that are used for the implementation are XML, DBMS and OOP.

One of the activities in the RMS project is to create a modeling environment that is capable of supporting the whole modeling process, starting from the model specification up to the model analysis stage. As a part of this activity, a sufficient documentation subsystem for the modeling process is to be integrated with the modeling environment. It has to be capable of:

- Documenting the whole modeling process, while fully satisfying the needs for documentation of separate modeling activities.
- Creating and maintaining an up-to-date dynamic online help system for the modeling process, progressively including the information about earlier stages to be available for the later ones.
- Producing on-demand documentation, where it is unreasonable to store it.
- Producing persistent documentation
- Providing enough portability and styling options to make the resulting documents available for the wide distribution.
- Using DBMS technology for storage of all persistent elements of the documents

* Helsinki University of Technology, vmoltcha@cc.hut.fi

2. BACKGROUND

2.1. Context[†]

The term *modeling* is used in various contexts and different types of modeling methods are widely used.

Many commonly used models can be classified as *Algebraic Models* (AM). Following the Oxford Dictionary and a common understanding, we use the term Algebraic Model (AM) for a set of relations (such as equations or inequalities) between quantitative inputs (decisions) and outputs (performance indices) that measure consequences of implementation of decisions; further on the term *model* usually stands for *AM* unless specified otherwise. AMs are used for model-based *Decision Support Systems* (DSS) that make it possible to find solutions to real problems that are better than those that could be found without model-based problem analysis.

The term *decision support* is typically associated with management and policy-making but in practice similar model-based activities are also being performed in almost all fields of industry and research. Thus, AMs are used in a wide range of application domains including (but not restricted to) planning problems in environmental systems analysis, telecommunications, logistics, transportation, finance, marketing, production, distribution, as well as in science, research and education, whenever decisions require various analyses of large amounts of data and/or complex relations. AMs have many common analytical features; thus modeling methods and tools developed for AMs can be useful in a wide range of application domains.

2.2. Modeling Process for Decision Support

Modeling is a network of activities (or phases) often referred to as a modeling cycle, a modeling process, or a modeling lifecycle. Geoffrion (1989) provides a detailed specification of a modeling cycle, together with references to earlier works in this topic. The modeling environment that is being implemented within the RMS project aims at supporting the whole modeling process.

[†] the content of sections 2.1 and 2.2 is based on [1]

Typically, such process starts with an *analysis of a problem*, including the role of a model in the corresponding decision-making process. Subsequently, a conceptual version of a model is set up to support further discussions between modelers and users. Next, such a conceptual model together with an understanding of the problem, directs modelers to define a *model specification*. The latter is of a general nature. It is composed of mathematical (symbolic) relations, and implemented using either a general-purpose modeling tool or by developing a problem-specific model generator. Different types of variables and relations are used depending not only on the kind of problem modeled but also on the choice of a model type that is relevant to its future use, available data, and resources for model development, analysis, and maintenance. For any non-trivial problem, model specification is an iterative process that involves a series of discussions between developers (typically OR specialists) and users until a common understanding of the problem and its model representation is agreed on. Substantial changes of model specification are usually made during this process.

The most time-consuming element of a modeling process is *data collection and verification*. The data typically come from different sources (it is also often the result of analyses of other models); therefore, assembling the data and making it complete and consistent (e.g., defined in units consistent with the specification of model relations) is a resource-consuming process. Especially for large models, data management and documentation require a much more sophisticated approach than is commonly perceived.

A *model instance* is defined by the model specification and a selection of data that define the parameters of its relations. During the model implementation many model instances are created and tested to verify that the symbolic model specification is properly implemented. Model instances differ by the various selections of data used for *instantiations* of the model specification, which typically correspond to various assumptions about the modeled problem. Typically, many instances of a model are used for different sets of data corresponding to various assumptions that the user wants to examine in order to check to what extent the model adequately represents the problem. An instance of the model is also called a *substantive model* because it represents relations between variables but does not include any preferential structure.

The next phase of the modeling process is *model analysis*. A typical decision problem has an infinite number of solutions, and users are interested in identifying and examining more closely a subset of solutions that correspond best to their preferences (including trade-offs between conflicting objectives), and to various assumptions that typically result in the selection of different sets of data defining model parameters. Therefore, a properly organized analysis of a model is the essence of any model-based problem support. Properly organized, means that the user is supported in using all relevant methods of analysis, comparing the results, documenting the modeling process, and also in moving back to the first stage, whenever he/she wants to change the type of model (i.e., using a different type of variable and/or relation e.g., for handling uncertainty, or imprecision of model parameters). During the analysis of each instance of the model, different *computational tasks* are generated; each task is solved by a

solver (a software tool specialized for specific types of mathematical programming problems). Thus, the model analysis is made up of two stages: first, various instances of the model are defined and analyzed; second, a comparative analysis of the results of various analyses of instances is performed.

For a more detailed description of the Sructured Modeling Technology background and implementation see [1]

2.3. Requirements for the Documentation Support

Typically, modeling tools are designed to support a particular phase of the modeling process or even a particular modeling task. Usually, such tools use a well-established standard format(s) for their input and output data and provide proprietary capabilities for the documentation. This approach leads to a limited number of features in the documentation support and lack of cross-connectivity in documentation between different phases and elements of the modeling process.

The key requirement of the documentation subsystem of a modeling environment is to provide, in a consistent way, relevant information about:

- Model specification
- Data used for the parameters of the model
- Updates of these data
- Specification of the model instances
- Specification of the preferential structure for each instance of the model analysis
- Results from the analysis

In short, the whole tool chain should have an integrated documentation support with cross-reference capabilities for both conceptual definitions and data.

2.4. Designing the Documentation Subsystem

There are a number of steps, which are essential to take in order to produce a documentation system that satisfies the requirements stated in section 4.

The first and most critical step is to define the documents and presentation standards for each of the supported activities of the modeling process. This step requires a detailed analysis of existing tools and models for each step of the modeling process.

Secondly, each of the defined documents is to be analyzed from the structural point of view and a data structure(s) for storing and manipulating the information is to be integrated into the modeling environment database. Taking into account that some stages may produce a large amount of output (millions of items) the definition of such documents and data structures has to be done with care. Such functionality as partial browsing and online availability and possibility to update documentation of a model definition/data has to be considered for each stage.

Thirdly, the modeling environment will consist of several interconnected tools. In some cases, some tools might be substituted with other tools in order to support different types of models. The documentation sub-system should be generic enough to provide compatibility and connectivity between different stages and tools of the modeling process.

Fourthly, each document has to be either stored in the DBMS or to be created on-demand. The distinction between these two types of documents is based on the usage scenarios, storage space and creation time needed. The documentation subsystem should also be capable of cross-linking the elements of documentation between each other and maintaining the links over the changes and updates to the model.

And, finally, it should be capable of supporting one or several widely used document formats for its output.

2.5. Some Particular Challenges for the Design

There are two major challenges in the work connected with the documentation system design. One problem is to define a document model for the whole modeling process. This is not an easy task, since it has never been done before and this topic has not been covered very well in related literature. Most of the attention is usually devoted to the functionality of a modeling tool.

The second important problem that is to be considered is the possible size and variety of modeling problems, for which the modeling environment will be used. Document structures should be flexible enough to be able to deal with both model and data complexity.

3. SYSTEM DESIGN

3.1. Overall Design

As it was stated in section 2.3, the modeling process is subdivided into a number of activities. Each activity has its own requirements (i.e. tasks to be completed by it). At the moment only the documentation of the model specification phase is implemented.

The documentation process for the model specification is the following:

- SMT stand for Structured Modeling Technology. Its implementation is described in [1], [2], [3]. The SMT system generates an XML file (described later in this chapter). Such a file contains all the entities of the model with the corresponding labels and desired placement pointers within the document structure.
- A document structure definition in form of another XML file is read and a document structure is generated within the database.
- Model specification XML file is parsed and inserted into the database, which was generated in the previous step.
- A user interface based on Turbine/Velocity framework is used to enter the textual descriptions to the entities of the model specification.
- At any moment when entering the description, an output XML file can be generated. It contains both the structure and the content of the model specification.
- From the output XML file a proper visualization format could be generated via XSLT transformation. In the current implementation, the document is translated to latex format and later to ps/pdf.

3.2. Data structures

Each step of the process is based on one or more predefined data-structures to convey the information. This subsection describes the most important structures that are used.

3.3. Database Schema

One of the most challenging tasks was to design a proper database schema, which is generic enough to store the model specification and to cope with the probable changes and future developments of the system. Current schema implementation supports storage of the document structure with reasonably unlimited subsection nesting capabilities and a generic data-item storage. This was made possible by stripping item-specific details from the document database, since they are not used for the document generation at the database level. The database structure is represented by the following 5 tables:

SMTDOC_SINDEX – is a meta-data table. Used by the smt documentation system to generate the indices for the data tables. This table has only a single row entry with index 0.

idx *int* :

primary key for the table

islocked *int* :

table is locked for the moment

reader *int* :

the id number of the locking process (so that locking process could access the locked table)

Next primary keys for the corresponding data tables

models *int*

sections *int*

items *int*

texts *int*

NOTE! This table has to be initialized in order for the system to operate. In the Appendix A initialization values are discussed.

SMTDOC_MODELS – this table contains information about the model identification parameters as well as the header in XML format. It also contains pointers to the first and the last section for the document in the SMTDOC_SECTIONS table.

d_id *int*:

Primary key for the table. It is generated with the use of SMTDOC_SINDEX table.

name *varchar(40)*:

The name is the unique identification for the model in the SMT modeling system. It will be used to create or update the document data.

Header *varchar*:

This field contains the header item of the model in XML form. It includes all such things as author and date of creation, but there is no need to work with such detailed information on the database level, so it is stored directly from the input file and will be reproduced in the output XML file.

Pointers to the first and to the last sections of the document. The term *pointer* in this context refers to the foreign key of the table. It is an index of a section in the SMTDOC_SECTIONS table:

head_section *int*

tail_section *int*

SMTDOC_SECTIONS – this table contains information about subsections of the documents. It could be linked to itself, thus providing the possibility for the infinitely deep structure of the documents.

d_id *int*:

Primary key for the table. It is generated with the use of SMTDOC_SINDEX table.

model_id *int*:

To which model this item belongs. This information will be very useful for fast removal of the documents from the database.

level *int*:

Depth level of the subsection

Label *varchar*

title *varchar:*

Displayed title of the subsection and the reference label for the items to be included into this subsection.

Following data is used to create the structure of the documents within the database. In general, the document is organized as a double-linked list. Each new level is a double-linked list itself and each subsection is a double-linked list. On top of it, a subsection could contain only data-items or only subsections but not both of them. This restriction was imposed to simplify the implementation. In later versions it could be omitted.

parent_section	*int*
next_section	*int*
prev_ssection	*int*
head_section	*int*
tail_section	*int*
head_item	*int*
tail_item	*int*

SMTDOC_ITEMS - contains information about the data entities in the document. Each data entity belongs to some section of the document. The entity-specific information is stored as XML in the data field. Each item also points to the entry in the SMTDOC_TEXTS table, since each entity has an associated text entry and its 1:1 mapping.

d_id *int*:

Primary key for the table. It is generated with the use of SMTDOC_SINDEX table.

model_id *int*:

To which model this item belongs. This information will be very useful for fast removal of the documents from the database.

Following information is used to identify and order the generic data item within the model specification document.

data_id	*int*
item_order	*int*
type	*varchar*

label *varchar*

descry *varchar*

data *varchar* :

The non-generic part of the item definition is stored in the 'data' column of the table. This is supposed to be a XML document as well, however, there has been some problems with escaping certain characters for the jdbc driver of Postrgres and Oracle. For that reason, an algorithm has been implemented for XML text to be encoded into the sequence of ASCII codes delimited by the character 'b'. Whenever the data is read, the sequence is being decoded into the correct XML document.

text_id *int* :

Index of an entry in the SMTDOC_TEXTS table, which contains the text related to a given entity.

Navigation pointers used to maintain document structure.

parent_section *int*

next_item *int*

prev_item *int*

SMTDOC_TEXTS – Contains text to be added to each entity in the output document. It is important to notice, that this is the only table that would be modified with the use of the web-interface. Besides references, this table contains three fields for text to be added. In general, each data item may have a text to be displayed before it, around it and after it. Such decomposition allows for a proper display of mathematical formulas and other sort of explanations. Other tables are created/modified by the automatic scripts, some of which, however, still might be triggered by the web-interface.

It is worth mentioning, that VARCHAR type might be substituted by VARCHAR2 type for databases supporting VARCHAR2 type.

3.4. Document Structure (XML)

SMT generates an XML-format file containing a model specification. This data however does not contain any information about how the document with the model specification is to be structured. It only contains a reference to the section of the

document, where a data item belongs. Such separation between structure and the content of the document allows to manipulate both parts separately and to produce various documents suitable for different purposes. In this section the XML application used to define the structure of the model specification document is described. Same application could be used to describe most of the documents, since it concerns only partitioning the document.

Root element of the document is <*document*>. The document may contain a number of <*section*> elements.

<*section*> elements have to contain <*id*> and <*title*> elements used to identify the section and to provide a printed title correspondingly. Then, the section may contain a number of <*section*> elements (lower level subsections). Such a recursive definition basically allows for an infinite branching for the structure definition. Example of the model specification document structure definition is given in Appendix B.

3.5. Model Specification (XML)

XML-format model specification is generated by the SMT specification parser, and input to the SMT documentation module, which during the parsing process inserts the content of the specification in to the document structure and stores it in the database. Since the development of SMT is ongoing, the input data format is designed to be fairly generic. Basically, the database is used to add comments to the entities of the model, so there is no need to even completely parse the input file. This is done during the document creation time.

For the documentation module only the following parts of the input file are important:

- The root of the document is <*model*> with a single attribute *name* = <the name of the model> which is unique.
- Model has to have one <*header*> element which contains some data like author name and model creation time but its not defined precisely. All the information might be used during the document creation time, otherwise it will be discarded.
- All other elements of the root element are not precisely defined and could be named differently, but they have to have the following elements:
    - <*section*> - with desired section label
    - <*label*> - own label
    - <*description*> - with a description text to be shown for the document editor
    - <*order*> - display order within the section

- <*id*> - own model-wide unique id. Used, to have a regression update on the existing parsed document with the new input from modeling environment.

The data mentioned above is used by the documentation subsystem. All elements (under root) are saved in the database . User supplied text fields are inserted as additional XML elements inside each element generated by the SMT. Therefore the documentation subsystem is transparent for the later modification of the input-output protocols.

Note: There is a special element<*text*>. It exists to define an empty sections of the document or to add extra texts to some sections. It requires only <*section*> definition and <*para*> element, which contains the text to be inserted. SSample input file could be found in Appendix C to this document.

3.6. Output Document (XML)

An output XML document is a combined structure/data/(user text) and is generated straight from the database. An appropriate XSLT transformation application is applied to the output XML document to create different representations of a document (for example, to create a latex document). A model specification output XML document has the following structure.

Its root element is <*model*> with an attribute *name.* Model has to have a <*header*> element, which contains the general information about the model. Then the model may have a number of <*section*> elements. Sections may contain items or *subsections.*

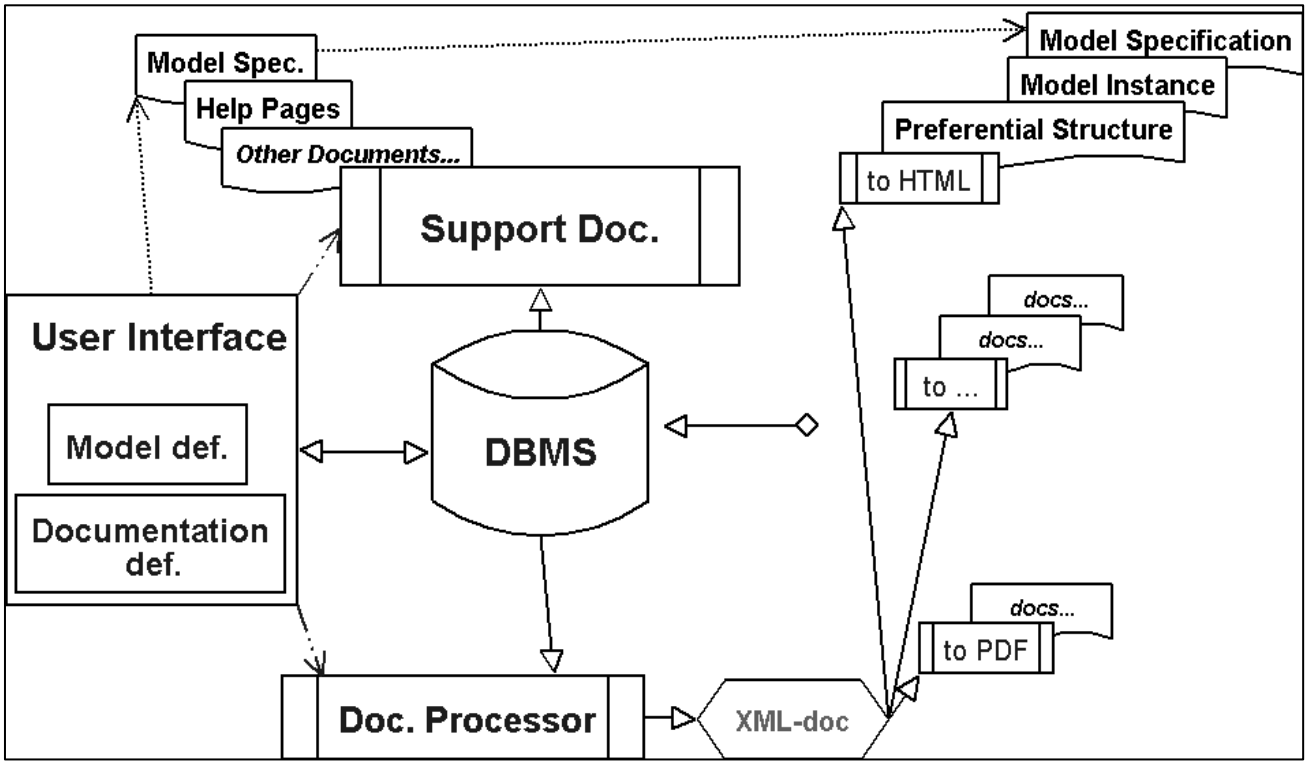
Items are elements from the input file with added <*text_before*>, <*text_about*> and <*text_after*> elements. The rest is same as in the input file, except the order. Order will be according to the desired sections and their appearance in the document.

Subsections are the same as sections. However, unlike the structure definition format, this file may contain only 3 levels of subsections, defined by <*section*>, <*subsection*> and <*subsubsection*>. The reason for this limitation is due to the specification of Latex (which is used for converting a model specification into a PDF file) that allows for a three-level section structure. However, in a typical documentation od a model specification more levels and sections would hardly be needed.

An example of the output file could be found in the Appendix D to this document.

4. SYSTEM ARCHITECTURE

The following system architecture diagram depicts an overall design for the currently implemented SMT documentation system.



The central part of the system is DBMS. It is used to store the structure of the document and to insert input data into this structure. It is also used to collect the text description from the user via the web-interface. Later, all the collected data can be extracted from the database by a document processor module in order to produce an output XML file.

User interface module has an important binding role. It is used to trigger the events for reading the input into the database, editing the text and creating output documents. It is implemented on the Jakarta Turbine container.

The document processor is a module which is used to parse and produce static documents. It is implemented with Java 1.4.

Support documents module is responsible for providing and maintaining on-demand and other non-static documents. However, it is not yet implemented.

Converter modules (or drivers) are specific to each output document type and to the operating environment. Currently model specification is generated in PDF via XSLT transformation and LaTex tools.

5. IMPLEMENTATION

We provide here an outline of Java classes developed for the prototype implementation of the SMT documentation system.

5.1. dbConnection

This class is implemented to interface the database through jdbc drivers. All the connection configuration is setup in the configuration file and will be read automatically, so that a simple call to *mConnect()* should establish a connection and change the *status* variable. *mClose()* method will close the connection. If the connection is not closed properly, the database may display an error in the log.

dbConnection also has methods *mQuery(), mUpdate(), mPrepare()* and *mReleaseQuery()*. These methods are used to execute SQL statements. For different statements different methods should be called depending on wether there is any data expected to be returned or not. mReleaseQuery() is used to the release result set after it has been processed. In general, examples of how dbConnection is used could be found

at implementation of the classes: stItems, stModels, stTexts and stSections (These classes are described in Section 5.4).

5.2. dbStored

This is a super class for stItems, stModels, stTexts and stSections. Its main functionality is to implement common operations for the mentioned classes. Namely, *mCreate()* method is a common way to operate on the unique indices, which is supposed to act as primary keys for the table. In order to differentiate the indices in the super class, stItems, stModels, stTexts and stSections are assigned a class-wide id number and all the database schema related to the index operations is defined in *iDbTables* interface and tied to the assigned numbers.

5.3. idxManager

This class manages unique indices distribution for the database. Its operation is pretty straightforward. Lock the table, get the index, increment the index, release the table. It should be used only by the dbStored class.

5.4. stModels, stTexts, stItems and stSections

These classes implement the base database access functionality for the system as well as run-time data storage capabilities. To speedup the development of these classes, the data members were set public.

Classes are capable of reading the corresponding data from the database based on given identification or writing a new data or updating existing data. Operations on document structure (like AppendSection or RemoveFromSection) are also implemented within these classes. More details can be obtained from the java code itself.

These classes need an existing database connection to work properly. *setDb()* method is used to assign a dbConnection object for the class. However, if no database operations

are needed to be performed and the object is to be used for temporary data storage in run-time, then the database doesn't have to be set.

5.5. XML_msModel, XML_msItem and XML_msSubs

These classes are derived from the classes described in section 5.4. Their purpose it to provide XML parsing/output functionality for the corresponding object. On the example of XML_msItem it is possible to see that there are in general two common methods *parseElement()* and *mOut()*. For example, the ParseElement() is invoked by the XML reading class to parse XML definition of the object and to convert it to the run-time data, which could be later saved to the database,.

mOut() is used to output the object to an XML document in a given XML representation.

XML_msItem has a bit more complicated code, because it has to deal with special cases of item declaration and have extra encoding implemented. This extra encoding is needed to overcome improper operation on the jdbc drivers with the escaped characters.

5.6. Other XML – related classes

XML_msSection, XML_msContent, XML_msStructure – these classes are the starting point for parsing the input and document structure definition files.

xmlDefinition – defines working folder for XML files as well as some other parameters of XML parsing.

5.7. smiwebItems, smiwebTexts, smiwebModels

These classes are derived from corresponding st-classes. They are implementing set and get methods for runtime data required by the web user interface. It turned out that in

Turbine/Velocity it is not possible to pass the data by public data members. No other functionality is added to these classes.

5.8. Trigger classes

The following classes:

- smtModel_Operator
- smtWriteText
- smToPdf

perform specific actions on the documentation. smtModel_Operator triggers the input parsing and update, output generation and XSLT conversion. smtWriteText is responsible for writing text in the database. It is invoked as an action from the user interface. smToPdf executes a sequence of commands that would convert a latex file into a pdf file.

Classes smDocGenerate, smXmlOut and smLatextTranslate implement triggers required by the Turbine/Velocity but in reality they create a smtModel_Operator instance and pass control to it.

5.9. User Interface Support Classes

Edit, EditMore and EditLast classes in screens support the web user interface. Their content is self-explanatory. All the described classes could be found in the appendices attached to this document. During the development work there have been a number of test applications written, but they will not be described in this document.

6. Conclusions

Structured Modeling Technology provides support for the whole modeling process. This feature makes it more important to have an adequate documentation support. During YSSP 2004 in the scope of the RMS project we have identified the key components of the documentation support system for the SMT. According to our design, the documentation support has to be based on the DBMS technology and rely on XML to pass the information between modules. In order to provide portability, web-application would be the best choice.

It is worth mentioning that one important part in designing such a system is to define formats for input, storage and output of the information. These formats should be consistent and extendable. Usage of the carefully designed XML-based application allows a documentation support system to be transparent for the data which is not meant to be processed by it. Such data will emerge unchanged at the output of the subsystem.

Due to a limited time available for the project only one part of the process has been implemented as a prototype – the model specification. It can serve as a reference implementation when the documentation support for the other parts of the modeling process will be implemented.

8. Bibliography

[1] Marek Makowski: A Structured Modeling Technology (to appear in EJOR, Feature Issue on Advances in Complex System Modeling, 2004)

http://www.iiasa.ac.at/~marek/ftppub/MM/ejor04.pdf

[2] Bartlomiej Predki*:* Qualitative Decision Models for Structured Modeling Technology

IR 04-050 (IIASA Laxenburg, Austria)

[3] Cezary Chudzian*:* Support of Model Analysis within Structured Modeling Technology

IR 04-051 (IIASA Laxenburg, Austria)

APPENDICES

Appendix A: Database Schema (Postgres)

The following SQL script was used to generate a clean database schema. Note, that it's a Postgres – specific implementation. For migrating to Oracle certain changes have been made, for instance VARCHAR was substituted with VARCHAR2 type.

Smtdoccreate.sql

```
/* smtDocCreate.sql
*       Modified: 12.08.2004 (Vladimir)
*
* Generates a new data structure within a database (dropping all the existing data!)
*
*/


/*============================================================
=============================================*/
/* 1. dropping the tables, if they do exists */

DROP TABLE SMTDOC_SINDEX;


DROP TABLE SMTDOC_MODELS;
DROP TABLE SMTDOC_SECTIONS;
DROP TABLE SMTDOC_ITEMS;
DROP TABLE SMTDOC_TEXTS;


/*============================================================
=============================================*/
/* 2. creating a new structure */


/*-------------------------------------------------------------------------------------------------------*/
/* META DATA */

CREATE TABLE SMTDOC_SINDEX (
        idx     int,

        /* lock & id of a reader */
        islocked int,
        reader  int,

        /* indices */
        models  int,
```

```
        sections  int,
        items     int,
        texts     int
);

INSERT INTO SMTDOC_SINDEX VALUES (0,0,0,1,1,1,1);


/*-------------------------------------------------------------------------------------------------------------*/
/* Models table */

CREATE TABLE SMTDOC_MODELS (
        d_id      int,
        name      varchar(40),

        /* all the information from the xml file */
        header varchar,

        /* navigation pointers to first and last section of the root level of the model */
        head_section      int,
        tail_section      int,

        /* data modifiers */

        PRIMARY KEY(d_id)
        );


/*-------------------------------------------------------------------------------------------------------------*/
/* Sections table */
CREATE TABLE SMTDOC_SECTIONS (
        d_id      int,
        model_id          int,

        /* section level */
        level     int,

        /* needed parsed data */
        label varchar,
        title varchar,


        /* navigation pointers */

        /* -- upper level */
        parent_section    int,

        /* -- same level */
        next_section      int,
        prev_ssection     int,

        /* -- lower level */
        head_section      int,
        tail_section      int,

        head_item         int,
        tail_itemint,
```

```
        /* data modifiers */

        PRIMARY KEY(d_id)
        );


/*-------------------------------------------------------------------------------------------------------*/
/* Items table */
CREATE TABLE SMTDOC_ITEMS (
        d_id      int,
        model_id          int,

        /* needed data */
        data_id  int,
        item_order        int,
        type      varchar,
        label     varchar,
        descr     varchar,

        /* all the information from the xml file */
        data      varchar,



        /* pointer to the items in the texts table */
        text_id   int,

        /* navigation pointers */

        /* -- upper level */
        parent_section    int,

        /* -- same level */
        next_item         int,
        prev_item         int,

        /* data modifiers */

        PRIMARY KEY(d_id)
        );


/*-------------------------------------------------------------------------------------------------------*/
/* Text table */
CREATE TABLE SMTDOC_TEXTS (
        d_id      int,
        model_id          int,

        /* for back linking */
        parent_item       int,

        /* this place is to be designed */
        pre       varchar,
        the       varchar,
        post      varchar,

        /* data modifiers */
```

```
PRIMARY KEY(d_id)
);
```

Appendix B: Document structure definition (XML)

```
<document>

<section>
<id>introduction</id>
<title> INTRODUCTION </title>
</section>

<section>
<id>in_sets</id>
<title>Indices and Sets</title>
<section>
<id>Index</id>
<title>Indices</title>
</section>
<section>
<id>Set</id>
<title>Sets</title>
</section>
<section>
<id>Subset</id>
<title>Subsets</title>
</section>
</section>


<section>
<id>entities</id>
<title>ENTITIES</title>
<section>
<id>C</id>
<title>Constants</title>
</section>

<section>
<id>P</id>
<title>Parameters</title>
</section>

<section>
<id>Vraibles</id>
<title>Varables</title>

<section>
<id>Vd</id>
<title>Decision Variables</title>
</section>

<section>
<id>Vo</id>
<title>Outcome Variables</title>
</section>
```

<section>
<id>Vdo</id>
<title>Decision Outcome Variables</title>
</section>

<section>
<id>Vaux</id>
<title>Auxiliary Variables</title>
</section>

</section>

<section>
<id>relations</id>
<title>Relations</title>

<section>
<id>Def</id>
<title>Definitions</title>
</section>

<section>
<id>F</id>
<title>Constraints</title>
</section>

</section>

</section>
</document>

Appendix C: A sample of input for the SMTDOC (XML)

```
<model name="ax5">

<header>
<section> Header </section>
<title> One of the simpliest LP models </title>
<acronym> ax5 </acronym>
<author> (unknown) </author>
<status> edit </status>
<created> July 19, 2004 </created>
<last_modified> July 23, 2004 </last_modified>
<spc_generated> Sat Aug 14 17:10:46 2004 </spc_generated>
</header>

<index>
<section> Index </section>
<id> 8 </id>
<label> i </label>
<description> row index </description>
<order> 10 </order>
</index>

<index>
<section> Index </section>
<id> 9 </id>
<label> j </label>
<description> column index </description>
<order> 20 </order>
</index>

<index>
<section> Index </section>
<id> 10 </id>
<label> p </label>
<description> pollution type </description>
<order> 30 </order>
</index>

<Set>
<section> Set </section>
<id> 12 </id>
<label> I </label>
<math_notation> $I$ </math_notation>
<member> $i \in I$ </member>
<description> row index </description>
<order> 10 </order>
</Set>

<Set>
<section> Set </section>
<id> 16 </id>
<label> P </label>
<math_notation> $P$ </math_notation>
<member> $p \in P$ </member>
<description> pollution type </description>
<order> 50 </order>
</Set>
```

<Subset>
<section> Subset </section>
<id> 14 </id>
<label> IJ </label>
<parent> I </parent>
<math_notation> I_{j} </math_notation>
<member> $i \in I_{j}$ </member>
<description> index subset of I </description>
<order> 40 </order>
</Subset>

(…)

<P>
<section> P </section>
<id> 38 </id>
<label> c </label>
<units> (not defined) </units>
<type> R </type>
<description> cost coefficients </description>
<order> 10 </order>
<zero_tolerance> 1.00e-06 </zero_tolerance>
<index> j </index>
<math_notation> c_{j} </math_notation>
<low_bnd> 0.0 </low_bnd>
<relation> \begin{equation}
0.0 \le c_{j}
\end{equation} </relation>
</P>

(…)

<Vd>
<section> Vd </section>
<id> 48 </id>
<label> x </label>
<units> (not defined) </units>
<type> R </type>
<description> decision variables </description>
<order> 40 </order>
<zero_tolerance> 1.00e-06 </zero_tolerance>
<index> j </index>
<math_notation> x_{j} </math_notation>
<low_bnd> lowBnd </low_bnd>
<upp_bnd> uppBnd </upp_bnd>
<relation> \begin{equation}
lowBnd_{j} \le x_{j} \le uppBnd_{j}
\end{equation} </relation>
</Vd>

<Vo>
<section> Vo </section>
<id> 40 </id>
<label> cost </label>
<units> (not defined) </units>
<type> R </type>
<description> outcome variable </description>

<order> 50 </order>
<zero_tolerance> 1.00e-06 </zero_tolerance>
<math_notation> $cost$ </math_notation>
<low_bnd> 0.0 </low_bnd>
<relation> \begin{equation}
0.0 \le cost
\end{equation} </relation>
</Vo>

(…)

<text>
<section> Vdo </section>
<para> (none) </para>
</text>

<text>
<section> Vaux </section>
<para> (none) </para>
</text>

<Def>
<section> Def </section>
<id> 41 </id>
<label> costD </label>
<units> (not defined) </units>
<type> R </type>
<description> definition of cost </description>
<order> 60 </order>
<zero_tolerance> 1.00e-06 </zero_tolerance>
<math_notation> $costD$ </math_notation>
<relation> \begin{equation}
*cost = \sum_{j \in J} c_j * x_j*
\end{equation} </relation>
</Def>

<F>

(…)

<section> F </section>
<id> 39 </id>
<label> constr </label>
<units> (not defined) </units>
<type> R </type>
<description> constraints </description>
<order> 100 </order>
<zero_tolerance> 1.00e-06 </zero_tolerance>
<index> i </index>
<math_notation> $constr_{i}$ </math_notation>
<low_bnd> lhs </low_bnd>
<upp_bnd> rhs </upp_bnd>
<relation> \begin{equation}
*lhs_{i} \le \sum_{j \in J} a_{ij} * x_j + p1 * p2 \le rhs_{i}, \qquad i \in IJ*
\end{equation} </relation>
</F>
</model>

Appendix D: A sample of SMTDOC Output (XML)

```
<?xml version="1.0" encoding="UTF-8"?>
<model name="ax5"><header>
<section> Header </section>
<title> One of the simpliest LP models </title>
<acronym> ax5 </acronym>
<author> (unknown) </author>
<status> edit </status>
<created> July 19, 2004 </created>
<last_modified> July 23, 2004 </last_modified>
<spc_generated> Sat Aug 14 17:10:46 2004 </spc_generated>
</header>

<section><title> INTRODUCTION </title></section>

<section><title>Indices and Sets</title>

<subsection><title>Indices</title>

<index>
<section> Index </section>
<id> 8 </id>
<label> i </label>
<description> row index </description>
<order> 10 </order>
<text_before>null</text_before>
<text_about>null</text_about>
<text_after>null</text_after>
</index>

<index>
<section> Index </section>
<id> 9 </id>
<label> j </label>
<description> column index </description>
<order> 20 </order>
<text_before>null</text_before>
<text_about>null</text_about>
<text_after>null</text_after>
</index>

<index>
<section> Index </section>
<id> 10 </id>
<label> p </label>
<description> pollution type </description>
<order> 30 </order>
<text_before>null</text_before>
<text_about>null</text_about>
<text_after>null</text_after>
</index></subsection>

<subsection><title>Sets</title>

<Set>
<section> Set </section>
```

<id> 12 </id>
<label> I </label>
<math_notation> I </math_notation>
<member> $i \in I$ </member>
<description> row index </description>
<order> 10 </order>
<text_before>null</text_before>
<text_about>null</text_about>

(…)

<Subset>
<section> Subset </section>
<id> 14 </id>
<label> IJ </label>
<parent> I </parent>
<math_notation> $I_{???}$ </math_notation>
<member> $i \in I_{???}$ </member>
<description> index subset of I </description>
<order> 40 </order>
<text_before>null</text_before>
<text_about>null</text_about>
<text_after>null</text_after>
</Subset></subsection></section>

<section><title>ENTITIES</title>

<subsection><title>Constants</title>

<C>
<section> C </section>
<id> 0 </id>
<label> zero </label>
<units> (not defined) </units>
<type> R </type>
<description> value of zero </description>
<order> 1 </order>
<value> 0.00e+00 </value>
<text_before>null</text_before>
<text_about>null</text_about>
<text_after>null</text_after>
</C>

(…)

<subsection><title>Parameters</title>

<P>
<section> P </section>
<id> 38 </id>
<label> c </label>
<units> (not defined) </units>
<type> R </type>
<description> cost coefficients </description>
<order> 10 </order>
<zero_tolerance> 1.00e-06 </zero_tolerance>
<index> j </index>

<math_notation> c_{j} </math_notation>
<low_bnd> 0.0 </low_bnd>
<relation> \begin{equation}
 0.0 \le c_{j}
\end{equation} </relation>
<text_before>null</text_before>
<text_about>null</text_about>
<text_after>null</text_after>
</P>

<subsection><title>Varables</title>

(…)

<subsubsection><title>Decision Variables</title>

<Vd>
<section> Vd </section>
<id> 48 </id>
<label> x </label>
<units> (not defined) </units>
<type> R </type>
<description> decision variables </description>
<order> 40 </order>
<zero_tolerance> 1.00e-06 </zero_tolerance>
<index> j </index>
<math_notation> x_{j} </math_notation>
<low_bnd> lowBnd </low_bnd>
<upp_bnd> uppBnd </upp_bnd>
<relation> \begin{equation}
lowBnd_{j} \le x_{j} \le uppBnd_{j}
\end{equation} </relation>
<text_before>null</text_before>
<text_about>null</text_about>
<text_after>null</text_after>
</Vd></subsubsection>

<subsubsection><title>Outcome Variables</title>

<Vo>
<section> Vo </section>
<id> 40 </id>
<label> cost </label>
<units> (not defined) </units>
<type> R </type>
<description> outcome variable </description>
<order> 50 </order>
<zero_tolerance> 1.00e-06 </zero_tolerance>
<math_notation> $cost$ </math_notation>
<low_bnd> 0.0 </low_bnd>
<relation> \begin{equation}
 0.0 \le cost
\end{equation} </relation>
<text_before>null</text_before>
<text_about>null</text_about>
<text_after>null</text_after>
</Vo>

<subsubsection><title>Decision Outcome Variables</title>

(…)

<text><data>none
</data></text></subsubsection>

<subsubsection><title>Auxiliary Variables</title>

<text><data>none
</data></text></subsubsection></subsection>

<subsection><title>Relations</title>

<subsubsection><title>Definitions</title>

<Def>
<section> Def </section>
<id> 41 </id>
<label> costD </label>
<units> (not defined) </units>
<type> R </type>
<description> definition of cost </description>
<order> 60 </order>
<zero_tolerance> 1.00e-06 </zero_tolerance>
<math_notation> $costD$ </math_notation>
<relation> \begin{equation}
cost = \sum_{j \in J} c_j * x_j
\end{equation} </relation>
<text_before>null</text_before>
<text_about>null</text_about>
<text_after>null</text_after>
</Def>

<subsubsection><title>Constraints</title>

(…)

<F>
<section> F </section>
<id> 39 </id>
<label> constr </label>
<units> (not defined) </units>
<type> R </type>
<description> constraints </description>
<order> 100 </order>
<zero_tolerance> 1.00e-06 </zero_tolerance>
<index> i </index>
<math_notation> $constr_{i}$ </math_notation>
<low_bnd> lhs </low_bnd>
<upp_bnd> rhs </upp_bnd>
<relation> \begin{equation}
lhs_{i} \le \sum_{j \in J} a_{ij} * x_j + p1 * p2 \le rhs_{i}, \qquad i \in IJ
\end{equation} </relation>
<text_before>null</text_before>
<text_about>null</text_about>
<text_after>null</text_after>
</F></subsubsection></subsection></section></model>

Appendix E: XSLT transform application to LaTex

```
<?xml version="1.0"?>
<xsl:stylesheet
version='1.0'
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

        <xsl:template match="/">
                <xsl:apply-templates select="model">
                </xsl:apply-templates>
        </xsl:template>


<xsl:template match="model">
\documentstyle{article}


\begin{document}

<xsl:apply-templates select="header"/>
<xsl:apply-templates select="section"/>

\end{document}
</xsl:template>

<xsl:template match="header">
\begin{titlepage}
{\LARGE \b <xsl:value-of select="title"/>}
\linebreak[5]
Acronym : <xsl:value-of select="acronym"/>
~\\
Author : <xsl:value-of select="author"/>
~\\
Status : <xsl:value-of select="status"/>
~\\
Created : <xsl:value-of select="created"/>
~\\
Last Modified: <xsl:value-of select="last_modified"/>
~\\
Space Generated: <xsl:value-of select="spc_generated"/>
~\\
\end{titlepage}
</xsl:template>

<xsl:template match="section">
~\\
~\\
{\LARGE \b <xsl:value-of select="title"/>}
<xsl:apply-templates select="subsection"/>

<xsl:apply-templates select="index"/>
<xsl:apply-templates select="Set"/>
<xsl:apply-templates select="Subset"/>
<xsl:apply-templates select="C"/>
<xsl:apply-templates select="P"/>
<xsl:apply-templates select="Vd"/>
```

```
<xsl:apply-templates select="Vo"/>
<xsl:apply-templates select="Def"/>
<xsl:apply-templates select="F"/>

</xsl:template>

<xsl:template match="subsection">
~\\
~\\
{\LARGE \b <xsl:value-of select="title"/>}
<xsl:apply-templates select="subsubsection"/>

<xsl:apply-templates select="index"/>
<xsl:apply-templates select="Set"/>
<xsl:apply-templates select="Subset"/>
<xsl:apply-templates select="C"/>
<xsl:apply-templates select="P"/>
<xsl:apply-templates select="Vd"/>
<xsl:apply-templates select="Vo"/>
<xsl:apply-templates select="Def"/>
<xsl:apply-templates select="F"/>

</xsl:template>


<xsl:template match="subsubsection">
~\\
~\\
{\LARGE \b <xsl:value-of select="title"/>}

<xsl:apply-templates select="index"/>
<xsl:apply-templates select="Set"/>
<xsl:apply-templates select="Subset"/>
<xsl:apply-templates select="C"/>
<xsl:apply-templates select="P"/>
<xsl:apply-templates select="Vd"/>
<xsl:apply-templates select="Vo"/>
<xsl:apply-templates select="Def"/>
<xsl:apply-templates select="F"/>

</xsl:template>


<xsl:template match="index">
<xsl:value-of select="text_before"/>
~\\
<xsl:value-of select="label"/> - (<xsl:value-of select="description"/>) <xsl:value-of
select="text_about"/>
~\\
<xsl:value-of select="text_after"/>
~\\
</xsl:template>

<xsl:template match="Set">
<xsl:value-of select="text_before"/>
~\\
<xsl:value-of select="label"/> - (<xsl:value-of select="description"/>) <xsl:value-of
select="text_about"/>
```

~\\
<xsl:value-of select="member"/>
~\\
<xsl:value-of select="math_notation"/>
~\\
<xsl:value-of select="text_after"/>
~\\
</xsl:template>

<xsl:template match="Subset">
<xsl:value-of select="text_before"/>
~\\
<xsl:value-of select="label"/> - (<xsl:value-of select="description"/>) <xsl:value-of select="text_about"/>
~\\
<xsl:value-of select="member"/>
~\\
<xsl:value-of select="math_notation"/>
~\\
<xsl:value-of select="text_after"/>
~\\
</xsl:template>

<xsl:template match="C">
<xsl:value-of select="text_before"/>
~\\
<xsl:value-of select="label"/> - (<xsl:value-of select="description"/>) <xsl:value-of select="text_about"/>
~\\
Units: <xsl:value-of select="units"/>
~\\
Value: <xsl:value-of select="value"/>
~\\
</xsl:template>

<xsl:template match="P">
<xsl:value-of select="text_before"/>
~\\
<xsl:value-of select="label"/> - (<xsl:value-of select="description"/>) <xsl:value-of select="text_about"/>
~\\
Units: <xsl:value-of select="units"/>
~\\
Zero tollerance: <xsl:value-of select="zero_tolerance"/>
~\\
Index: <xsl:value-of select="index"/>
~\\
<xsl:value-of select="math_notation"/>
~\\
<xsl:value-of select="relation"/>
~\\
<xsl:value-of select="text_after"/>
~\\
</xsl:template>

<xsl:template match="Vd">

<xsl:value-of select="text_before"/>
~
<xsl:value-of select="label"/> - (<xsl:value-of select="description"/>) <xsl:value-of select="text_about"/>
~
Units: <xsl:value-of select="units"/>
~
Zero tollerance: <xsl:value-of select="zero_tolerance"/>
~
Index: <xsl:value-of select="index"/>
~
<xsl:value-of select="math_notation"/>
~
<xsl:value-of select="relation"/>
~
<xsl:value-of select="text_after"/>
~
</xsl:template>

<xsl:template match="Vo">
<xsl:value-of select="text_before"/>
~
<xsl:value-of select="label"/> - (<xsl:value-of select="description"/>) <xsl:value-of select="text_about"/>
~
Units: <xsl:value-of select="units"/>
~
Zero tollerance: <xsl:value-of select="zero_tolerance"/>
~
Index: <xsl:value-of select="index"/>
~
<xsl:value-of select="math_notation"/>
~
<xsl:value-of select="relation"/>
~
<xsl:value-of select="text_after"/>
~
</xsl:template>

<xsl:template match="Def">
<xsl:value-of select="text_before"/>
~
<xsl:value-of select="label"/> - (<xsl:value-of select="description"/>) <xsl:value-of select="text_about"/>
~
Units: <xsl:value-of select="units"/>
~
Zero tollerance: <xsl:value-of select="zero_tolerance"/>
~
Index: <xsl:value-of select="index"/>
~
<xsl:value-of select="math_notation"/>
~
<xsl:value-of select="relation"/>
~
<xsl:value-of select="text_after"/>
~
</xsl:template>

```
<xsl:template match="F">
<xsl:value-of select="text_before"/>
~\\
<xsl:value-of select="label"/> - (<xsl:value-of select="description"/>) <xsl:value-of
select="text_about"/>
~\\
Units: <xsl:value-of select="units"/>
~\\
Zero tollerance: <xsl:value-of select="zero_tolerance"/>
~\\
Index: <xsl:value-of select="index"/>
~\\
<xsl:value-of select="math_notation"/>
~\\
<xsl:value-of select="relation"/>
~\\
<xsl:value-of select="text_after"/>
~\\
</xsl:template>

<xsl:template match="text">
<xsl:value-of select="data"/>
~\\
</xsl:template>


</xsl:stylesheet>
```