SANKHYA Translation Framework

Murali Desikan
Sankhya Technologies Private Limited
Tel: +91 44 28227358
Fax: +91 44 28227357
muralid@sankhya.com

Gopi Kumar Bulusu
Sankhya Technologies Private Limited
gopi@sankhya.com

ABSTRACT

In this paper, we describe a framework for developing dynamic, model-driven translation and transformation tools. The framework includes a translation modeling language using which data can be represented in multiple formats within a single model. The framework provides translation tools and Application Programming Interfaces (APIs) using which structure of input data can be validated against a model and the input can be translated to a different format specified in the model. The framework also allows data to be obtained from different sources like files, databases etc for processing.

Keywords

Model-driven translation, data integration, EAI, adapters, dynamically target-able program translation tools

1. INTRODUCTION

Activities like data integration and Enterprise Application Integration (EAI) require conversion of data from one format to another either statically or dynamically. In a data integration system, user queries will generate information from multiple data sources and they need to be collated and converted to a format required by the end-user or other applications. In an EAI environment different applications interact with each other possibly in a distributed environment and there will be a need for adapters and connectors for bridging the differences in data formats, network protocol formats, database formats etc.

There exist a very wide range of solutions and tools for data integration and application integration. These range from dedicated solutions to general-purpose integration platforms that can be customized to a specific environment.

In this paper, we provide a brief introduction to the SANKHYA Translation Framework (STF), a general-purpose framework for building model-driven translation and transformation tools. STF can be used to build dynamic model-driven parsers, data integration systems and automate EAI activities like document and message processing, protocol conversion, text to XML transformations, XML to C++ and Java code generation, server page processing, data conversion and adapter development.

The SANKHYA Translation Framework (STF) provides a powerful modeling language called the SANKHYA Translation Modeling Language (STML) for modeling language grammars, document schema, and for specifying translation and transformation rules. By using STML it will be possible to model document formats, message formats etc in multiple representations in a single unified model.

STF includes a command-line tool called the STML Translator, which is a model-driven translation and transformation tool. STML Translator can be used to automatically translate between different data representations specified using a STML model. For more complex translation and transformation requirements, the STML C++ library provided by STF can be used and extended. In addition to this, STF also provides a CORBA based server, which can be integrated with existing systems to provide document translation and data integration capabilities in a distributed environment.

2. SANKHYA TRANSLATION FRAMEWORK (STF)

The SANKHYA Translation Framework is a collection of tools and libraries for developing model-driven, dynamic tools for language parsing, data integration and EAI. The term ‘model-driven’ refers to the use of a model for capturing translations/transformations, document structure, message formats etc. The term ‘dynamic’ refers to the fact that the modeled information is external to the framework and can be changed without affecting the framework. This is in contrast to integration tools where the input and output data formats, translation logic and other conversion details are statically specified (i.e., hard coded within the application).

The high-level architecture of STF is shown in figure 1 below.

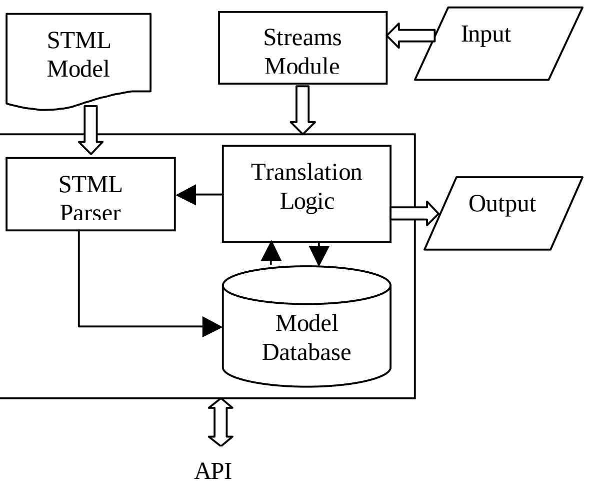

Figure 1. High-level Architecture of STF

2.1 Components of STF

2.1.1 STML Model

The STML Model consists of the STML specification of one or more modeled entities (document format, message formats etc).

2.1.2 STML Parser
The STML parser performs syntax and semantic analysis on the STML model description and creates an internal representation of the model information in the Model Database.

2.1.3 Model Database
This structure stores all the information about the entity modeled using STML Model. The translation logic consults this database for providing information to the translation tools and during translation.

2.1.4 Translation Logic
This module is responsible for translating/transforming the input present in a specific STML representation into the corresponding output representation.

2.1.5 Streams Module
STF provides a pluggable streams module using which input can be obtained from multiple sources for processing. The different data sources could be file, database, string, ftp, http and any other repository of data. STF provides a streams library that includes support for file stream, string stream and Open Data Base Connectivity (ODBC) [1] stream. The ODBC stream can be used to obtain data from any database for which ODBC support (driver) is available in the system. The stream support can be extended to implement new streams of data. The new streams can be created as dynamically linked libraries and registered with STF so that they can be used directly without any further changes to the framework.

3. SANKHYA TRANSLATION MODELING LANGUAGE (STML)
The SANKHYA Translation Modeling Language is a simple but powerful language for modeling language grammars, document structure, translations and transformations. STML can be used to establish the equivalence between multiple representations of the same information in a unified model. STML was designed with the following guidelines in mind:

1) The modeling language should be simple, text-based and easy to understand.

2) The language should allow the description of an element in many different representations within a single model. This is a very important requirement since it establishes an equivalency between these multiple representations and allows different translation tools to make use of the same model of the information.

3) The modeling language should support hierarchical description of information. It should be possible to view the modeled entity as a tree structure with a root node, internal nodes and leaf nodes.

4) The language should provide support for different types like integers, characters, strings etc. It should also provide support for specifying range of values, array types and sequence types.

5) The language should provide support for inheritance and aggregation of element properties. This will be useful for describing an element, which is a superset or subset of another element that has already been described in the model.

6) The modeling language should support constructs for easily adding or removing elements from a model. This will be useful for dynamically modifying the property of the modeled entity.

3.1 STML Elements
STML allows the depiction of document formats, message formats or any other hierarchical information by providing constructs for specifying a tree structure. These constructs are the STML Root Element, STML Node Element and STML Leaf Element.

3.1.1 STML Root Element
The STML Root element represents a node that can occur in the root of a hierarchy of the information being modeled. Each STML Root element can contain references to other STML Node elements. There can be multiple STML Root element descriptions in a model corresponding to different alternatives for matching the root element of the hierarchy.

3.1.2 STML Node Element
The STML Node element represents an internal node in the hierarchy. Each STML Node can contain references to other STML Node elements and STML Leaf elements.

3.1.3 STML Leaf Element
The STML Leaf element represents a leaf node in the hierarchy of the entity being modeled.

STML also provides the following constructs for aggregating individual STML elements:

3.1.4 STML Sequence Element
The STML Sequence element defines an element that is a repeated sequence of a previously defined STML element. This can be used to match repeated occurrence of an element in input.

3.1.5 STML Union Element
The STML Union element defines an element as a set of other STML elements, out of which it can represent any one element at any time.

3.2 STML Representations
The power of STML is that it allows an entity to be described in multiple representations (different document formats, protocol formats etc) within a single model. What this means is that once the input information is parsed according to the STML Model and the model database is created, the database will contain information about the entity in all the representations. This can then be used to convert information from one representation to any other representation specified in the STML Model.

In a STML Model, each STML Element description of the modeled entity consists of descriptions of the entity in each of the representations. An STML Representation consists of a list of

STML Values that describe the entity in a particular representation. The list of values could be constant values or reference to other STML Elements.

3.3 STML Values

An STML Value is a placeholder for data. It can either contain constant data or it can be a reference to another STML Element. An STML Representation contains a list of such STML Values. An STML Value can contain attributes (name-value pairs) associated with it. The attributes can be used to match a specific element based on the attribute values.

3.4 STML Types

STML provides certain data types for describing STML Values. These include:

a) STML Word – represents a sequence of characters from a set of characters defined in the STML Model

b) STML Any – represents any sequence of characters excluding white space

c) STML Symbol – represents a symbol that can be set externally and looked up from a symbol table

3.5 Examples

This section provides three examples of how STML can be used to model translations for tool integration, data integration and processor modeling.

1) XMI to IDL Conversion Tool:

The XML Metadata Interchange (XMI) [2] is an XML specification standardized by the Object Management Group (OMG) for interchanging metadata between modeling tools. It provides constructs to describe the Unified Modeling Language (UML) [3] specification of a system in textual format.

Consider a tool that allows a system to be modeled graphically using UML and which generates a CORBA IDL [4] specification for the UML model. This tool can provide a GUI environment for creating UML diagrams of the system. The diagrammatic design information can then be converted into XMI format. Finally, the XMI specification can be converted into the corresponding IDL. The XMI-IDL conversion can be modeled using STML and performed using STF tools.

An outline of the STML Model for modeling a simple UML class definition in XMI and IDL formats is provided below.

```
----- Start of Sample Model -----
STMLModel XMI_IDL {
        STMLLeaf ClassBegin {
                STMLWord name;
                xmi  = { "<UML:Class>", name };
                idl = { "interface", name, "{" };
        };
        STMLLeaf ClassEnd {
                xmi  = { "</UML:Class>" };
                idl = { "};" };
        };
        STMLNode ClassBody {
                // Definition not provided…
        };
        STMLNode ClassDef {
                ClassBegin cbeg;  ClassBody cbody;
                ClassEnd cend;
                xmi =  { cbeg, cbody, cend };
                idl  =  { cbeg, cbody, cend };
        };
        STMLRoot Document {
                ClassDef cd;
                xmi  = { cd };
                idl   = { cd };
        };
};
----- End of Sample Model -----
```

Explanation:

The STMLModel construct defines a model and associates a name with it. In the above example the model is given the name 'XMI_IDL'. The STML Root element 'Document' includes a reference to a UML class definition node 'ClassDef'. This node in turn contains references to ClassBegin, ClassBody and ClassEnd elements. The ClassBegin element maps a UML Class to an IDL interface. The ClassBody (definition not provided) can describe the XMI definition of a class and the equivalent IDL constructs. This can include definition of attributes, operations etc.

The above model is a very brief outline only and is provided to highlight a possible usage scenario in tool integration.

2) Text to XML Conversion:

Consider a Purchase Order (PO) for a list of items that includes the following information: Serial Number, Product Description, Unit Price, Quantity and Row Total.

Suppose that a text version of the PO is available in the following format:

```
POSTART 1 item1 100 10 1000 POEND
```

Suppose that the textual data has to be converted to XML format as specified below:

```
<PO>
   <SERIAL> 1 </SERIAL>
  <DESCRIPTION> ITEM1 </DESCRIPTION>
  <PRICE> 100 </PRICE>
  <QUANTITY> 10 </QUANTITY>
  <TOTAL> 1000 </TOTAL>
</PO>
```

To convert between the text format and XML format (and vice versa) of the PO, a STML Model can be written describing the PO in both these representations. This can be done as follows:

```
----- Start of Sample Model -----
STMLModel PO {
       STMLLeaf POHeader {
              xml = { "<PO>" };   text = { "POSTART" };
       };
       STMLLeaf POFooter {
              xml = { "</PO>" };  text = { "POEND" };
       };
       STMLLeaf SNO {
              STMLAny sno;
              xml = { "<SERIAL>", sno, "</SERIAL>" };
              text = { sno };
       };
       STMLLeaf DES {
              STMLAny des;
              xml ={"<DESCRIPTION>", des,
                       "</DESCRIPTION>" };
              text   = { des };
       };
       STMLLeaf UP {
              STMLAny up;
              xml = { "<PRICE>", up, "</PRICE>" };
              text   = { up };
       };
       STMLLeaf QTY {
              STMLAny qty;
              xml = { "<QUANTITY>", qty, "</QUANTITY>" };
              text  = { qty };
       };
       STMLLeaf RT {
              STMLAny rt;
              xml = { "<TOTAL>", rt, "</TOTAL>" };
               text  = { rt };
       };
       STMLLeaf PORow {
              SNO sno;  DES des;  UP up;  QTY q; RT rt;
              xml = { sno, des, up, q, rt };
              text = { sno, des, up, q, rt };
       };
       STMLLeaf PORowSeq : sequence (PORow);
       STMLNode POBody {
              POHeader h;  POFooter f;  PORowSeq r;
              xml = {  h, r, f };
              text = {  h, r, f  };
       };
};
----- End of Sample Model -----
```

Explanation:

In the above example the model is given the name 'PO'. Each of the components of the purchase order – viz. serial number, product description etc are described as STML Leaf elements in the model. Each of these elements has two STML Representations corresponding to XML ('xml') and text ('text') formats. Note the reference to previously defined STML Leaf elements in the definition of PORow element. The PORowSeq element specifies a sequence of PORow elements to define the rows of the PO. Finally the STML Node element, POBody, is defined as an aggregate of the individual PO components already defined as STML Leaf elements.

By providing this model to the STML Translator, the textual form of the PO can be converted to the XML representation and vice versa. This example can be extended to support other representations (like HTML) also in the same model.

3) Processor Modeling:

This example shows how STML can be used to model the instruction set architecture of a processor in two different formats – viz. assembly code and machine code. A sample processor is assumed and its general-purpose registers and one instruction are modeled.

```
 ----- Start of Sample Model -----
STMLModel Proc {
        STMLLeaf reg {
            range (i = 0, 31,1) {
                    asm = { “r$i” }; mcode = { $i };
            };
         }; // reg
        STMLNode add {
                reg r1, r2, r3;
                asm = { “add”, r1, r2, r3 };
                mcode = { or(26,6,8), r1(21,5), r2(16,5),
                            r3(11,5), or(0,11,0) };
        }; // add instruction
};
 ----- End of Sample Model -----
```

Explanation:

A sample processor, 'Proc' is modeled above. The STMLLeaf element 'reg' describes the general-purpose registers of the processor. The 'range' specification in the element indicates that 32 registers are available and named as 'r0' through 'r31' in the assembly code (represented by 'asm'). In machine code (represented by ‘mcode’), the registers have value 0-31 as indicated by ‘$i’. The STMLNode element ‘add’ describes the add instruction of the processor in assembly and machine code. The three register operands of the add instruction refer to the previously defined ‘reg’ element. The machine code representation of the instruction specifies the bit encoding of the add instruction. The ‘or’ values represent constant bit fields with starting bit, length and value specified in them. The other values in the mcode specification (for e.g., r1(21,5)) represent the encoding for each of the three register operands of the instruction.

The above model can be expanded to include all operands and instructions of the processor. An assembler can use this model and translate input assembly code to machine code for the processor. Similarly, a dis-assembler can use the model to translate from machine code to assembly code.

4. THEORY OF OPERATION

In a formal sense, a STML Model can be viewed as a unified specification of the grammars of multiple languages or document schema, which exhibit certain structural equivalence [5]. Consider a set of languages, $L_1 \ldots L_n$ with corresponding grammars $G_1 \ldots G_n$. Each of the grammars consists of a set of production rules, which specify the rules for deriving sentences of the corresponding language [6]. Let the production rule P_i of a grammar G_j be represented as follows:

$$P_i{:}Gj \qquad X \leftarrow \alpha$$

where, X is a non-terminal symbol of G_j and α is a string of terminals and/or non-terminals symbols of G_j. The production rules of the grammars $G_1 \ldots G_n$ are considered as equivalent if they represent the same concept in the corresponding languages (for e.g., a ‘sentence’ in English and French or a ‘class’ in two programming languages). Such equivalent productions rules are unified and described by a single element in STML. So, an element in STML can be viewed as a unified production rule of a set of equivalent production rules of the individual grammars.

The STML parser takes the STML description and creates a unified specification of the multiple representations. When an input is provided in any of the representations, a parse tree is created, which contains the unified STML elements as nodes. The input sentence can then be translated to an equivalent sentence in any of the representations specified using STML by traversing the parse tree and applying the production rules corresponding to the output representation.

5. STF CONFIGURATIONS

STF can be used in the following configurations:

1) Command-line Tool:

The STML Translator is a tool that can automatically translate input between different representations specified in a STML Model. The STML Model and the data to be translated are provided as inputs to STML Translator. The tool uses the STML Parser to validate the STML Model and the Translation Logic of STF to convert between the representations specified in the STML Model. The STML Translator can source data from different sources like files, URLs and databases (through ODBC) during translation.

2) C++ API:

For complex translation, transformation and document processing applications, the STML C++ library can be used instead of the STML Translator. The application can access the STML Parser, the Translation Logic and the Model Database of STF through the library. This provides greater control to the application for processing, translation and transformation tasks.

3) Client-Server:

STF is also available as a CORBA based server that can be used for developing distributed document-processing and translation applications.

6. COMPARATIVE TECHNOLOGIES

The SANKHYA Translation Framework is a novel concept for building dynamic, model-driven translation and transformation tools. To the best knowledge of the developers, no similar technology exists at present. So, direct comparison with any existing technology is not possible. However, it is possible to compare and contrast individual aspects of STF with other existing technologies. A brief summary of this is provided below.

6.1 Data Modeling

STML provides capabilities similar to the eXtensible Markup Language (XML) [7] for modeling data. XML allows description of layout and structure of data using user-defined markups. An XML processor can verify if input data conforms to a specific XML schema. In a similar way, STML provides constructs to describe structure and format of data. In addition, STML allows dynamic information (in the form of actions) to be associated with the model elements. STML differs from XML in the following ways:

1) Ability to represent multiple representations of the same data in a single model
2) Ability to attach parsing actions to an element, which will be performed when the element is processed during translation/transformation.

6.2 Translation/Transformation Modeling

The eXtensible Stylesheet Language Transformation (XSLT) [8] is a XML based language using which transformations on XML documents can be specified. XML tools can be used to transform an XML document to another XML document or HTML document based on an XSLT specification.

STF provides a powerful setup for modeling and performing translations and transformations. STF differs from XSLT based systems in the following ways:

1) General-purpose transformation modeling. Translations and transformations are not restricted to a single format (like XML).
2) Ability to execute element-specific external actions during transformation.
3) Ability to obtain information from different sources for processing.

6.3 Parsing

STF can be used to build parsers for formal languages. The language structure (grammar) can be described as a STML Model and the STML Parser and STF APIs can be used to build a lexical analyzer and parser for the language. In this respect, STF can be used as a replacement for lex, yacc [9] and other lexical analyzer and parser tools. STF provides the following features that are not available with the above-mentioned tools.

1) In STF, the grammar is specified external to the tools and hence any changes to the grammar will not require the application to be rebuilt. Whereas, in tools like yacc, the grammar specification has to be statically processed and the generated parser has to be statically linked to the application.
2) Ability to attach attributes to grammar elements and to control parsing based on attributes.
3) Ability to inherit grammar elements and attributes.
4) Ability to perform top-down as well as bottom-up parsing.

7. APPLICATIONS OF STF

The following are some application areas where STF can be used:

1) Developing data integration tools and EAI tools

   As mentioned earlier, STF can be used to automate EAI activities like document conversion, message processing, protocol conversion, data conversion and adapter development, to name a few.

2) Developing model-driven program translation tools

   STF can be extended to support modeling of microprocessors and for developing model-driven, dynamically target-able program translation tools like parser, code generator, assembler, linker and simulator. The instructions and operands of a processor can be described using STML elements. Each of these elements can contain the intermediate code, assembly code and machine code representation of the instruction and operands. The program translation tools can use the same model of a processor and translate between different representations of a program using STF. For e.g., a code generator can use the model of a processor to translate between intermediate code to assembly code (using STF) and an assembler can use the same model to translate between assembly code and machine code. The SANKHYA Tools Collection, a collection of compiler tools (code generator, assembler, simulator) being developed by Sankhya, uses STF for program translation and transformation.

3) Developing natural language translators

   STF can be used to build tools for natural language translation. The syntax of the languages to be translated can be specified using STML and either the STML Translator or a program that makes use of the STF libraries can be used to perform the translation between the languages.

4) Developing model-driven document servers

   STF can be used to develop model-driven document servers that obtain document information from multiple sources, compose the document and transform it to the format required by clients. The input format of the documents and the format required by clients can be different and hence format conversion may be required. The different document representations can be modeled using STML and the server can use the STML Model and convert the documents to the required format. This approach has been used in one of Sankhya's products to obtain processed information from application server and render them as HTML pages for use with a web browser.

8. CONCLUSION

This paper provides an overview of the SANKHYA Translation Framework. STF is a novel framework for developing model-driven, dynamic document processing and translation tools. These technologies are the result of over 4 years of Research & Development work conducted at our organization. We believe that these tools and concepts represent the state-of-the-art in the areas of dynamic and model-driven tools technology.

9. ACKNOWLEDGEMENTS

We thank the Languages and Tools Group of our organization for their contributions to the technologies mentioned in this paper and for their inputs to this paper.

10. REFERENCES

[1] ODBC 2.0 Programmer's Reference and SDK Guide, Microsoft Press, ISBN1-55615-658-8

[2] XML Metadata Interchange (XMI) Specification, Version 2.0, OMG Document

[3] Unified Modeling Language (UML) Version 1.5, OMG Document

[4] CORBA 3.0.2 Specification, Chapter 3 IDL Syntax & Semantics, OMG Document

[5] Gopi Kumar Bulusu, Murali Desikan et al. Method for Specifying Equivalence of Language Grammars and Automatically Translating Sentences in One Language to Sentences in Another in a Computer Environment (PCT Patent Application No: PCT/IN02/00159)

[6] Tremblay, J.P. and Manohar, R. Discrete Mathematical Structures with Applications to Computer Science, ISBN 0-07-463113-6, pp. 294-308.

[7] Extensible Markup Language (XML) 1.0 (Second Edition) - http://www.w3.org/TR/REC-xml

[8] XSL Transformations - http://www.w3.org/TR/xslt

[9] John R. Levine, Tony Mason, Doug Brown. Lex and Yacc, O' Reilly, ISBN 1-56592-000-7