HAL
archives-ouvertes.fr

Experiences using an Application Generator Builder

A Plantec, Vincent Ribaud

▶ To cite this version:

A Plantec, Vincent Ribaud. Experiences using an Application Generator Builder. 12th IEEE International Conference on Software Engineering and Knowledge Engineering (SEKE'99), Jun 1999, Kaiserlautern, Germany. <hal-01450893>

HAL Id: hal-01450893
http://hal.univ-brest.fr/hal-01450893

Submitted on 3 Feb 2017

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Experiences using an Application Generator Builder

A.Plantec and V.Ribaud
Syseca & LIBr
Syseca: 34 quai de la Douane, 29200 Brest, France
LIBr: Faculté des sciences, Département d'Informatique, 29285 Brest Cedex, France
E-mail:{plantec,ribaud}@univ-brest.fr

Abstract

Application generators translate specifications into products (programs, documentations. An application generator builder offers a way to define specification languages and associated parsers, to describe and traverse the meta-models structure and to specify the derivation on this structure.

STEP is an ISO 10303 standard developed to facilitate product information sharing by specifying sufficient semantic content for data and their usage. STEP technology offers very useful software tools and can be applied for the design and the implementation of application generators. EUGENE is a STEP-based framework, that is intended for the building of application generators.

After a concise presentation of EUGENE, this paper presents different application generators built with this framework within an industrial project. Then, different properties of generators are examined to establish the kind of generators which can be successfully built with EUGENE. Finally, we conclude with the main benefits of our approach.

1 Introduction

Application generators translate source specifications into target products as programs or documentations. They use data structures called dictionaries or meta-models to store the source specifications and intermediate representations. As in compilers, there are two main functions in a generator: parsing of the source specifications and the target code generation.

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 derivation on this structure. In most builders, derivation 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.

STEP is an ISO 10303 standard developed to facilitate product information sharing by specifying sufficient semantic content for data and their usage. Within STEP, data are modelized and manipulated with a dedicated technology, mainly an object oriented modelling language EXPRESS and a data access interface.

Application generator building can benefit from the STEP technology at specification and implementation levels. We used these capabilities in a tool, called EUGENE, that is intended for the building of application generators [6]. The meta-models are defined with EXPRESS schemata; code templates are programmed in EXPRESS and are interpreted by a STEP data access interface.

EUGENE is briefly presented in chapter 2 with a small example to illustrate the concepts. Chapter 3 discuss different generators built with EUGENE. Chapter 4 tries to characterize the « suitable » candidates to build with EUGENE.

2 Building application generators with EUGENE

An application generator built with EUGENE uses meta-data in order to generate a target textual representation. Meta-data come from a source specification analysis and are stored in different kind of meta-model. Derivations are programmed with imperative functions, called translation functions. They are made of fixed parts that are mainly either meta-model traversal routines or string constants directly put into the target and made of variable parts that are values fetched from the meta-models.

Figure 1 shows that an application generator built with EUGENE is only a process that consumes meta-data and produces some realization. The meta-data are themselves produced by other processes or tools that can use the automatically built SDAI in order to write standard STEP files.

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.

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

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 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.

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.

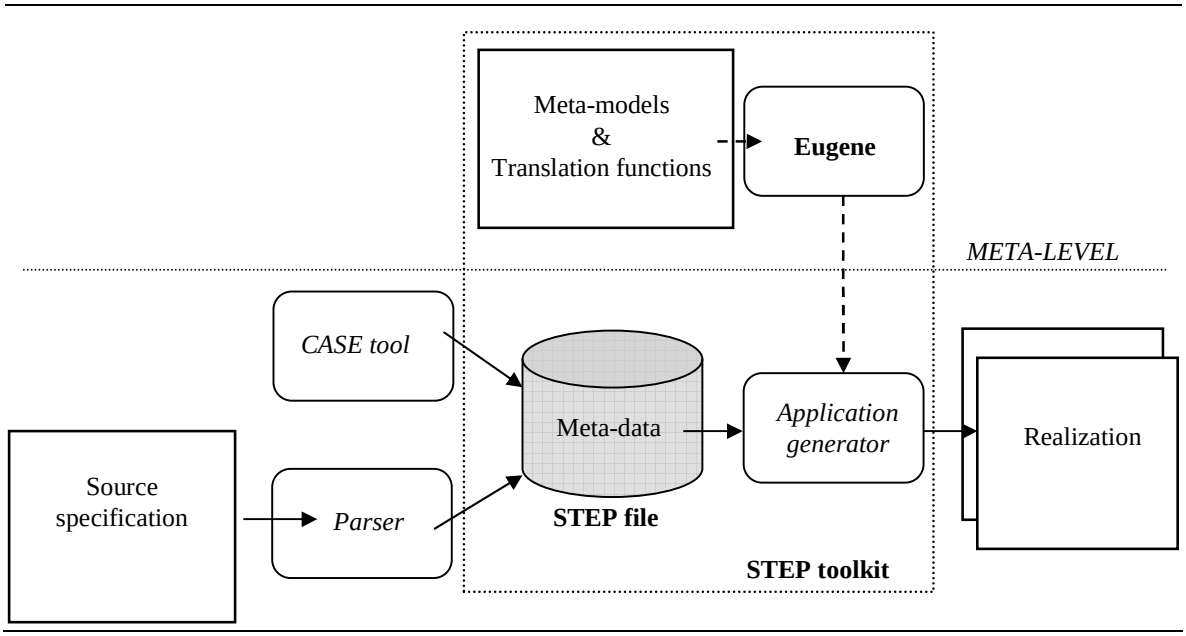

Figure 1 The building and the using of an application generator

Most application generators builders produce the generators while exploiting the definition of the syntax and the semantic of the source specification language. EUGENE differs from this way, because the generator is not produced from a grammar-like specification, but from the structure and the organization of the meta-data (globally referred to as the meta-models). Thus the tool which feeds the meta-models with meta-data can be a classical parser (generally automatically built in other environments than EUGENE) but another tool not relying on a precise and formal concrete syntax.
Let us take the example of the classical work done by an interface generator between a programming language and a Relational Data Base Management System (RDBMS). The generator uses the SQL table definitions and produces data access functions embedding SQL.

2.1 The meta-models

The specification of the generator being built consists in several meta-models. A precise explanation of the nature of these meta-models can be found in [6]. To simplify the example, we will use one source meta-model.

The source language meta-model consists in a set of EXPRESS schemata that describes the source language data constructs. The main components of a source language meta-model are types and entities, describing concepts that can be used with the source language. Entities provide buckets to store meta-data while global and local EXPRESS constraints are used to ensure meta-data soundness.

Considering the classical example of building data access functions from SQL table definitions, the source language is SQL and figure 2 shows a simplified SQL meta-model. The table definition is related to a list of columns.

```
SCHEMA sql_dictionary;

 ENTITY simple_type ABSTRACT SUPERTYPE OF
          (ONEOF(real_type, integer_type, string_type));
 END_ENTITY;

 ENTITY table;
    name : STRING;
    columns : LIST [1:?] OF column; ...
 END_ENTITY;

 ENTITY column;
    name : STRING;
    domain : simple_type;
 END_ENTITY;

END_SCHEMA ;
```

Figure 2 An example of source meta-model: a simple SQL dictionary

2.2 The translation functions

The translation functions are written in EXPRESS and are specified in the translation schema. The specification of translation functions is a programming activity in which EXPRESS is used as an imperative language. A typical translation function returns a string and is parameterized with types that are entities defined in the source language meta-model. The resulting string represents part of the target textual representation. Because of the nature of parameter types, this activity is often called meta-programming [1, 5].

Figure 3 shows two skeletons of translation functions related to the paper example. *column_to_field* and *table_to_function* can be used in order to produce code for a column and a table.

```
SCHEMA sql_traduction;
USE FROM sql_dictionary ;

FUNCTION column_to_field ( c : column) : STRING ;
LOCAL
      result : STRING := '';
END_LOCAL ;
     (* build statements for the management of a
        column in a programming langage*)
      RETURN (result) ;
END_FUNCTION ;

FUNCTION table_to _function( t : table ) : STRING ;
LOCAL
      result : STRING := '';
END_LOCAL ;
      (* build statements to read/write a table *)
      REPEAT no := 1 TO HIINDEX(t.columns) ;
       ....
      END_REPEAT ;
      RETURN (result) ;
END_FUNCTION ;

END_SCHEMA ;
```

Figure 3 An example of translation functions

3 Experiments

EUGENE has been experimented at Brest University with a lot of research projects described in [7].

At Syseca Brest, EUGENE is currently used for an industrial project : the management of the textile department of a supermarket chain. This application uses ORACLE and is implemented with Visual Basic (VB).
Expected benefits for the use of EUGENE were the software quality of generated code and the productivity gain (initially estimated at 10%).

Four generators have been built :

- *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.

The table below shows the ratio of input specification to generated code and the main advantage of each generator.

	ratio	main advantages
gari_iobdd	1/25	leverage effect, reliability
gari_prc	1/3	code homogeneity
gari_sql	1/3	reliability
gari_taico	1/30	leverage effect, code complexity

Using these generators, we delivered the first subsystem including 17000 generated lines. Comparing with another hand-made subsystem (for the same project) these 17,000 lines represent about 60 man-day work. The four generators have been built in 30 days. This demonstrates that building specific generators can be profitable even for a single project.

4 To use or not to use EUGENE

4.1 Non-relevant application domains

Most application generator builders first focus on the formal definition of the concrete and abstract syntax of the source specification. From these definitions, an internal representation structure and specialized tools, such as structured editors and parsers, are automatically derived. Stage [3], Centaur [2] or the meta-environment described in [4] are examples of such application generator builders. In these tools, the automatic derivation from a syntax definition to an internal representation structure is enabled because the langage used to specify the syntax is itself defined in an orthogonal way, i.e. there is only one possibility to describe a given situation.

EXPRESS is not defined in an orthogonal way. It can not be used in order to describe concret or abstract syntaxes. When the source specification can be described with a grammar, the lack of formal syntax definition denies EUGENE the capacity to generate parsers automatically. Hence, EUGENE is not suited to build syntax-directed translators such as compilers or structured editors.

Using EUGENE essentially requires data design and imperative programming. EUGENE is not adapted to build generators which use non-imperative features such as inference or deductive capabilities.

4.2 Relevant situations

The benefits of using EUGENE will be apparent in many different circumstances :

- the same language, EXPRESS, is used overall during the building process; this facilitates learning and the use of the environment,
- a generator is designed in (relative) independence of the system and of the target language in which the generator should be implemented; this facilitates the diffusion and the portability of the generator,
- meta-schemata can be re-used as they are, either from one generator to another, or from another project which needed some meta-modeling (especially from standardized STEP constructs).
- the source specification does not have to be a textual language described by a grammar; this allows the building of a generator dealing with irregular cases and exceptions, provided that it can be expressed explicitly in the meta-schemata or by the meta-data.
- as instances of EXPRESS schemata, meta-data can be exchanged by means of the STEP neutral exchange structure; it provides inter-operability between application generators as well as between an application generator and another CASE tool.

From its very nature, EXPRESS can be used as a substitute source specification language. EXPRESS is a very powerful modeling language. Because the EXPRESS parser belongs to the EUGENE environment, it can be re-used to produce meta-data. Specifications using models of other methods (like E-R or UML) can be manually translated into EXPRESS, and it is possible to avoid the development of a source specification parser. In this case, the source language used manually acts as a specification language and EXPRESS is used as a design language while the transformation from specification to design is done manually.

5 Conclusion

This paper has presented the EUGENE environment, a STEP-based application generator builder. A generator is automatically built from the specification of the intermediate representation manipulated by the generator (the meta-schema) and from the specification of translation functions.

EUGENE is not suitable to build syntax-directed translators, such as compilers, interpreters or structured editors. Because the lack of a formal meta-language to define the grammar of the source language, EUGENE is unable to produce software components automatically, namely parsers dealing with the specification source.

Different generators are briefly described. It demonstrates the power of the approach for relatively straightforward applications. This result meets the justification of the approach because simple solutions can be implemented in a simple and profitable way with EUGENE and therefore applicable to a software company.

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] P. Borras, D. Clément, Th. Despeyroux, J. Incerpi, G. Kahn, B. Lang, and V. Pascual. CENTAUR: the system. In ACM SIGSOFT'88, Third annual symposium on software development environment, 1988.

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

[4] Paul Klint. A Meta-Environment for Generating Programming Environments. In ACM Transaction on Software Engineering and Methodology, volume 2, 1993.

[5] David A. Ladd and J. Christopher Ramming. A*: A Language for Implementing Language Processors. IEEE Transactions on Software Engineering, 21(11), November 1995.

[6] Alain Plantec and Vincent Ribaud. EUGENE: a STEP-based framework to build Application Generators. AWCSET'98, CSIRO-Macquarie University, 1998.

[7] Alain Plantec and Vincent Ribaud. Using and re-using application generators. COSET'99, CSIRO-Macquarie University, 1999.