STEP-based CASE Tools cooperation

Alain Plantec, Vincent Ribaud

STEP-based CASE Tools cooperation

Alain Plantec [1] and Vincent Ribaud [2]

[1] *SYSECA, 34 quai de la Douane, 29285 Brest Cedex, France,*
alain.plantec@syseca.thomson-csf.com
[2] *EA2215-LIBr, Faculté des Sciences, BP 809, 29285 Brest Cedex, France,*
ribaud@univ-brest.fr

Abstract

Computer-Aided Software Engineering (CASE) tools need to cooperate and this can be accomplished by exchanging or sharing meta-data stored in a repository.

STEP is an ISO 10303 standard developed to facilitate product information sharing by specifying sufficient semantic content for data and their usage. STEP is providing a dedicated technology, mainly an object oriented modeling language *EXPRESS* and a standardized data access interface *SDAI*.

Meta-modeling the repository in EXPRESS allows a facilitated cooperation. Both exchange and sharing are provided by the SDAI generated from the EXPRESS meta-schema. Some experiments are related and an industrial project is depicted. Designer/2000 modeling is jointly used with dedicated Visual Basic code generators. Consistency is needed between these two tools families. This is achieved with a simple tool, but the use of the experimental method proposed is still difficult. Impedance mismatch between relational and object database paradigms may be the origin of the difficulties.

Keywords : CASE tools interoperability, CASE tools implementation, STEP standard, SDAI, EXPRESS

Introduction

CASE tools assist system development in managing system documentation. Documentation is structured with the help of various models, elaborated throughout the system development cycle. Information on the different models are the data (in fact meta-data) processed by the CASE tools. Cooperation of CASE tools rely on common meta-data access. This kind of cooperation is described as a data integration in [12].

CDIF (*CASE Data Interchange Format*) [3] and IRDS (*Information Resource Dictionary System*) [6] are two examples of proposals intended to facilitate the cooperation of CASE tools and the exchange of models between the vendor's tools.

In early 90's, CDIF and IRDS are the major representatives of the two approaches used to (meta-)data integration : exchange of meta-data files or sharing through a common repository. These approaches are still valid today, although the technology slightly differs (e.g. use of marked-up language such as XMI or dedicated API).

One major component of a CASE tool is the repository. A repository holds the system documentation in a central place online. Various tools pick information in the repository, process them and store the results in the repository. The structure of data in the repository is often referred as the *meta-model*. The repository itself is usually implemented using either a relational or an object-oriented database management system.

STEP is an ISO 10303 standard developed to facilitate product information sharing by specifying sufficient semantic content for data and their usage. Parts of ISO 10303 are intended to standardize conceptual structures of information which are either generic or within a subject area (e.g. mechanics). Standardized parts are expressed with a dedicated technology, mainly an object-oriented modeling language called EXPRESS and a standard data access interface called SDAI.

As mentioned in the STEP box, the SDAI is a functional interface for EXPRESS-modeled database and is independent of any particular system and language. The SDAI allows data sharing as well as data exchange. The key point is that a SDAI is automatically generated from the EXPRESS schema of the database (as long as an SDAI generator has been made for the target database management 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.
The constraints expressed in an entity definition can be of four kinds: **(1)** the *unique* constraint allows entity attributes to be constrained to be unique either solely or jointly, **(2)** the *derive* clause is used to represent computed attributes, **(3)** the *where* clause of an entity constraints each instance of an entity individually and **(4)** the *inverse* clause is used to specify the inverse cardinality constraints. 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.

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.

This paper argues that given a CASE tool, data interoperability can be accomplished through an SDAI generated from the EXPRESS schema resulting from the meta-model used in the CASE tool. Benefits of this method include data exchange as well as data sharing, allowing system developers to use best suited CASE tools to their projects, even if they belong to different CASE toolsets. However, complex repository causes a complex meta-modeling and the resulting SDAI can be difficult to use.

The paper is organized as follows: an example of different data integration is described in section 1. Section 2 shows how different CASE tool were needed and used in a commercial system. Then we finish with perspectives and a conclusion.

1 Examples of data integration

1.1 UML

Within the context of a research project, colleagues were faced to use jointly two kinds of CASE tools: a UML tool and a SDL tool. The cooperation should be the following: an UML tool will be used to design class diagrams and collaboration diagrams. SDL code will be generated from both diagrams and then imported into the SDL tool.

Within another research project, a colleague wished to use UML to design class diagrams and then generate a SmallTalk-80 implementation. Unfortunately, he didn't find any UML tool able to generate SmallTalk-80 code.

We started two different projects of two persons within the context of final-year course-work (bachelor students). We chose Argo/UML from Jason Elliot Robbins [10] for its open-implementation and its conformity to the UML Meta-model 1.1 [1]. Moreover, Argo/UML allows the two types of data integration mentioned above : a set of Java classes providing an API (*Application Programming Interface)* to the meta-data as well as a file exchange format (*.xmi*).

Meta-programming with an API For the cooperation between Argo/UML and SDT [11], a SDL tool, meta-programming with ARGO API was chosen. Argo/UML does not use a database management system to store information about diagrams. Hence in order to share meta-data with the class and collaboration diagrammers, students [4] incorporated a SDL generator in Argo/UML. This generator was written in Java.

Part of the time devoted to the project has been used to understand the UML meta-model (available only in a

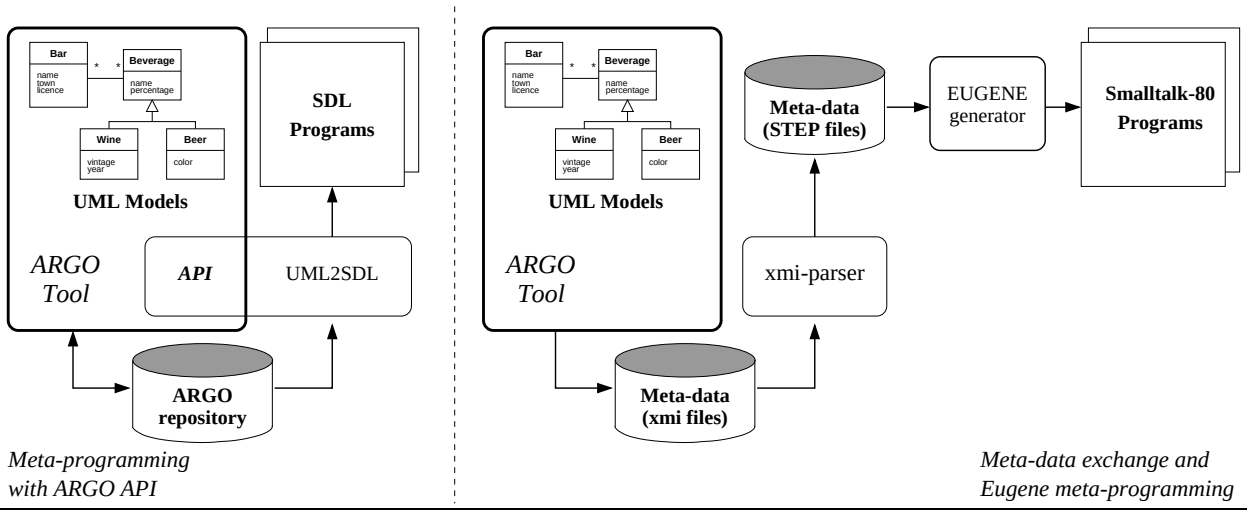

Figure 1. Meta-programming with ARGO API *versus* meta-data Exchange and Eugene meta-programming

graphical UML form) and to learn the use of the API (formed by a total of 120 classes) and the way it matches the meta-model. Then the students were able to use the API to write their SDL generator.

Meta-data exchange In the second project, meta-data exchange between Argo/UML and the generator was the solution we kept. The generator was built with *Eugene*, our STEP-based application generators builder [9]. *Eugene* is used within the context of research projects at Brest University and also in industrial projects in Syseca, a software company.

Like the first project, part of the time was devoted to UML meta-model understanding. Building an application generator with Eugene requires an EXPRESS description of the meta-model of the generator inputs (here a *.xmi* file) and students did it [5]. Then a meta-program was written in order to generate Smalltalk-80 code from meta-data.

Discussion We cannot compare the time devoted to real development in each project. The SDL generator was written without any meta-environment whereas the SmallTalk-80 generators uses that type of environmment. But there were two successive phases in both projects, i.e. learning the system (API or meta-model) and programming. Two points should be noted:

- Learning an API is an experimental task, and no learning method can be provided. Consistency in the naming of elements and operations in the API helps to make learning and use more efficient.

  The use of Eugene implies writing of a schema of the meta-model. The learning phase is in fact a meta-modeling phase. This activity helps the students in the learning of UML meta-model.

- Programming an API depends on the API itself. Little experience can be re-invested in another API.

  Meta-programming is based on the meta-modeling phase, and another project will require another meta-modeling activity. So some meta-modeling experience will grow from a project to another.

1.2 STEP use

Cooperating with a CASE tool is made easier if the CASE tool provides an access to meta-data (API, meta-data files or others formated outputs). Experience gained from the above projects enables us to provide a method (supported by a tool, an SDAI generator) to write a CASE tool intended to cooperate with an existing CASE tool (see fig. 2):

Meta-modeling The structure of the existing (source) CASE tool repository is modeled with EXPRESS schemata.

SDAI generation An SDAI for the management system running the new CASE tool (called the *target system* below) is generated. This requires naturally an SDAI generator suited to the target system, but such an SDAI generator is re-used for each CASE tool available within this target system.

The SDAI is useful for each source CASE-tools: meta-data produced by a source CASE-tool are imported into the new CASE tool. For such a task, a specific program (i.e. a program parser or a meta-data converter) is implemented. An SDAI can be generated within the source system and used for this implementation.

CASE tool development The development of the new CASE tool is based on the SDAI, which provides a standard access to the meta-data exported from the

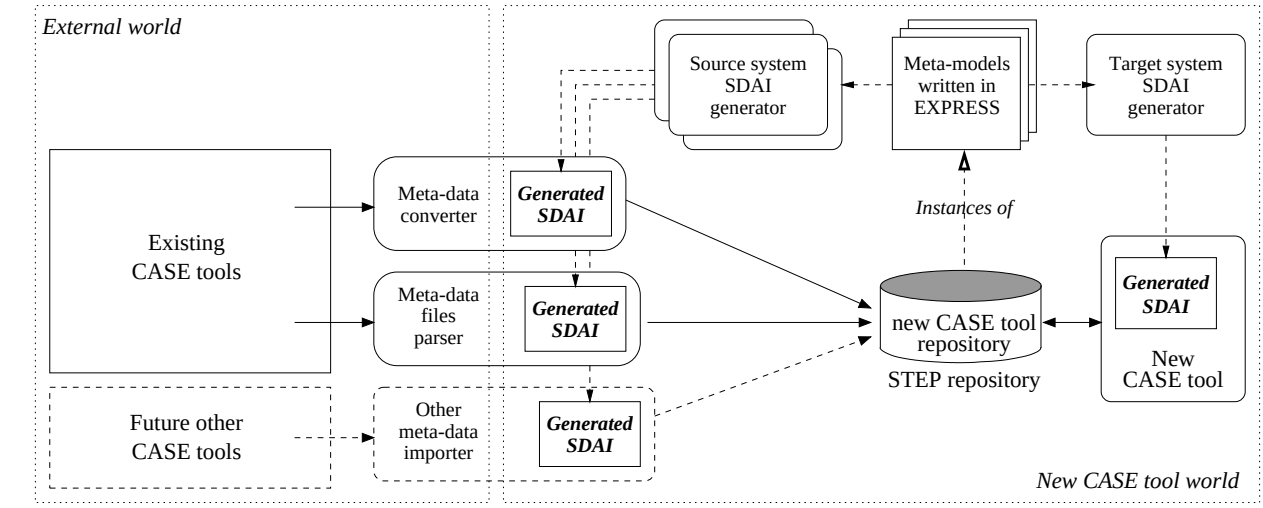

Figure 2. Using generated SDAI to interoperate with a given CASE tool

existing CASE tool and managed by a STEP repository.

2 Working with different CASE tools

2.1 The context

At Syseca Brest, a small team (3-6 persons) has been developing new software within a global 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 Oracle7 (now Oracle8) 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). Eugene is the environment used to build the generators. Input to these generators are either SQL select statements or EXPRESS schemata hand-made from Designer/2000 information.

2.2 Designer/2000

Oracle Designer/2000 is a suite of software toolsets for designing Windows-based client/server applications that interact with an Oracle database. Designer/2000 incorporates support for busines process modeling, system analysis, software design and code generation [8]. Designer/2000 provides a multi-user repository implemented using Oracle's RDBMS. The repository consists of tables that store information on the system we are analysing, designing and producing. A good introduction to Designer/2000 software toolsets and also a software development method using these tools can be found in [2]. 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).

2.3 Visual Basic Generators

The GARI family is used throughout the projects. Some generators use EXPRESS schema as inputs and still produce VB code. These schemata need to be hand-written from the meta-data of the repository. They may include entities and their attributes or tables and their columns, all of which have individual properties useful for the generators. The re-writing in EXPRESS schema of the information still present in the Designer/2000 repository is a tedious task, prone to errors and requiring repeated efforts to maintain the mapping between Designer/2000 information and VB code generated.

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.

2.4 Possible solutions

2.4.1 Generating EXPRESS from repository data

A first solution will be making a translation tool able to produce EXPRESS schema from the repository meta-data. The seamless integration is obtained through three steps (see our current implementation depicted in figure 3): analysis and design using Designer/2000 tools, generation of the EXPRESS schema, generation of the VB code with the GARI family.

Pros and cons It took three weeks to make the above translation tool (called *Malam*) [7]. 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.

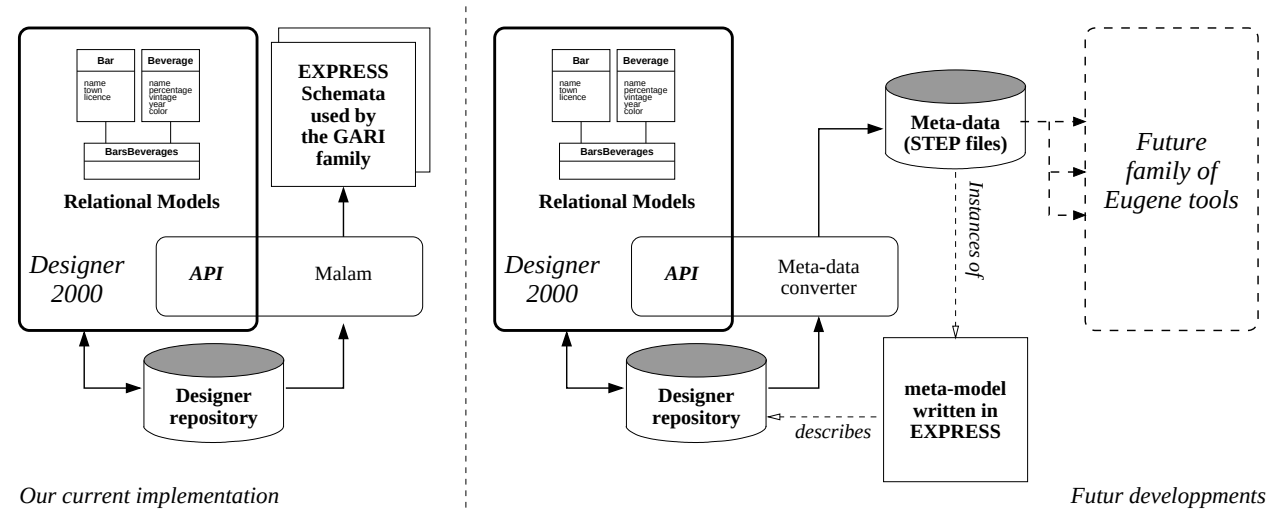

Figure 3. Cooperation between Designer/2000 and GARI with an intermediate translation tool

We are working essentially on table definition which are easy to translate.

2.4.2 Meta-modeling the repository structure in EXPRESS

As stated in conclusion of section 1, cooperating with Designer/2000 will be made easier by an SDAI operating on the repository, this SDAI being generated from a meta-model of the repository.

Since the repository is a standard SQL database, the translation tool above depicted in 2.4.1 can be used to produce automatically an EXPRESS schema of the repository structure. We did so but we now need to refine the schema. As a matter of fact, the repository consists of a relatively small number of tables that store the 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 objetcs created through the toolsets [2]. Unfortunately, if translating automatically SQL DDL statements is straightforward, this is not true with SQL DML statement, specially if they are complex.

Pros and cons The generated schema contains more than 5000 EXPRESS statements. Generating an SDAI for this schema provides a complex API, usable in many situations. Until now, we haven't built new tools that will use this SDAI. Intuitively, we expect that using this SDAI will be so complicated that it requires company investments.

3 Perspectives

Perspectives depends on the quality and the readibility of the Designer/2000 repository meta-modeling. Commercial tools often change but our experience with Oracle CASE tools indicates that the repository (formerly named Case*Dictionary in previous version of Oracle CASE) is stable, at least for the analysis and design phase. So, we are pursuing our efforts in repository understanding and meta-modeling refinements. The data-flow between Designer/2000 and the futur family of tools is depicted in figure3.

The difference of paradigm between a relational database (the repository) and an object-oriented schemata causes some problems, which may not be solved automatically.

4 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. However, when the repository structure is complex, following this approach requires investments. In fact it depends on the quality of the meta-model. Hence in some situations, dedicated translation tools using the repository API are easier to develop.

References

[1] UML metamodel 1.1. Technical report, Object Management Group, 1997.

[2] Paul Dorsey and Peter Koletzke. *Designer/2000 Handbook*. McGraw-Hill, 1998.

[3] EIA. *CDIF - Framework for Modeling and Extensibility*, 1994.

[4] Divi Lainé et Armelle Prigent. Modélisation UML et SDL dans le développement des systèmes temps-réel. Technical report, Université de Bretagne Occidentale, 1999.

[5] Céline Courbalay et Jean-Marc Douarinou. Traducteur d'UML vers SmallTalk-80. Technical report, Université de Bretagne Occidentale, 1999.

[6] ISO/IEC 10027. *Information technology - Information Resource Dictionary System (IRDS) framework*, 1990.

[7] Mikael Le Moal. Intégration d'oracle designer dans smalltalk-80. Technical report, Université de Bretagne Occidentale, 1999.

[8] Oracle. *Oracle Designer/2000 A Guide to Repository Administration*, 1995.

[9] 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.

[10] Jason Elliot Robbins. Argo/UML. http://www.ics.uci.edu/pub/c2/uml/index.html.

[11] Telelogic. SDT. http://www.telelogic.com.

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