The STEP standard as an approach for design and prototyping

Alain Plantec, Vincent Ribaud

The STEP Standard as an Approach for Design and Prototyping *

Alain Plantec
SYSECA, 34 quai de la Douane,
29285 Brest Cedex, France
plantec@univ-brest.fr

Vincent Ribaud
Faculté des Sciences, BP 809,
29285 Brest Cedex, France
ribaud@univ-brest.fr

Abstract

STEP is an ISO standard (ISO-10303) for the computer-interpretable representation and exchange of product data. Parts of STEP standardize conceptual structures and usage of information in generic or specific domains. The standardization process of these constructs is an evolutionary approach, which uses generated prototypes at different phases of the process. This paper presents a method for the building of prototype generators, inspired by this standardization process, together with a tool used to support the method. Throughout stages of model integration, the embedded logic of prototype generators is defined. The successive stages focuse on data model integration but this integration relies on a common agreement about construct functionalities under elaboration.

Introduction

STEP is a standard developed to facilitate product information sharing by specifying sufficient semantic contents for data and their usage. Parts of ISO 10303 are intended to standardize conceptual structures of informations either generic, or within a subject area (e.g. mechanics). STEP advocates an evolutionary approach in development of data specifications and uses prototypes at each phase of this development.

Prototyping cycles build different kinds of prototypes, from screen mock-up to pilot systems. Horizontal prototyping tends to produce horizontal layers of the software like user interface with forms and menus, or functional layer such as database transactions. Vertical prototyping completely implements a selected part of the application [4].

This paper describes a method inspired from the STEP standardization process, for the generation of application prototypes and a tool for its support. It is used to generate some horizontal layers such as database access interface and user interface, and vertical components related to a particular subject area.

The paper is organized as follows: section 1 introduces STEP terminology and concepts, section 2 discusses the integration process and architecture of the STEP approach. Section 3 describes our method and section 4 presents some applications and perspectives.

1 STEP

As described in [5], STEP is an International Standard for the computer-interpretable representation and exchange of product data. The objective is to provide a neutral mechanism capable of describing product data throughout the life cycle of a product independent from any particular system. The nature of this description makes it suitable not only for neutral file exchange, but also as a basis for implementing and sharing product database and archiving.

A fondamental concept of STEP is the definition of consensus data specifications that describe the data to be exchanged or shared and that cover some particular application domain. These data specifications are called *application protocols* [12]. In order to define and implement *application protocols*, the information modelling language EXPRESS [6], the standard file exchange format [7] and a standard data access interface (SDAI) [8] was developed.

1.1 The EXPRESS language

EXPRESS is an object-oriented modelling language and EXPRESS-G is a graphical subset of EXPRESS. The application data are described in schemata and a schema can reference other schemata. This allows the designer to write generic schemata referenced by more specific ones. A schema owns the type definitions and the object descriptions of the application called *Entities*. An entity is made of attributes and constraint descriptions (see an example in figure 1).

* Final version submitted at RSP'98

```
FUNCTION leap_year (year : year_number) : BOOLEAN;
  IF (((year MOD 4) = 0) AND ((year MOD 100 <> 0)))
     OR ((year MOD 400) = 0)
    THEN RETURN (TRUE);
    ELSE RETURN (FALSE);
  END_IF;
END_FUNCTION;

ENTITY ordinal_date SUBTYPE OF (date);
  day : day_in_year_number;
WHERE
  WR1 : (NOT leap_year(SELF.year) AND 0 < day ≤ 365)
        OR (leap_year(SELF.year) AND 0 < day ≤ 366)
END_ENTITY;
```

Figure 1. An EXPRESS entity

1.2 The STEP physical file format

A STEP file is an exchange structure using a clear text encoding of product data for which a conceptual model is specified in the EXPRESS language.

1.3 The Standard Data Access Interface

The SDAI [8] defines an access protocol for *EXPRESS*-defined databases and is defined independently from any particular system and language. The representation of this functional interface in a particular programming language is referred to as a language binding in the standard. As an example, ISO 10303-23 is the STEP part describing the C++ SDAI binding [9].

The main goals of the SDAI are to access and manipulate data which are described using EXPRESS, to allow access to multiple data repositories by a single application at the same time, to allow commit and rollback on a set of operations, to allow access to EXPRESS definition of all data elements and to allow the validation of the constraints defined in EXPRESS.

1.4 Application protocols

An *application protocol* (AP) is a part of STEP that defines the context, scope and information requirements for designated domain(s) and specifies the STEP ressource constructs used to satisfy these requirements [12]. APs were first proposed as a means of ensuring that STEP would enable a more reliable way of exchanging product data. APs define the form and contents of a block of data that is to be exchanged in such a way that claims of conformance to the standard for particular software products can be properly tested. In order to avoid overlapping between APs, STEP provides *integrated resources* (IR) that are common generic data constructs used by APs and *application interpreted constructs* (AIC) that are common usages of the same generic data constructs taken from *integrated resources*. Figure 1 shows the *ordinal_date* IR. The entity *drawing_revision_id* presented in figure 2 is taken from the standard AIC *Draughting*.

```
ENTITY drawing_definition;
  drawing_number : identifier;
  drawing_type : OPTIONAL label;
END_ENTITY;

ENTITY drawing_revision SUBTYPE OF (presentation_set);
  revision_identifier : identifier;
  drawing_identifier : drawing_definition;
  intended_scale : OPTIONAL text;
INVERSE
  sheet : SET [1:?] OF drawing_sheet_revision FOR in_sets;
UNIQUE
  UR1 : revision_identifier, drawing_identifier;
WHERE
  WR1 : . . .
END_ENTITY;
```

Figure 2. An AIC

An AP is made of three kinds of information models: the *application activity model* (AAM), the *application reference model* (ARM) and the *application interpreted model* (AIM).

An AAM describes the activities and processes that use and produce data in a specific application context. An ARM is dependent upon the AAM and contains the conceptual structures and constraints used to define the information requirements of an application context. An AIM consists of a selected set of *integrated resources* which are specialized, constrained or completed to satisfy the requirements of the ARM.

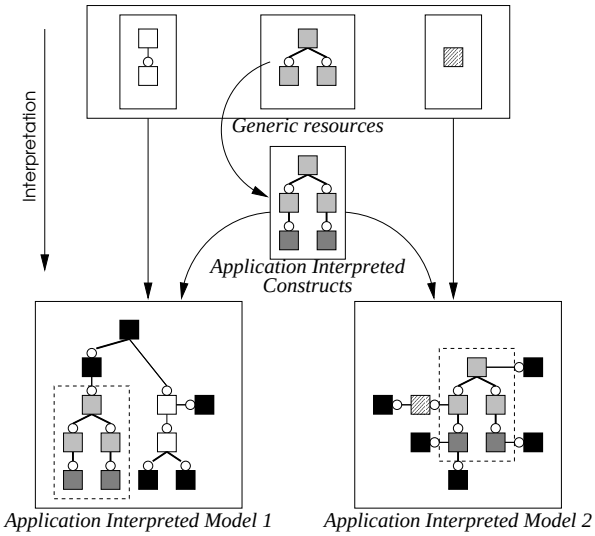
2 The STEP integration framework

2.1 A conceptual integration framework

The STEP integration framework is defined as establishing an explicit architecture for the conceptual models that are part of STEP. An application is related to a domain (such as manufacturing, avionics). Each domain may dispose of one or several *application protocols* which define the form and the contents of the data used by the business activities of the domain.

The goal of an AP development process is the definition of an AIM. The AIM specification is based on the reuse of generic resources: an AIM is an EXPRESS schema that selects the applicable constructs from the *integrated resources* as baseline conceptual elements. Then, this schema is specialized with additional constraints, relationships and entities inheriting from generic constructs. This process is called *application interpretation* (see figure 3), it assigns a

meaning to generic resources in the context of a particular ARM. The *application interpretation* process is a formal and well established part of the AP development process.



Several AIMs can reuse the same data structures. The standard integrated resources provide these common data structures. Several AIMs can have the same interpretation of the same generic resources. This common usage of generic resources is specified in AICs.

Figure 3. Generic resources reuse in the STEP application interpretation process

As an example, in the ISO standard *application protocole* 201, *Explicit draughting*, an *Approval* is information that indicates that a drawing has been reviewed and accepted. The date of an *Approval* is the *integrated resource* depicted in figure 1. A *Drawing* is a pictorial and textual presentation of product data. *Drawing_definition* and *drawing_revision* are taken from the *application interpreted construct* partly presented in figure 2.

Designing the data schema is a kind of model integration. While data semantic is the sound of this integration, we should not forget that each *application protocol* relies on an *application activity model* (a functional point of view related to the domain) and that the SDAI defines a set of data management operations. Hence, there is a lot of functional services expressed in the data model by the underlying services and operations expected.

2.2 The resulting architecture

We have specified a method for building generators and implemented a generator builder (see figure 5 depicting this generator builder). We have adopted a solution similar to *Stage*, described in [2]. Generators translate specifications described in a source language into products written in a target language. The source specifications handled by generators are those specified from classical application design: OMT class models, database data definition language (DDL) schemata or EXPRESS schemata. The target language is typically a programming language, a database data manipulation language (DML) or an hypertext documentation.

Using generators built with our approach, code components are generated: SDAI set of operations, *application protocol* services, neutral file exchange, user help or instances browser. These components need to interoperate with the application specific code and the infrastructure components (the services offered by the target platform). Mixing specific code, generated code and reusable libraries, requires some architectural decisions. As programming tool writers, we are influenced by the architecture proposed in PCTE and by the different levels of integration defined in [14]. The kind of integration furnished by the architecture is a data integration, supported by the overall use of the SDAI in each component. The platform furnishes some infrastructure components (database, GUI, communications). The SDAI manages the shared repository. Instances browser, help and neutral file exchange are kind of horizontal tools. *Application protocol* services are kind of vertical tools, such as specific code components (see figure 4).

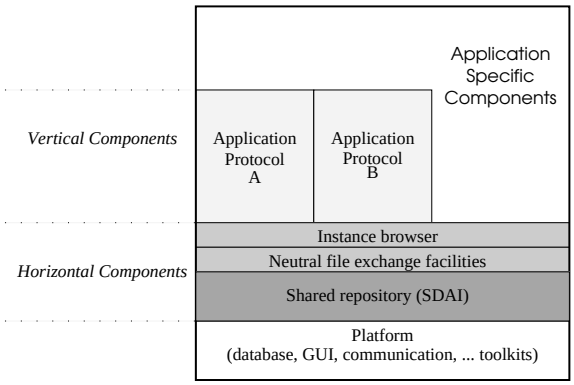


Figure 4. Architecture of the STEP integrated framework

3 Application prototyping with STEP

3.1 Method overview

The method specified in this section is based on the STEP technology: EXPRESS is used to describe data, the SDAI is

used to instanciate EXPRESS schemata, to compute derived attributes and to process *traduction functions*.

The method is first based on the description of the source and of the target language data constructs and on the description of *traduction parameters* that shall be used by the traduction process. These descriptions are defined in EXPRESS within two meta-models and the *traduction parameter* schemata. These schemata are used by another schema that consists in the target language meta-model interpretation. The resulting description, called the *interpreted meta-model*, serves as the basis for evaluation of *traduction functions* that are written in EXPRESS within the *traduction model*. The result of this evaluation consists in one part of the target application prototype. Figure 5 shows the specification, the building and the utilisation of a generator.

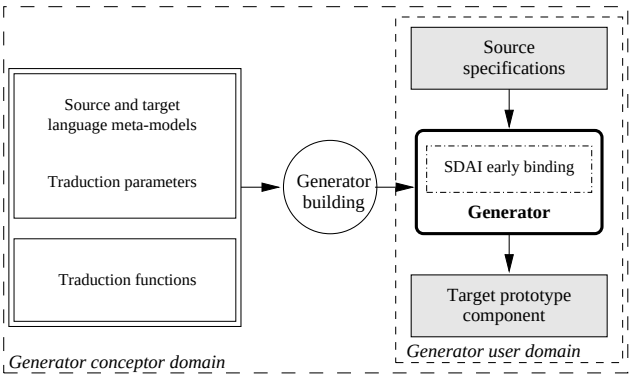


Figure 5. Building of a generator

3.2 Data models

We illustrate the main concepts of this section with examples taken from a generator intended to translate from OMT class models to SQL schemata.

3.2.1 Source and target language meta-models

The source and target language meta-models consist of a set of EXPRESS schemata that describe the source and target language data constructs. These schemata are instanciated with definitions of the type, attribute, object and data schema concepts. This is called *Metaprogramming* in [1].

The building-block entities used by the language meta-models consist in a description of a common set of basic language description concepts. Schemata containing those basic entities are considered as the *generic resources* (e.g. *SCHEMA metamodels_IR*).

Parallel to the idea of STEP *application protocol*, the source (e.g. *SCHEMA omt_dictionary*) and target (e.g. *SCHEMA sql_dictionary*) language meta-models are considered as *application interpreted constructs*.

```
SCHEMA metamodels_IR;
  ENTITY model_def;
    name : STRING;
    objects : LIST OF object;
  END_ENTITY;
  ENTITY object;
    name : STRING;
    attributes : LIST OF attribute;
  END_ENTITY;
  ENTITY attribute;
    name : STRING;
    domain : attribute_domain;
  END_ENTITY;
END_SCHEMA;                                  – metamodels_IR
```

```
SCHEMA omt_dictionary;
  USE FROM metamodels_IR;
  ENTITY omt_model SUBTYPE OF (model_def);
    SELF\model_def.objects : LIST OF class;
  END_ENTITY;
  ENTITY class SUBTYPE OF (object);
    superclasses : LIST OF class;
  END_ENTITY;
END_SCHEMA;                                  – omt_dictionary;

SCHEMA sql_dictionary;
  USE FROM metamodels_IR;
  ENTITY sql_schema SUBTYPE OF (model_def);
    SELF\model_def.objects : LIST OF table;
  END_ENTITY;
  ENTITY table SUBTYPE OF (object);
    SELF\object.attributes : LIST OF column;
  END_ENTITY;
  ENTITY column SUBTYPE OF (attribute);
    SELF\attribute.domain : simple_type;
  END_ENTITY;
END_SCHEMA;                                  – sql_dictionary
```

We use a similar approach in another tool, *idefix* [13], intended to automatically build database management applications from managed object descriptions.

3.2.2 The traduction parameter schema

The *traduction parameter* schema (e.g. *SCHEMA traduction_parameters*) consists of a set of entities that describe data that are used by the traduction process. It is specific to the application and target system, and is defined or altered by the generator conceptor. It aims to describe data useful for the naming of target programming constructs such as the class or type names. It can also contain target system descriptions such as the name of basic classes used by produced classes. The *traduction parameter* schema is a kind of

descriptions of programming rules used by a project team.

```
SCHEMA traduction_parameters;
  ENTITY naming;
    prefix : STRING;
  END_ENTITY;
END_SCHEMA;                                  – traduction_parameters
```

3.2.3 The interpreted meta-model

The interpreted meta-model (IMM) schema (e.g. *SCHEMA omt_to_sql*) contains all constructs of the target meta-model schema. Subtypes of the entities from the target meta-model schema are created. The creation of subtypes enable more specific attribute definitions to be given in the context of the source language meta-model schema and of the traduction parameters schema. The context is represented in the subtypes by associations between them and entities from the source language meta-model and the traduction parameters.

The goal is to redefine all attributes of the created subtypes of the target meta-model schema as derived attributes in order to compute their value in the given context. The idea is very similar to the STEP *application interpretation* process described in section 2.1.

3.3 The traduction model

The *traduction functions* are written in EXPRESS and are specified in the *traduction model* schema (e.g. *SCHEMA omt_to_sql_traduction*). Inputs of these functions consist of instances of the *interpreted meta-model*. The outputs consist of a textual target program or documentation.

Parallel with the idea of a STEP *application protocol*, the *traduction model* is considered as a representation of the AAM. The basic idea is that, for a particular AP, containing data descriptions in the AIM and underlying functionalities description in the AAM, a target representation of the AP in a given programming language can be automatically generated. The method enables the constitution of high level libraries which are a form of design reuse.

4 Practical experiments

4.1 The tools *daigen* and *adoc*

daigen is a generator that is able to automatically translate source application data schemata written in EXPRESS to a target textual representation. *daigen* is now used by three projects: to generate an early binding of the SDAI in Smalltalk-80 and in Java, and to generate Java graphical data browsers.

```
SCHEMA omt_to_sql;
  REFERENCE FROM traduction_parameters;
  REFERENCE FROM omt_dictionary;
  USE FROM sql_dictionary;
  ENTITY omt2sql_schema SUBTYPE OF (sql_schema);
    from_model : omt_model;
    naming : naming;
  DERIVE
    SELF\sql_schema.objects : LIST OF omt2table
      := build_tables(SELF);
  END_ENTITY;
  ENTITY omt2table SUBTYPE OF (table);
    from_class : class;
  DERIVE
    naming : naming := sql_schema.naming;
    SELF\object.name : STRING
      := naming.prefix + from_class.name;
    SELF\table.attributes : LIST OF omt2column
      := build_columns(SELF);
  INVERSE
    sql_schema : omt2sql_schema FOR objects;
  END_ENTITY;
  ENTITY omt2column SUBTYPE OF (column);
    from_attribute : attribute;
  DERIVE
    naming : naming
      := table.sql_schema.naming;
    SELF\attribute.name : STRING
      := naming.prefix + from_attribute.name;
    SELF\column.domain : simple_type
      := from_attribute.domain;
  INVERSE
    table : omt2table FOR attributes;
  END_ENTITY;
END_SCHEMA;                                  – omt_to_sql
```

adoc is a generator that produce *LaTeX* programmer reference documentation from the analysis of comments contained in sources of programs.

4.2 An example of a vertical component

Telecommunications use a lot of formal methods and languages to describe various models of systems, using GDMO. GDMO uses ASN.1 syntax but has no rigourous way to describe the behaviour of managed objects. An achieving PhD Thesis [10] has pointed out the interest of STEP/EXPRESS in the OSI management area. As an application of this work, the V5 interface (used between telecommunication network and management network) was fully specified using GDMO/EXPRESS. This specification was used to produce a prototype of a V5 handler in Smalltalk-80.

5 Conclusion

This paper has presented a method and a tool to build generators. The method is closely related to the STEP integra-

```
SCHEMA omt_to_sql_traduction;
  USE FROM omt_to_sql(omt2sql_schema);
  FUNCTION create_table (table : omt2table) : STRING;
    LOCAL
      text : STRING;
      attr : LIST OF column := table.attributes;
    END_LOCAL;
    text := 'CREATE TABLE ' + table.name + '(';
    REPEAT no := LOINDEX(attr) TO HIINDEX(attr);
      text := text + attr[no].name + ' ';
      text := text + get_attribute_type(attr[no]);
      IF no < HIINDEX(attr) - 1 THEN
        text := text + ', ';
      END_IF
    END_REPEAT;
    text := text + ');'
    RETURN (text);
  END_FUNCTION;
END_SCHEMA;                                  – omt_to_sql_traduction
```

tion framework as it is used to standardize *application protocols*. The generators are intended to produce toolkit prototypes (horizontal layers) as well as functional component prototypes (vertical layers).

As stated before, STEP concentrates on data specifications consensus and provides a structured, pragmatic and reliable method to realize this consensus. This consensus is related to a more informal functional model integration. When integrating data models (AIMs), it supposes a kind of integration of AAMs. The integration process of activity models, even formally defined with IDEF0, can not be so precise and reliable as data models integration. It relies more on system analysts, designers and programmers.

Usually, a good design reuses practical implementation experiments. The effervescence around design patterns shows that the reuse problem has moved from code reuse to design reuse [3]. The traduction functions for the various experiments recalled above were written and tuned after a successful hand-made implementation, relying on a good comprehension of the domain requirements and functionalities. In a sense, these traduction functions capture design and "know-how" of the application, in similar technique such as templates or schemas [11]. The successful development of traductions functions for a given domain needs a narrow cooperation among domain experts, system analysts and prototype designers, exactly as preconised in STEP for an *application protocol* development.

References

[1] Y. Ait-Ameur, F. Besnard, P. Girard, G. Pierra, and J. C. Potier. Formal Specification and Metaprogramming in the EXPRESS langage. In *Int'Conf' on Software Engineering and Knowledge Engineering (SEKE)*, 1995.

[2] J. C. Cleaveland. Building Application Generators. *IEEE Software*, July 1988.

[3] E. Gamma and al. *Design patterns*. Addison-Wesley, 1996.

[4] M. S.-H. H. Lichter and H. Zullighoven. Prototyping in Industrial Software Projects – Bridging the Gap Between Theory and Practice. *IEEE Transaction on Software Engineering*, 20(11), November 1994.

[5] ISO 10303-1. *Part 1: Overview and fundamental principles*, 1994.

[6] ISO 10303-11. *Part 11: EXPRESS Language Reference Manual*, 1994.

[7] ISO 10303-21. *Part 21: Clear Text Encoding of the Exchange Structure*, 1994.

[8] ISO 10303-22. *Part 22: Standard Data Access Interface*, 1994.

[9] ISO 10303-23. *Part 23: C++ Programming Language Binding to the Standard Data Access Interface Specification*, 1995.

[10] C. Jacolot. *La technologie STEP/EXPRESS dans le domaine de la gestion des réseaux de télécommunications*. PhD thesis, CNET Lannion, 1998 (to appear).

[11] C. W. Krueger. Software Reuse. *ACM Computing Surveys*, 24(2), June 1992.

[12] M. Palmer. Guidelines for the development and approval of STEP application protocols. Technical report, ISO TC184/SC4/WG4 N511, 1995.

[13] A. Plantec and V. Ribaud. Data Management: From EXPRESS Schemata To User Interface. *Journal of Computing and Information*, 2(1), November 1996.

[14] A. I. Wasserman. Tool Integration in Software Engineering Environments. In *Lecture Notes in Computer Science, Software Engineering Environments*, pages 137–149. Springer-Verlag, 1989.