



Forward Engineering and UML: From UML Static Models to Eiffel Code

Liliana Favre*, Liliana Martinez and Claudia Pereira

INTIA, Facultad de Ciencias Exactas, Universidad Nacional del Centro de la Provincia de Buenos Aires, Argentina

* CIC (Comision de Investigaciones Cientificas de la Provincia de Buenos Aires, {lfavre, lmartine, cpereira}@exa.unicen.edu.ar

ABSTRACT

In a previous research work we have proposed a rigorous process to forward engineering UML static models. This approach is based on the integration of semi-formal notations in UML, algebraic specifications and object-oriented code. The GSBL^{oo} language was defined to cope with concepts of UML models. The emphasis is given to the last steps in the road from UML to code: we describe how to transform GSBL^{oo} specifications into object-oriented code. Eiffel was the language chosen to demonstrate the feasibility of our approach. In particular, we analyze the transformation of different kinds of associations and the generation of Eiffel assertions.

INTRODUCTION

Unified Modeling Language (UML) has emerged as a standard modeling language in the object-oriented analysis and design world. It is a set of graphical and textual notations for specifying, visualizing and documenting object-oriented systems (OMG, 1999; Booch 1999).

There are CASE tools which offer code automatic generation and reverse engineering from object-oriented models. They present problems due to the lack of formal semantics of UML models and these models are semantically richer than the object-oriented languages, though.

These problems have motivated the analysis of different approaches to give semantics to the UML notations. The UML formalization is an open problem still and many research groups have already achieved the formalization of parts of the language. The Precise UML Group, pUML, is created in 1997 with the goal of giving precision to UML (Evans et al. 1998). Different results give semantics to UML subsets based on different formalisms (Lano 1995; Breu et al. 1997; Bruel and France, 1998; Gogolla and Ritchers, 1997; Kim and Carrington, 1999; Overgaard, 1998; Barbier et. al., 2001). Currently, there are few methods that include OCL, Catalysis is the most important reference (D'Souza and Wills, 1999). Other research works propose the integration of UML with OCL (Ritchers and Gogolla, 2000). Siau and Halpin (2001) and JDM (2000) identify some problematic aspects of UML and propose possible solutions.

A variety of advantages have been attributed to the use of formal software specifications to solve these problems. A formal specification can reveal gaps, ambiguities and inconsistencies. However, formal specifications alone do not address the need of industrial practitioners, who require an understandable and scalable semantics that can be integrated by using tools.

Favre and Clérici (2001) propose a rigorous process to forward engineering UML static models using the algebraic language GSBL^{oo}. The contribution was towards an embedding of the code generation within a rigorous process that facilitates reuse. The GSBL^{oo} language was designed to cope with concepts of the UML models. This language is relation-centric, it expresses different kinds of relations as primitives to develop specifications. The transformation of UML static models specified in OCL into GSBL^{oo} and a system of transformation rules were described. The formal model *SpRelm* for defining structured collections of reusable components that integrates algebraic specifications and object-oriented code was defined. The manipulation of *SpRelm* components by means of building operators is the basis for reusability.

The emphasis is given to the last steps in the road from UML to code. Eiffel was chosen to prove the feasibility of our approach. It is based on the "Design by Contract" principle. Contracts imply obligations and benefits for clients and contractors, and are made explicit by the use of Eiffel assertions. These features facilitate to integrate axioms of specifications with object oriented code. We describe how to

transform GSBL^{oo} specification into Eiffel, analyzing the transformation of different kinds of relations and the generation of Eiffel assertions.

This paper has the following structure: -Section 2 describes the basis of a rigorous forward engineering method, -Section 3 outlines GSBL^{oo} language features, -Section 4 analyzes how to transform GSBL^{oo} specifications into Eiffel code, -Section 5 considers conclusions.

MOTIVATION

There are CASE tools for code generation starting from UML models. Unfortunately, the current techniques available in these tools are not enough for the complete automated generation of source code. As an example, Rational RoseTM will be analyzed (Quatrani, 1998). This allows generating databases definitions, class interfaces and relations in which the class participates. The current modeling languages available in Rational RoseTM (Booch, OMT, UML) do not have a precisely defined semantics. This hinders the code generation processes. Another source of problems in these processes is that, on one hand, the UML models contain information that can not be explicitated in object-oriented languages and on the other, the object-oriented languages express implementation characteristics that have no counterpart in the UML models. For instance, languages like C++, Java and Eiffel do not allow explicit associations, their cardinality and their constraints. It is the responsibility of the designer to make good use of this information, selecting from a limited repertoire of implementations or implementing the association by himself. The forward and reverse processes in Rational RoseTM are facilitated by means of the insertion of annotations in the generated code. These annotations are the link between the model elements and the language, they should be kept intact and not changed. It is the programmer's responsibility to know what he can modify and what he can not.

The programmer, to solve implementation problems, can modify the code by adding or removing classes, modifying class attributes or operations, changing operation signatures, etc. These code modifications require the programmer's ability to keep the integration between model and language.

Moreover, the existing tools do not exploit all the information contained in the UML models, for instance, cardinality and constraints of associations, preconditions, postconditions and class invariants in OCL are only translated as annotations. Assertions in OCL could be translated to assertions in an object-oriented language that supports them, like Eiffel. Furthermore, reuse is based on object-oriented language libraries and not on specifications that describe relations between classes and their operations free from implementation details.

To overcome these problems, a rigorous process to forward engineering UML static models using the algebraic language GSBL^{oo} was proposed (Favre and Clérici, 2001).

Starting from UML class diagrams, an incomplete algebraic specification can be automatically built by instantiating reusable schemes.

This specification contains the highest information obtained by translating the UML constructions and OCL constraints to GSBL⁰⁰.

Preconditions, postconditions and invariants in OCL will be translated to GSBL⁰⁰. The process of transformation is supported by a transformation rules system (Favre et. al, 2000).

Transformations are based on a reusable component library defined by the *SpReIm* model. (Favre and Clérice, 2001). This model allows defining structured collections of reusable components that integrate algebraic specifications and object-oriented code. It takes advantage of the algebraic formalism power to describe behavior in an abstract way integrating them with concrete implementations. It consists of three abstraction levels:

- *Specialization*: it describes a hierarchy of incomplete specifications related by specialization relationships through two views: one based on OCL specifications and the other on GSBL⁰⁰.
- *Realization*: it describes a hierarchy of complete specifications related by realization relationships.
- *Implementation*: it relates hierarchy of concrete classes schemes in an object-oriented language.

The reuse of components is based on the application of operators. Reuse operators were defined on the three levels of the *SpReIm* model.

Thus, an algebraic specification can be semi-automatically built. It can be used to detect inconsistencies in the class diagrams. The GSBL⁰⁰ specifications must be integrated with object-oriented code. The emphasis in this paper is given to the transformation of GSBL⁰⁰ to Eiffel.

FORMALIZING UML STATIC MODELS IN THE GSBL⁰⁰ LANGUAGE

The existing algebraic specification languages do not possess specific constructions to express all kinds of relations in UML (dependency, association, generalization and realization). These are generally buried in client and inheritance relations. However, associations are semantics constructions of equal weight to the classes and generalizations in the UML models, and should not be treated just as implementation constructions. In fact, the associations allow abstracting the interaction between classes in the design on large systems and they affect the partition of the systems in modules. Since an integrated method requires common structuring mechanisms for object-oriented models and algebraic specifications, the GSBL⁰⁰ language has been defined. It enriches GSBL (Clérice, 1988). The ability to specify deferred and effective parts, inheritance relations among classes and mechanisms such as implicit parameterization to support reuse of specifications and their incremental development are among the most important features of GSBL.

GSBL⁰⁰ extends GSBL with constructions that allow expressing different kinds of UML relations. OBJECT CLASS and ASSOCIATION class specify classes and associations (ordinary, qualified, association-class) respectively.

An OBJECT CLASS distinguishes incomplete and complete parts. The DEFERRED clause declares new sorts, operations or equations incompletely defined. The EFFECTIVE clause either declares new sorts, operations or equations completely defined, or completes the definition of some inherited sort or operation. Sorts and operations are declared in the SORTS and OPS clauses. In GSBL⁰⁰, it is possible to specify partial functions restricting operations by preconditions.

GSBL⁰⁰ expresses dependencies in the context of classes by means of the USES clause. The mechanisms for defining inheritance are the RESTRICT and REFINES clauses. The REFINES clause relies on the module viewpoint of classes and the RESTRICTS clause on the type viewpoint of a class. Both of them reflect IS-A relations between abstractions in the external model of any software system. Associations are defined as standard elements in GSBL⁰⁰.

Generic relations can be used in the definition of concrete relations by instantiation. New associations and whole-part relations (ag-

gregation and composition) can be defined by means of the following syntax:

```
ASSOCIATION <relationName>
IS <constructorTypeName>[...<Class1>;...<Class2>; ...<Role1>;...<Role2>; ...: mult1;...mult2;
...visibility1;...visibility2 ]
CONSTRAINED BY <constraintList>
END
```

```
WHOLE-PART <relationName>
IS <constructorTypeName> [...<Whole>;...<Part>; ...<Role1>;...<Role2>;...: mult1;...mult2;
...visibility1;...visibility2]
CONSTRAINED BY <constraintList>
END
```

The IS clause expresses the instantiation of <constructorTypeName> with classes, roles, visibility and multiplicity. The CONSTRAINED-BY clause allows the specification of static constraints in first order logic. Relations are defined in an Object Class by means of the following syntax:

```
OBJECT CLASS C...
"<relationName>"ASSOCIATES <className>
"<relationName>" HAS-A SHARED <className>
"<relationName>" HAS-A NON-SHARED <className>
...
END-CLASS
```

The keywords ASSOCIATES and HAS-A identify ordinary association or aggregation respectively. The keywords SHARED and NON-SHARED refer to simple aggregation and composition respectively. An association may be refined to have its own set of operations and properties. Such an association is called an Association Class.

The PACKAGE is the mechanism provided by GSBL⁰⁰ for grouping classes, and controls its visibility. It matches the UML semantics. Classes and their relations from a system design into a series of packages might be separated using the GSBL⁰⁰ import dependencies to control access among these packages.

Figure 1 shows an information system for a company. There is an association between *Person* and *Company*, specifying that managers (instances of *Person*) manage companies. Every manager may manage only one department and every company may have only one manager. There is a qualified association between *Person* and *Bank*. In the context of *Bank*, an *accountNumber* (qualifier) identifies a particular customer. In an *employer/employee* relation between *Company* and *Person*, there is a *Job* that represents the properties of that relation, which applies to exactly one pairing of *Person* and *Company*. Figure 2 partially shows the GSBL⁰⁰ specification of Figure 1.

Figure 1: Company information system. A UML class diagram.

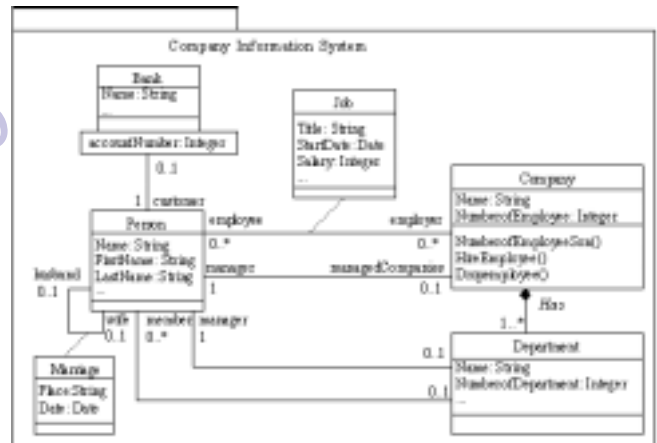


Figure 2: Information system company. A GSBL⁰⁰ specification

```

PACKAGE Information-System
OBJECT CLASS Person
USES Date, Integer, String, Sex, Boolean
«Company-Person» ASSOCIATES managedCompany:
    Company
«Manager-Department» ASSOCIATES department:
    Department
«Member-Department» ASSOCIATES department:
    Department
«Person-Bank» ASSOCIATES bank: Bank
«Job» ASSOCIATES employer: Company
«Marriage» ASSOCIATES person: Person
EFFECTIVE
SORT Person
OPS
Create_Person: Boolean x Boolean x Date x Integer x
String x String x Sex x Integer x ... -> Person
IsMarried: Person -> Boolean
IsUnEmployed: Person -> Boolean
BirthDate: Person -> Date
Age: Person -> Integer
FirstName: Person -> String
LastName: Person -> String
Sex?: Person -> Sex
NumeroSon: Person -> Integer
EQS {p: Person; d: Date; b1, b2: Boolean; s1, s2: String;
l: Integer; b-sex: Sex}
IsMarried(Create_Person(b1, b2, d, s1, s2, s-sex, l)) = b1
IsUnEmployed(Create_Person(b1, b2, d, s1, s2, s-sex, l)) = b2
BirthDate(Create_Person(b1, b2, d, s1, s2, s-sex, l)) = d
Age(Create_Person(b1, b2, d, s1, s2, s-sex, l)) = l
FirstName(Create_Person(b1, b2, d, s1, s2, s-sex, l)) = s1
LastName(Create_Person(b1, b2, d, s1, s2, s-sex, l)) = s2
Sex?(Create_Person(b1, b2, d, s1, s2, s-sex, l)) = 1-sex
NumeroSon(Create_Person(b1, b2, d, s1, s2, s-sex, l)) = l
END-CLASS

OBJECT CLASS Company
USES String, Integer
«Company-Person» ASSOCIATES manager: Person
«Has» HAS-A NON-SHARED department:
    Department
«Job» ASSOCIATES employee: Person
EFFECTIVE
SORT Company
OPS
Create_Company: String x Integer -> Company
Name: Company -> String
NumeroEmployee: Company -> Integer
HireEmployee: Company (c): Person (p) -> Company


```

 pre: not includes (getEmployee(Job, c), p)
 DropEmployee: Company (c): Person (p) -> Company
 pre: includes (getEmployee(Job, c), p)
 NumeroMarriedEmployee: Company -> Integer
 NumeroEmployeeSon: Company -> Integer
EQS {r: Name; s: String; c: Company; p: Person; l:
 Integer; ...}
Name(Create_Company(s, ...)) = s
NumeroEmployee(c) = size(getEmployee(Job, c))
NumeroEmployee (HireEmployee(c, p)) =
 NumeroEmployee(c) + 1
NumeroEmployee(DropEmployee (c, p)) =
 NumeroEmployee(c) - 1
getEmployee(Job, HireEmployee(c, p)) =
 including (getEmployee(Job, c), p)

```


```

TRANSFORMING GSBL⁰⁰ SPECIFICATIONS IN EIFFEL CODE

The resulting specifications of transforming UML static models in GSBL⁰⁰ must be integrated with an object-oriented code. These transformations are described below and they are exemplified for the classes and relationships expressed in the UML diagram in Figure 1.

Transformation of Classes

The algebraic specification obtained in the previous step must be transformed into Eiffel code. To achieve this, every clause and relation present in a GSBL⁰⁰ OBJECT CLASS specification was analyzed.

GSBL⁰⁰ and Eiffel have the same syntax for the declaration of

class parameters. Then, this transformation is reduced to a trivial translation.

The relation introduced in GSBL⁰⁰ using the clause USES will be translated into a client relation in Eiffel. The relation expressed through the keywords REFINES and RESTRICTS in GSBL⁰⁰ will become an inheritance relation in Eiffel. This provides mechanisms to carry out modifications on the inherited classes that will allow adapting them.

The DEFERRED and EFFECTIVE clauses in GSBL⁰⁰ declare sorts and operations of the class with the equations that define their behavior. If an OBJECT CLASS is incomplete, i.e., it contains sorts and operations in the clause DEFERRED, the keyword *class* in Eiffel is preceded by the keyword *deferred*. Sorts do not require explicit translation.

From the signature of the operations, the interfaces for the methods of the Eiffel class are generated (*feature* in Eiffel). The translation of each operation has a different treatment according to the type of feature to which it makes reference (functions, procedures, variables or constants). It should also be considered that of all the domains of an operation, the first one that coincides with the sort of the specified class is the object *Current* in the object-oriented language and it should be eliminated from the list of parameters of the resulting feature.

Functions and procedures can present arguments. Once each name is obtained, either by an explicit requirement to the user or by extracting it from the specification, the list of arguments of each feature is built. Functions and procedures require a body defined by the keywords *do end*, which will be completed by the user.

Transformation of Axioms

Eiffel provides an assertion language. Assertions are boolean expressions expressing semantics properties of the classes. They “serve to express the specification of software components: indications of *what* a component does rather than *how* it does it” (Meyer 97). They can play the following roles:

- Precondition: expresses the requirements the client must satisfy to call a routine.
- Postcondition: expresses the conditions the routine guarantees on return.
- Class invariant: expresses the requirements every object of the class must satisfy after its creation.

Preconditions and axioms of a function written in GSBL⁰⁰ are used to generate preconditions and postconditions for routines and invariants for Eiffel classes.

It is worth clarifying for the assertion generation that a basic functionality $f: s \times a_1 \times a_2 \times \dots \times a_n$, where s is the sort of interest, is translated into Eiffel syntax as $f(a_1, a_2, \dots, a_n)$.

A GSBL⁰⁰ precondition, which is a well-formed term defined over functions and constants of the global environment classes, is automatically translated to Eiffel method precondition.

Axioms in a formal specification language represent the constraints the class introduces on the operations. Axioms are translated to Eiffel postconditions and invariants.

The system can automatically derive an invariant if it can establish a correspondence between the functions in the axiom and the class features that only depend on the object state.

- Bruehl, J.; France, R. (1998) Transforming UML Models to Formal Specifications. In : Proc. of UML'98-Beyond the notation, Lecture Notes in Computer Science 1618, Springer Verlag, 78-92.
- Cléricali, S.; Orejas, F.(1988) GSBL: An Algebraic Specification Language Based on Inheritance. In: Proc. of the European Conference on Object-oriented Programming ECOOP 88, 78-92.
- D'Souza, D.; Wills, A.(1999) Objects, Components, and Frameworks with UML. Addison Wesley.
- Evans, A.; France, R.; Lano, K.; Rumpe, B. (1998) The UML as a Formal Modeling Notation. In: Proc. of UML'98-Beyond the Notation, Lecture Notes in Computer Science 1618. Springer.
- Favre, L.: Object-oriented Reuse through Algebraic Specifications In: Proc. of Technology of Object-Oriented Languages and Systems, TOOLS 28, IEEE Computer Society, 1998; S 101-112.
- Favre, L.; Martinez, L.; Pereira, C. (2000) Transforming UML Static Models to Object Oriented Code. Technology of Object-Oriented Languages and Systems, TOOLS 37, IEEE Computer Society. ISBN 0-765-0918-5 (170-181).
- Favre, L.; Cléricali, S.(2001) A Systematic Approach to Transform UML static Models to Code. In: (K.Siau and T. Halpin), Chapter II. Unified Modeling Language: System Analysis, Design and Development Issues, Idea Group Publishing, USA.
- Gogolla, M.; Richters, M.(1997) On combining Semi-formal and Formal Object Specification Techniques. In: Proc. WADT97, Lecture Notes in Computer Science 1376, Springer, 238-252.
- JDM (2000) Journal of Database Management, 11(4) Idea-Group Publishing
- Kim, S.; Carrington, D.(1999) Formalizing the UML Class Diagram using Object-Z. In: Proc. UML 99, Lecture Notes in Computer Science 1723, 83-98.
- Lano, K (1995) Formal Object-Oriented Development. Springer-Verlag.
- Meyer, B (1997) Object-oriented Software Construction. Prentice Hall.
- OMG (1999) Unified Modeling Language Specification, v. 1.3. document ad/99-06-08, Object Management Group.
- Overgaard, G.(1998) A Formal Approach to Relationships in the Unified Modeling Language. In: Proc. of Workshop on Precise Semantic of Modeling Notations, International Conference on Software Engineering. ICSE'98, Japan.
- Quatrani, T. (1998) Visual Modeling with Rational Rose and UML. Addison Wesley.
- Richters, M.; Gogolla, M.(2000) Validating UML Models and OCL Constraints. In: (Evans, A. ;Kent,S.) Proc. of <<UML>> 2000. The Unified Modeling Language, Lecture Notes in Computer Science 1939. Springer, 265-277.
- Siau, K.; Halpin, T. (2001) Unified Modeling Language: System Analysis, design and Development Issues. K. Siau; T. Halpin (eds) Idea-Group Publishing.

0 more pages are available in the full version of this document, which may be purchased using the "Add to Cart" button on the publisher's webpage:
www.igi-global.com/proceeding-paper/forward-engineering-uml/31852

Related Content

Application and Research of Interactive Design in the Creative Expression Process of Public Space

Yuelan Xu (2022). *International Journal of Information Technologies and Systems Approach* (pp. 1-13).
www.irma-international.org/article/application-and-research-of-interactive-design-in-the-creative-expression-process-of-public-space/307028

Estimation and Convergence Analysis of Traffic Structure Efficiency Based on an Undesirable Epsilon-Based Measure Model

Xudong Cao, Chenchen Chen, Lejia Zhang and Li Pan (2024). *International Journal of Information Technologies and Systems Approach* (pp. 1-25).
www.irma-international.org/article/estimation-and-convergence-analysis-of-traffic-structure-efficiency-based-on-an-undesirable-epsilon-based-measure-model/332798

Arsenic Removal from Drinking Water Using Carbon Nanotubes

Kausar Jahan, Kenneth Sears, Jaimie Reiff, Sarah Dores, Paulina Kruszewski and Shawn Williams (2015). *Encyclopedia of Information Science and Technology, Third Edition* (pp. 2908-2916).
www.irma-international.org/chapter/arsenic-removal-from-drinking-water-using-carbon-nanotubes/112714

Interpretable Image Recognition Models for Big Data With Prototypes and Uncertainty

Jingqi Wang (2023). *International Journal of Information Technologies and Systems Approach* (pp. 1-15).
www.irma-international.org/article/interpretable-image-recognition-models-for-big-data-with-prototypes-and-uncertainty/318122

Critical Success Factors in E-Democracy Implementation

Aderonke A. Oni and Adekunle O. Okunoye (2018). *Encyclopedia of Information Science and Technology, Fourth Edition* (pp. 3561-3568).
www.irma-international.org/chapter/critical-success-factors-in-e-democracy-implementation/184066