Meta-programming with Express and SQL

Philippe Saliou, Alain Plantec, Vincent Ribaud

Meta-programming with EXPRESS and SQL

Philippe Saliou, Alain Plantec and Vincent Ribaud

{*Philippe.Saliou,Alain.Plantec,Vincent.Ribaud*}*@univ-brest.fr*

EA2215, Département d'informatique, Université de Bretagne Occidentale,
BP 809, 29285 Brest Cedex, France

Preface

Meta-programming can be defined as creating application programs by writing programs that produce programs (application generators).

Meta-programming is programming and meta-data management is data management. STEP is an ISO 10303 standard intended to data modeling and exchange. STEP standard defines very useful software tools, as EXPRESS an object-oriented modeling manguage, and STEP technology can be applied for the design and the implementation of application generators.

The position of the paper is that SQL-PL/SQL or EXPRESS declarative and imperative features are powerful enough to build applications generators.

1 Introduction

A lot of formal models (entity-relationship, relational, object-oriented...) are used at different phases of system life cycle. The programming environments often offer the possibility to describe all the used models. This description is structured in a dictionary or *meta-model*, which, following Codd's idea [1] is usually implemented with the same programming environment constructions as the models themselves. Software engineering (CASE) tools and programmers use these meta-informations to transform models from one phase to another or to generate programs and documentation.

Programming using meta-informations is usually called metaprogramming. The programmer's effort is primarily in the development of application-code generators to produce the deliverable software [4].

Application generators translate specifications into products (programs, documentations). They parse specifications statements into data structures (called *dictionaries* or *meta-models*), from which desired products can be derived.

An application generator builder offers a means to define specification languages and associated parsers, to describe and traverse the meta-model structure and to specify the code generation from this structure. In most builders, code generation is specified with templates (or skeletons) of code. Templates contain a mixture of commands operating on the meta-models and "real" code directly inserted into the products.

The meta-modeling approach can be made easier by the use of several meta-models and a way to link it. Declarative or imperative techniques are then employed to define the relationships between meta-models.

Meta-data management is data (information) management. STEP is an ISO 10303 standard developed to facilitate product information sharing by specifying sufficient semantic content for data and their usage. Parallelizing the STEP standardization process, we defined a method intended for the building of application generators [5].

One major component of a CASE tool is the repository. A repository holds the meta-models definitions as well the models themselves. The repository itself is usually implemented using either a relational or an object-oriented database management system.

We have employed Oracle Designer for several industrial projects according a waterfall process. Designer supplies transformers in order to generate models along the phases of the project. When supplied-transformers are not well-suited, either a dedicated-transformer can be built or the work is accomplished manually.

Oracle repository is implemented in SQL and PL/SQL (a procedural extension). SQL owns declarative features which can be used for metaprogramming. However, in most of cases, we prefer to use Eugene, a STEP-based framework intended for the building of application generators. Meta-models, meta-models relationships and code templates are programmed with the object-oriented data modelling language, EXPRESS.

2 Building an application generator

2.1 The building process

Application generators translate source specifications into target products as programs or documentations. The problem can be seen as belonging to the field of compilation: an intermediate representation is built through an analysis (parsing) process and this representation is used by a synthesis process to produce a product in a target language.

As in the compiler field, intermediate representations, close to the target, reduce the code generation work. As source representation, an intermediate representation is stored in a meta-model. The structure of this meta-model is related to the target language. In order to obtain this meta-model, called *interpreted meta-model* (IMM), we defined a method which is similar to the method used in the STEP standard to define an application model from standardized models.

The generated application is obtained through three steps (see figure 1): analysis of source specification, building of the IMM, generation of the target representation.

2.2 Data models

The source and target language meta-models describe the source and target language data constructs. These descriptions are called meta-model schemata.

The translation parameters schema consists of a set of entities describing other data that are used by the translation process. 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. This schema is considered as a description of a part of programming rules usually described and used for building and for integration of an application within a target system.

STEP description and implementation methods

The EXPRESS language [1] is an object-oriented modelling language. The application data are described in schemata. A schema has the type definitions and the object descriptions of the application called *Entities*. An entity is made up of attributes and constraint descriptions. Entities may inherit attributes and constraints from their supertypes.

The STEP physical file format defines an exchange structure using a clear text encoding of product data for which a conceptual model is specified in the EXPRESS language. The mapping from the EXPRESS language to the syntax of the exchange structure is specified in [2].

The Standard Data Access Interface (SDAI) [3] defines an access protocol for *EXPRESS*-modelled 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 [4].
The five main goals of the SDAI are: **(1)** to access and manipulate data which are described using the EXPRESS language, **(2)** to allow access to multiple data repositories by a single application at the same time, **(3)** to allow commit and rollback on a set of SDAI operations, **(4)** to allow access to the EXPRESS definition of all data elements that can be manipulated by an application process, and **(5)** to allow the validation of the constraints defined in EXPRESS.
An SDAI can be implemented as an interpretor of EXPRESS schemata or as a specialized data interface. The interpretor implementation is referred to in the standard [3] as the SDAI late binding. An SDAI late binding is generic in nature. The specialized implementation is referred to in the standard as the SDAI early binding.

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. 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*.

References

1. ISO 10303-11. *Part 11: EXPRESS Language Reference Manual*, 1994.
2. ISO 10303-21. *Part 21: Clear Text Encoding of the Exchange Structure*, 1994.
3. ISO DIS 10303-22. *Part 22: Standard Data Access Interface*, 1994.
4. ISO CD 10303-23. *Part 23: C++ Programming Language Binding to the SDAI Specification*, 1995.

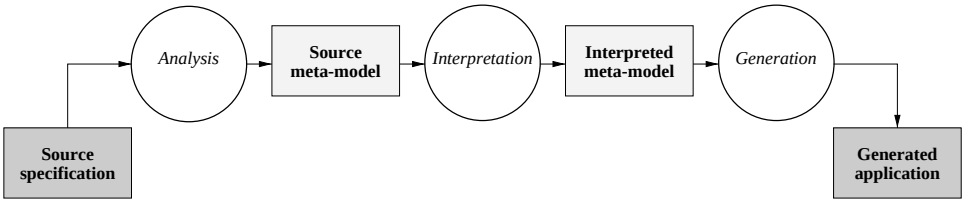

Fig. 1. Generation with an intermediate meta-model

2.3 The interpretation process

The interpreted meta-model (IMM) schema 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 allows more specific attribute definitions to be given in the context of the source language meta-model schema and of the translation parameters schema. The context is represented in the subtypes by associating them with entities from the source language meta-model and from the translation parameters. The goal is to redefine all the attributes of subtypes of the target meta-model schema as derived attributes in order to compute their value in the given context.

3 A concrete example

3.1 The context

The three authors worked previously in a software company, Thales-IS Brest, within a small team (3-6 persons) developing a project named *ARIANE* : the management of the textile department of a supermarket chain. Technical choices made at the beginning of the project (1995) and still valid are Oracle for the database management system and VisualBasic and SQL for the client software. System analysis and design is done with the help of Designer/2000; the repository is continually updated and SQL DDL code (the database schema) used in the project is always obtained by the code generators of Designer/2000.

Since 1998, a part of the team's effort has been devoted to developing and maintaining a family of VisualBasic generators, called *GARI* (for Generator ARIane).

Four generators have been built and are used :

- *gari_iobdd* generates interface VB data access functions from SQL table descriptions,
- with *gari_prc* functions calling stored procedures are generated from the procedures descriptions,
- *gari_sql* embeds SQL clauses in VB functions,
- *gari_taico* handles the mapping between VB two-dimensional matrices and flat SQL tables.

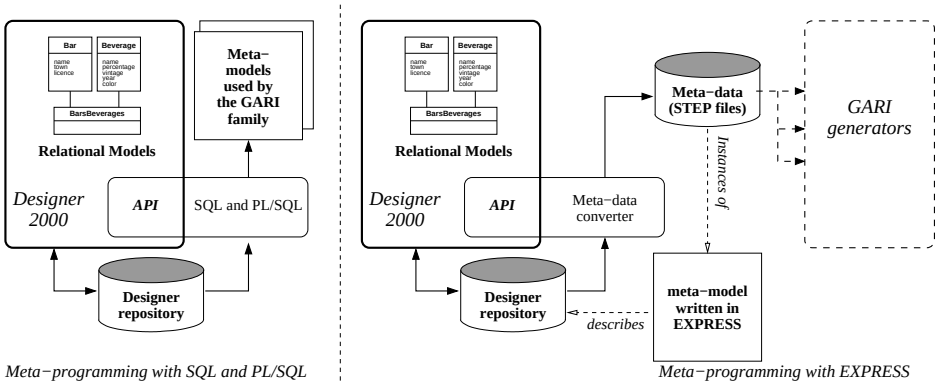

Fig. 2. Cooperation between Designer/2000 and GARI with an intermediate translation tool

So the problem lies on providing inputs to GARI generators with a guaranteed and automatic consistency with the Designer/2000 repository. This will provide a seamless integration of all CASE tools used in the project.

3.2 Possible solutions

Meta-programming with SQL and PL/SQL Designer/2000 provides an Application Programming Interface (API) to the repository. The API is a set of database views and PL/SQL packages that allow safe access to the repository data (meta-data).

Since the repository is a standard SQL database, GARI meta-models could be implemented in a same way.

Thus, the interpreted meta-model consists in a set of views and tables based on the repository tables. When the repository meta-data are changing, views are automatically updated and tables need to be refresh with the help of triggers and stored procedures.

Pros and cons The use of SQL and PL/SQL make easier the integration between tools.

As a matter of fact, the repository consists of a relatively small number of tables that store the source meta-data. These tables have complex (undocumented) relationships. There are, however, many views of these tables that represent repository objects, such as entities and attributes. These views are an important part of the API because they allow us to examine the definition of objects created through the transformers [2]. Unfortunately, if working on SQL DDL statements is straightforward, this is not true with SQL DML statement, specially if they are complex.

Meta-programming with EXPRESS The source language is SQL and each GARI generator has its own interpreted meta-model.

All the meta-models can be specified with EXPRESS (the source meta-model is a subset of the SQL meta-model). We need then a translation tool able to produce EXPRESS meta-data schema from the repository meta-data.

Pros and cons The seamless integration is obtained through three steps (see our current implementation depicted in figure 2): analysis and design using Designer/2000 tools, generation of the EXPRESS meta-data, generation of the VB code with the GARI family. It took few weeks to make the meta-data converter. It works as expected and provides consistency. But this consistency is possible because there is no semantic loss between the information needed in the repository and the translation in EXPRESS.

4 Declarative features

The Eugene method is independent of the language used to describe meta-model and to program it. We often claim that SQL extended with an imperative language such as PL/SQL would be a good candidate for implementation of the Eugene environment.

Moreover, the declarative features that we use in meta-programming can be mostly found in SQL.

4.1 Why EXPRESS

We choose EXPRESS for two main reasons :

Powerful modeling The structure of any meta-model is easily modeled with an object-oriented EXPRESS schemata.

SDAI generation An SDAI for the meta-data management system is generated. This requires naturally an SDAI generator suited to the target system, but such an SDAI generator is re-used for each project available within this target system.

4.2 The EXPRESS language

EXPRESS is an object-oriented modelling language. 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 types definitions and the objects descriptions of the application called *Entities*. An entity is made of attributes and constraints descriptions and can inherit from others entities. The constraints expressed in an entity definition can be of several kinds [3], briefly:

- the *UNIQUE* constraint allows entity attributes to be constrained to be unique either singly or jointly (e.g any one value of that (these) attribute(s) is (are) associated with only one instance of the owner entity),

- the *DERIVE* constraint is used to represente computed attributes. Such constraint specifies the way derived attributes are computed,
- the *WHERE* clause of an entity constraints each instance of an entity individually,
- the *INVERSE* clause is used to specify the inverse cardinality constraints.

EXPRESS allows the definition of global rules. These rules are used when either all instances of a given entity or instances of at least two entities need to be examined concurrently to determine whether a given constraint is satisfied.

5 Conclusion

We need a cooperation between different CASE tools, especially if we wish to guarantee consistency. This requires access to the CASE tool repositories. STEP is an ISO standard (ISO-10303) for the computer-interpretable representation and exchange of product data. We successfully used STEP framework to produce SDAI automatically from the repository meta-modeling, and using this standard meta-data access more easily than the dedicated repository API.

References

1. C. J. Date. *An Introduction to Database Systems, vol. 1.* Addison Wesley, Reading MA, 1990.
2. Paul Dorsey and Peter Koletzke. *Designer/2000 Handbook.* McGraw-Hill, 1998.
3. ISO 10303-11. *Part 11: EXPRESS Language Reference Manual*, 1994.
4. Leon S. Levy. A Metaprogramming Method and Its Economic Justification. *IEEE Transaction on Software Engineering*, 12(2), February 1986.
5. Alain Plantec. *Exploitation de la norme STEP pour la spécification et la mise en œuvre de générateurs de code.* PhD thesis, Université de Rennes I, 35065 Rennes cedex, France, 1999.