Methodology for an implementation of the STEP standard:
a Java prototype

S. Ma*, Y. Marechal, J.-L. Coulomb

Laboratoire d’Electrotechnique de Grenoble, LEG, ENSIEG, (INPG-UJF, CNRS UMR 5529), BP 46, 38402 St Martin d’Hères, Grenoble, France

Received 6 July 1999; received in revised form 20 March 2000; accepted 22 March 2000

Abstract

The ISO standard STEP (standard for the exchange of product model data) provides a complete representation of a product, along with a description language, EXPRESS, and methods for the exchange of these product data. This paper presents the methodology that was adopted for creating a prototype of an electromagnetism-dedicated database manager, meant to support STEP compliant data exchange. The STEP architecture and its mechanisms will be described. The use of the described methodology in a Java application will be discussed. © 2000 Elsevier Science Ltd. All rights reserved.

Keywords: STEP standard; Java prototype; Code generation

1. Introduction

The new set of standards, ISO 10303, also called STEP (standard for the exchange of product model data), is intended to replace all existing standards on data exchanges. A recent study [5] concluded that STEP offers some interesting methods that could be used for the description of finite element analysis (FEA) as applied to electromagnetism.

This paper presents a methodology for implementing the STEP standard for electromagnetism applications. It benefits from the newly published STEP part 27: Java language binding to SDAI [1], which includes the description of an automatic Java code generator.

1.1. Context of this study

In the field of electromagnetism, numerical analysis is well known. But integration of the analysis software in the computer aided design environment is still an up-to-date subject. The main problem regarding integration is the exchange of product data; communication between different computer systems has to be thought out. In 1996, a consortium of researchers and service companies, the ICS STEP Committee, was created to evaluate one of these standards, STEP, as a means of exchange for electromagnetic model data. This study has been done in the framework of the ICS STEP Committee.

1.2. Contents of this paper

The next section gives an overview of the STEP standard and the EXPRESS language. Section 3 focuses on the implementation class, which describes the STEP methods to access data. It is followed by the description of an automatic code generator, which creates Java files from a model described in EXPRESS. Section 5 shows an application for testing the described methodology. The last section is a conclusion on this study.

2. Overview of the STEP standard

2.1. Needs for a standard on product data

The first standards proposed for tackling the problem of exchanging data were IGES (initial graphics exchange specification), SET (standard d’échange et de transfert) and VDA FS (verband der automobilindustrie flächen-schnittstelle). But these standards specify only a neutral format: the different computer aided manufacturing systems were asked to translate their database or a part of it and produce a file with the specified format. The main problems with these first technologies based only on a neutral format are the duplication of information and ambiguous interpretation.
of this information due to a lack of a data model. Duplicating the information wastes storage space and makes the system hard to maintain since both sets of data (the native one and the neutral one) need to be updated. The idea of exchanging the data model of the product rather than the product data comes from the experiences with IGES [2].

2.2. **STEP's structure**

STEP began to be implemented at the ISO meeting in Frankfurt am Main, Germany, in 1989. The present structure of STEP is the consequence of the large number of comments that were received. The result is that STEP is now divided into seven groups of parts, each called a class, as shown in Fig. 1.

- **Introductory ISO 10303-1**: this class contains only part 1. It presents, in 10 pages, an overview of STEP. It is like a schedule of conditions for the fundamental principles of the standard.
- **Description methods ISO 10303-1n**: this class standardizes the methods used to describe STEP entities. For example, ISO 10303-11 describes data in "normal" language and ISO 10303-12 gives a first example of the data-description language EXPRESS.
- **Implementation methods ISO 10303-2n**: this class describes physical exchange format, i.e. the structure of the files, key words, characters used, etc. It contains part 21, which specifies how physical files should be written, and part 22 (Standard Data Access Interface), which describes how to access data.
- **Resource information models ISO 10303-41 to 199**: this class defines the resources needed to represent a product. The information concerning the product is encapsulated in an implementation-independent form. This information can only be accessed via an application protocol.
- **Application protocols ISO 10303-2nn**: this class standardizes the resource information models in a specific field. An application protocol is always a specialisation of the resource information models class. The application protocol is the only way to implement STEP.
- **Conformance-testing methodologies ISO 10303-3n**: this class provides the standard procedures and tools required to test products’ compliance to the standard STEP. The procedures must give the same results if repeated. The way the product is tested must be guaranteed. This means that there must be some accredited testing laboratories.
- **Abstract test suites ISO 10303-1nnn**: this class provides a suite of abstract (not implemented by a computer) tests on products. These tests are only made on application protocols. Each application protocol is accompanied by a list of test conditions to test whether a certain implementation of a specific application protocol complies with the STEP standard.

2.3. **The EXPRESS description language**

EXPRESS is an object-oriented data-description language. It is structured in schemas, which represent the model of the product. A schema consists of entities, which are the main objects and data types that support the definitions of these entities. Within the entities are encapsulated attributes and constraints, which restrict the value of the attributes. An EXPRESS schema also has FUNCTION, PROCEDURE and RULE declarations that constrain one or more entities and types. EXPRESS is specified in ISO 10303-11 [1]. The following example shows a model described in EXPRESS.

```plaintext
SCHEMA example_geometry;
  TYPE length_measure = NUMBER;
```
3. Computer implementation of the STEP standard

3.1. ISO 10303-22: standard data access interface

This part describes how STEP compliant data can be accessed. It also specifies a meta-representation of the EXPRESS language. The meta-representation describes in EXPRESS the EXPRESS language. Fig. 2 shows a part of this meta-representation. The entity named “schema_representation” represents a SCHEMA, the entity named “entity_representation” represents an ENTITY, the entity named “defined_type” represents a TYPE and the entity named “global_rule” represents a RULE.

3.2. Standard specifications for language binding

The implementation class also has parts dedicated to mapping from EXPRESS to computer languages. For instance mapping from EXPRESS to C++ is specified in the ISO 10303-23. It gives all the functionalities that need to be supported by any STEP-compliant C++ application and specifies the construction of the C++ classes from an EXPRESS model.

4. Automatic generation of computer code

4.1. Choice of an automatic process

The conversion from EXPRESS to a language such as Java by hand is tedious. The first reason is the number of objects to be handled. Indeed, the FEA part of STEP (ISO 10303-104) is described in EXPRESS using 715 entities. The entities related to the following parts of STEP are included in this count because they were needed in the FEA part:

- Description (ISO 10303-41)
- Geometry (ISO 10303-42)
- Representation (ISO 10303-43)
- Configuration (ISO 10303-44)
- Material (ISO 10303-45)

The ISO 10303-27 part of STEP, the Java language binding, specifies that each EXPRESS entity should be converted into one interface, one class for implementing the interface, and at least another class for representing an aggregation of the entity. Thus a Java application which would use the FEA part of STEP [1] needs to convert at least 715 EXPRESS entities into 715 Java interfaces and more than 1400 Java classes. This results in a total of more than 2000 Java files and each file should contain a class or an interface with all the fields and methods specified in the Java language binding part. This number of classes is the minimum number required for only describing the objects needed in a Finite Element code.

The second reason why the conversion should not be done by hand is code maintenance. As a matter of fact, both EXPRESS and Java models describe the same data model. Thus, if the EXPRESS model is modified, the Java or any language model must be changed also. When the classes are generated automatically from the EXPRESS model, it becomes easier to keep the code updated. Even if the generated classes are modified—to add specific code for instance—some methodologies and tools, such as the Revision Control System [3], can help handling the modifications.

4.2. Code generation process—example in Java

4.2.1. A lexical and syntactical analyser

A lexical analyser checks if the tokens read from its input are legal. An analyser checks if these tokens are in a correct order from the point of view of a given grammar. Tools such as Lex (lexical analyser) and Yacc (yet another compiler compiler) are normally used to perform such analyses. They generate C code that can be compiled and used to check the correctness of a given EXPRESS model. There are some new tools more adapted to the object oriented languages: the target language is C++ (Flex++ and Bison++) or Java (JavaCC). Fig. 3 shows the process for generating an EXPRESS parser with JavaCC.

One of the reasons why we chose the Java Compiler Compiler tool is that the grammar used is very close to the Wirth Syntax Notation used in ISO 10303-11 to describe the EXPRESS grammar. For instance, production 180 is described in ISO 10303-11 as:

\[
180 \text{ case_action} \quad \text{case_label} \{', ' \text{case_label}\} '':': \text{stmt}
\]

It will be converted in a JavaCC file to:

```java
void case_action() {
    case_label( ) {
        (COMMA) case_label()*)
        (COLON) stmt( )
    }
}
```

The parser generated can state if a given EXPRESS model is a legal parse, that is to say, if the model follows the grammar
rule of the EXPRESS language. The next subsection describes how this parser can be modified to generate code.

4.2.2. A code generator

The main idea is to use a meta-representation, which would represent as closely as possible the EXPRESS model being read. Fig. 4 shows the process designed to generate Java code. First, the tokens and productions of the EXPRESS grammar are described with the Java Compiler Compiler language. The file written by hand is the same as for the parser. This file is then completed with Java code so that the parser generated can instantiate a meta-representation of the EXPRESS model being parsed. This instantiation of a meta-representation needs two different parsers. The first one instantiates the objects. The second parser calls the first parser, gets the objects instantiated and completes them to ensure coherence. This means also that each EXPRESS model is read twice. The two parsers, called ExpIdentifer1 and ExpIdentifer2 in Fig. 4, realize a lexical, syntactical and semantical analyser, which was not done in
the parser described in Section 4.2.1. Code can then be generated because each instantiated object of the meta-representation is able to write itself in a given language. The specifications for the code that must be generated in a given language are given in the STEP language binding parts as follows:

- ISO 10303-23: C++ language binding
- ISO 10303-24: C language binding
- ISO 10303-26: IDL language binding
- ISO 10303-27: Java language binding
- ISO 10303-xxx: XML language binding

5. Application for solving an electromagnetism problem

The previous code generator was tested for the generation of Java classes that solve FEA problems in electromagnetism. The test was based on a Java application made for pedagogic purpose in an engineering school [4]. First the EXPRESS model of the needed objects was made. For instance an ENTITY of this SCHEMA was the following:

```
ENTITY Element
ABSTRACTSUPERTYPEOF(OONEOF(ElementL02, ElementT03))
SUBTYPE OF (ObjectMef);
   nodes_: ARRAY[0:?] OF Node;
END_ENTITY;
```

Then this EXPRESS model was converted into Java code with the code generator. This resulted in about 70 interfaces and classes. Then the methods for solving the finite element problem were added to some classes. The class generated for the element ENTITY was:

```
package SDAI.SMef1;
public abstract class CElement extends CObjectmef implements EElement {
   ...
   protected ANode nodes_;
   protected boolean nodes_Set;
   public ANode getNodes_(EElement type)
      throws SdaiException {
         return nodes_;}
   ...
}
```

This application uses classes which should comply with the Java language binding specified in ISO 10303-27: Java Language Binding to SDAI; these classes are able to solve a FEA problem in electromagnetism and write the solution in the STEP neutral file format. This problem was tested is a meshed section of a motor. A part of the neutral file generated was as follows:

```
#259 = NODE('260', 2, (0.0851679444, −0.123476379),
   0.0., 1, 10000);
#260 = NODE('261', 2, (0.0994792432, −0.11226698),
   0.0., 1, 10000);
#261 = NODE('262', 2, (0.112266988, −0.0994792357),
   0.0., 1, 10000);
#262 = NODE('263', 2, (0.123422861, −0.0852454901),
   0.0., 1, 10000);
#263 = ELEMENTT03('1', (#122, #106, #105));
#264 = ELEMENTT03('2', (#195, #151, #1));
#265 = ELEMENTT03('3', (#25, #24, #23));
#266 = ELEMENTT03('4', (#33, #32, #31));
```

6. Conclusion

The STEP standard specifies a most interesting mechanism for describing and exchanging data. It should induce a better database structure and enable applications to exchange data with more success than with the other standards. This paper presented a methodology which will be used for the design of an electromagnetism-dedicated database meant to support STEP compliant data exchanges. This methodology could also be applied to other fields, as it was designed to support most of the generic specifications described in STEP. Some future studies will be done to make a more complex STEP model for simulation in electromagnetism and the methodology described in this paper will be used to validate this model.

References


