Automatic object-oriented visual programming with *OO-METHOD*

Jose Romero, Pedro J .Molina, Oscar Pastor
Department of Computer Systems and Languages
Polytechnic of Valencia
Camino de Vera s/n
Valencia 46071. SPAIN

Abstract: -Following the steps of other related works published in our research group, this paper deals with the problems of creating efficient automatic code generators from an object-oriented conceptual model.

Our own methodology called "OO-Method" is based not only on the object-oriented paradigm but also on the automatic programming paradigm. This method combines the advantages of formal specification systems with the practice provided by conventional object-oriented methodologies. Firstly, the OASIS formal object-oriented specification language is introduced to the reader through a brief review of its main concepts. Secondly, a description of OO-Method, in its current state, is presented enhancing some new interesting design decisions added on. Finally, a description of the CASE tool, which effectively supports the method, is given.

Finally, the results of our latest research and practice are discussed. In that way, we highlight the good properties of our method that make it adaptable and suitable for a world where technology is in a continually state of change. IMACS/IEEE CSCC'99 Proceedings, Pages:6541-6549

Key-Words :- Object-oriented paradigm, Object-oriented software production methodologies, formal specification languages, CASE tools, automatic code generation, visual programming, Windows environments, three-tier architectures.

1. Introduction

After some years of work, our research group created OASIS [5], a formal an object-oriented (OO) specification language following a similar approach to other formal languages like Troll [4]. At that point we realized the convenience of a method having OASIS as its formal high-level data dictionary and remaining faithful to the automatic programming paradigm [1]. It was called OO-Method, and later on, adopted the standard notation from the Unified Modeling Language [12] for its diagrams. Once the method was established, the decisions of implementing a CASE tool that support it was taken. Nowadays, the OO-Method/CASE Tool [6] is becoming a successful reality putting into practice the methodology through an agreement with a software company. Due to industrial interests, the CASE tool was adapted to generate code in imperative environments, even though it can also generate logic programs equivalent to an information system modeled with the tool. Focusing on imperative environments, the code generated will be functionally equivalent to the requirements modeled following the execution model [7] steps. This code will take profit of the existing rapid application development environments and new Internet three-tier architectures.

Summing up, the OO-Method/CASE takes profit of the UML standard notation and puts it over a formal and well-defined framework generating complete executable code (not only templates for classes) for visual programming environments.

At the end of this paper, we will summarize the future work related with OO-Method in an academic context and industrial practice alike.

2. OASIS

As OASIS is a formal object-oriented specification language, it will provide the proper way to describe information systems due to its well-founded semantics. An OASIS specification can be considered both as a high-level data dictionary and a formal modeling documentation. In fact, it constitutes the basis of the repository for our methodological proposal OO-Method and its corresponding OO-Method/CASE Tool implementation. Therefore, it is of a crucial importance to revise the OASIS concepts because they determine the elements that can be used for modeling a specific problem domain.

In order to clarify the concepts, we will introduce an

example of a car rental company. A contract relates a client with an automobile, which has a specific rate. The organization needs to store information about some special clients that are considered Very Important Persons.

Class and object in OASIS

The object classification process of a system is considered under an ontological perspective. Objects represent entities that compose the Universe of Discourse [11].

A class provides an intensional definition including the common properties of the objects that constitute the class population (extensional definition). It has static and dynamic properties.

Static properties – constant and variable attributes - represent the object state at any time. Aside from the OID system assignation is the set of constant attributes, which compose the class identification function.

Dynamic properties – events – will change the object state only if their associated preconditions and the class restrictions (static and dynamic) take place.

Services are offered by classes to the system. They could be both event and transactions. Transactions may be local or global. They represent event sequences with an "all-or-nothing" policy and not intermediate state observability.

Events must modify attribute values in our model. More precisely, each one modifies a predefined attributes set through an evaluation formula. These dynamic formulas for attribute evaluation are of the form $\psi[i:e]\psi'$, where ψ is a formula that is true in a given state and every execution of the event e by an actor i leads to a situation where ψ' is true. In addition, three types of formulas ψ' are distinguished, determining three categories of variable attributes. The first category is defined when ψ' increases or decreases the value of the attribute in a given amount, this is called a *push-pop* attribute. In the second case, if ψ' assigns a value that is independent of the values it had previously, it is called a *state-independent* attribute. Finally, if ψ' assigns a value in a discrete domain we will consider it as a *discrete-domain valued* attribute.

Summarizing, an object in OASIS can be defined as an observable process. The corresponding process specification in a class allows us to specify object dynamics and determines the access relationships between the states of instances. Processes are constructed by using those services commented before.

In our example, the class Client will be expressed as follows:

```
class Client
identification by_identification0 :  (client_code)  ;
constant_attributes
        client_code : Nat ;
        client_name : String ;
        client_type : String ;
variable_attributes
        phone : String ;
private_events
 var   nphone: String;  end_var
        create_client () new;
        destroy_client () destroy;
        change_phone (nphone);
```

Inheritance

Inheritance is the OASIS concept to support the "is-a" relationship, also present in traditional semantic data models. We considered it as a standard way to share code and behavior.

Inheritance is represented in OASIS by means of two class operators:

- Specialization: deals with derivation of children classes from parent classes. It can be *permanent* when a specialization condition specified on the constant class attributes is supplied or *temporal*, if creation and destruction events, or a specialization condition specified on the variable class attributes, determines the role creation or destruction. In our example, the Very Important Persons are defined as:

```
complex class VIPS specialization of Client where
client_type="VIP"
```

- Generalization: inverse of the previous one. Deals with ascendant inheritance or generation of parent classes collecting common properties of predefined classes. In a *disjoint* generalization, an instance of a generalized class will be an instance of one and only one of its descendent classes. We assume a *no disjoint* option by default.

OASIS also supports *multiple inheritance* by providing a parent classes list.

Aggregation

OASIS aggregation represents the "part-of" relationship where a complex object has other objects as components, structured as an object hierarchy. It categorizes the different existent types of aggregations according to the following dimensions:

- Inclusive/referential (also called relational)
- Null/not null
- Univalued/multivalued
- Flexible/strict
- Disjoint/no disjoint
- Static/dynamic

The *inclusive aggregation* verifies that components cannot exist out of the composite object. The *referential aggregation* differs from the inclusive in the fact that we only denote an existing relationship among objects of the involved classes, without having a complete inclusion of the component objects into the composite one.

Following the Entity-Relationship approach, each aggregation relationship has maximum and minimum cardinalities. So, we can know how many components and composite objects are related and vice versa. As we will see these cardinalities fix up a variety of design dimension:

- From container to component:
  Minimum cardinality values:
  0 = *Null* y 1= *Not Null*
  Maximum cardinality values:
  1= *Univalued* y M= *Multivalued*

- From component to container:
  Minimum cardinality values:
  0 = *Flexible* y 1= *Strict*
  Maximum cardinality values:
  1= *Disjoint* y M= *No Disjoint*

If a composite object has a fix composition once created we will have a *static* aggregation, else we will deal with a *dynamic* aggregation.

Looking at the contract class in our example we have:

complex class Contract aggregation of
Automobile(relational,static,univalued, disjoint,flexible,not null),
Client(relational,static,univalued, nodisjoint,flexible,not null)

Shared Event

This concept represents in OASIS a synchronous communication mechanism between different objects. It must be declared in all classes that shared a specific event. For instance, the rent and return vehicle events are declared in the Client class as follows (in the Automobile class they are defined in a similar way):

shared_events
var nmiles_arrival,nfuel:Nat; place_return:String;
end_var
rent_vehicle () with Automobile;
return_vehicle (nmiles_arrival, nfuel,nplace_return) with Automobile;

Triggers

They are services of a given class which the system activates in an automated way when a condition is satisfied by an object of the same or another class. So, this OASIS concept introduces internal system activity as condition-action rules. The system designer may assure the correct trigger confluence and termination. In our example if a car has less than one liter of gasoline, it will be automatically disabled for renting.

triggers
Self :: disable () if fuel<1;

Global Interactions

They are transactions involving services provided by different objects of the same or distinct classes. With these global interactions, interobjectual transactions can be declared. Formally, they can be seen as a local service of the aggregation among the involved classes. It provides an interclass communication mechanism. In the following example we illustrate a transaction where a new car is purchased and other one is sold in order to revamp the fleet.

global_transactions
CHANGE_CAR=**buy_vehicle**(p_agrRate,p_atrauto mobile_code,p_atrfuel,p_atrdelivering_date,p_atrloc alization,p_atrmodel,p_atrplate_number,p_atrselling _price).**sell_vehicle**(p_thisVehicle,ndate,namount);

In the next section we will see how combining these OASIS concepts with the Unified Modeling Language (UML) standard notation, we can specify an information system conceptual modeling using OO-Method.

3. OO-Method

OO-Method is an object-oriented software production method that encompasses two main components or phases: the conceptual model and the execution model. Nowadays, it is fully supported by a CASE tool developed as a research project between our department at the university and a software company.

In this context, we start with an Analysis step where three models are generated: the Object Model, the Dynamic Model and the Functional Model. They describe the *Object Society* from three complementary points of view within a well-defined OO framework. For these models we have preserved the names used in other well known and widely used OO methodologies (especially OMT[10]), even if the similarities are purely syntactic as can be seen throughout this paper.

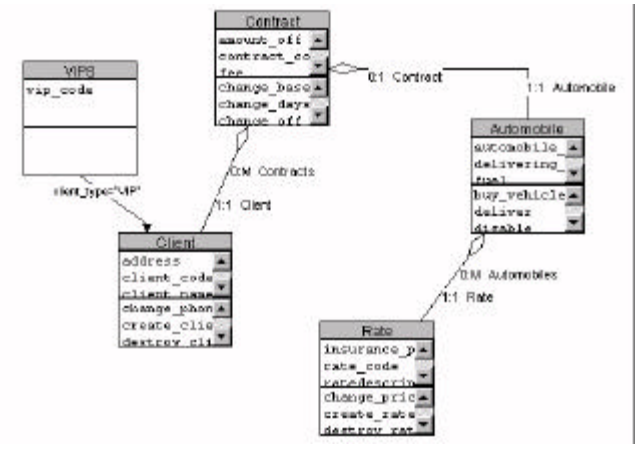
3.1 Conceptual Model

It is composed of:

I) Object Model

In the method, a Class Configuration Diagram (CCD) represents the object model, being unique in the system. It represents the static system view and uses the standard UML notation.

Figure 1



The OASIS concepts of: aggregation, inheritance, attributes events (private and shared), local transactions, agents, derivations and restrictions are specified in this diagram.

Drawing the CCD of our example, the structure relations of aggregation and inheritance (besides the class structures) are graphically depicted in a UML-compliant way.

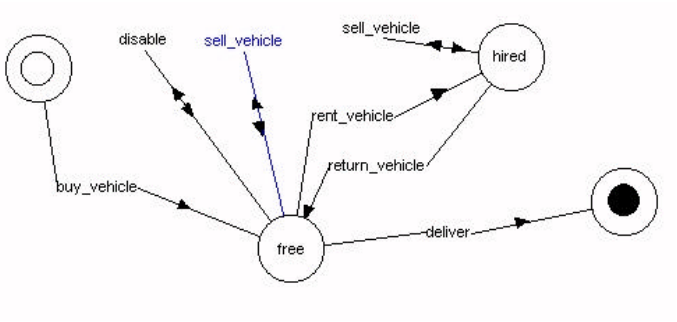
II) Dynamic Model

The method represents the system dynamics through two standard diagrams:

a) *State Transition Diagram (STD)*

With this diagram we fix the class dynamics, in other words, the possible life (or valid states) for the class objects. It has its correspondence with the specification process paragraph in OASIS. The valid lives of an automobile object in our example are declared in this diagram:

Figure 2



b) *Object Interaction Diagram (OID)*

This diagram fixes up the system interaction skills. There is only one interaction diagram associated to the whole system and it represents the concepts of triggers and global transaction in OASIS. As an example we represent the trigger presented before in a graphical way following the UML notation:

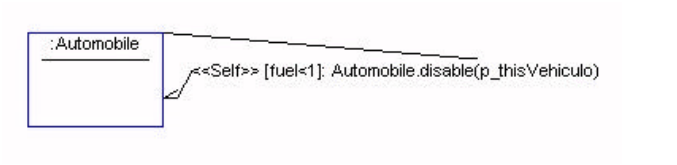


Figure 3

III) Functional Model

The aim of the Functional Model is to capture semantics attached to any change of state. This model specifies the effect of an event on its relevant attributes in a tabular way (condition, event and effect) and in the CASE implementation this model is represented through an interactive dialogue, which offers the chance to write the OASIS evaluation formulas. In this example it is shown how is updated the state of a client when

changes its phone number:

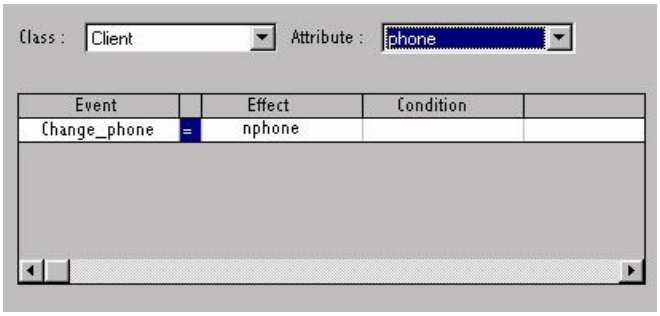


Figure 4

3.2 Execution Model

In the second main OO-Method phase, once an appropriate conceptual system description is obtained, the Execution Model sets the implementation details for the automatic generated software product in order to determine user interface, access control, service activation, etc.

Any service execution is characterized as the following sequence of actions performing the corresponding OASIS concepts:

a) Object identification: as a first step, the object who acts as server has to be identified. This object existence is an implicit condition for executing any service, except if we are dealing with a *new*[1] event. At this moment, their values (those that characterize its current state) are retrieved.

b) Introduction of event arguments: the rest of the arguments of the event being activated must be introduced.

c) Current state correctness: we have to check that the selected service can be activated in the current object state.

d) Precondition satisfaction: the precondition associated to the service that is going to be executed must hold. If not, an exception will arise, informing that the service cannot be activated because its precondition has been violated.

e) Valuation fulfillment: once the precondition has been checked, the induced event modifications are effective in the selected persistent object system.

f) Integrity constraint checking in the new state: to assure that the service activation leads the object to a valid state, we must check that the (static and dynamic) integrity constraints hold in this final resulting state.

g) Trigger relationships test: after a valid change of state, and as a final action, the sets of rules condition-action that represent the internal system activity have to be checked. If any of them holds, the corresponding service activation will be triggered. It is the analyst's responsibility to assure the termination and confluence of such triggers.

The previous steps guide the implementation of any program to assure the functional equivalence among the object system description collected in the conceptual model and its reification in a software-programming environment according to the execution model.

This model is a good starting point to develop software products, but we can consider it as quite abstract. Thus, a set of design decisions will complete it for obtaining a better code, trying also to take advantage of the particular features provided by the programming environment. Following this idea we show the main considerations in order to develop an automatic generated object-oriented programming environment.

3.3. Design Decisions

The first decision to take is the separation between the interface of a generated prototype and its functionality [9]. With the advent of increasingly more efficient Web technology, this separation gives us the possibility of adapting these prototypes to the Internet technology of distributed objects present at the market: Distributed Component Object Model -DCOM- [2] or Common Object Request Broker Architecture -CORBA- [3]. In this way, we generate ActiveX components (DCOM perspective) that can be executed and incorporated into different Windows transactional environments. Our CORBA code generation is currently under study. An important advantage of this type of architecture is to take profit of three-tier systems, implementing a thin client -interface- (for instance HTML pages), a middleware in who the application relies on its functionality, and finally, the required scripts for the Data Base Management System.

There is an additional issue underlying this first decision: the problem of editing the generated code for customization purposes. Obviously, the

[1] Formally, a new event is a service of a metaobject representing the class, which acts as object factory for creating individual class instances. This metaobject (one for every class) has as main properties *the class population* attribute, the next oid and the quoted *new* event.

simplest solution is the one that only allows edition of requirements in the Conceptual Model, and never retouching the generated code. This lead us to the reuse of specifications instead of the reuse of pure code, where sometimes it is not easy to detect the traceability of the requirements modeled. This situation is due to the fact that the OASIS object model (concepts of class, aggregation and inheritance) is not exactly the same as the object model supported by a object-oriented programming language such as Visual C++, Visual Basic, etc.

Every concept not supported by a programming language (object-oriented or not) must be replaced by a supplementary and functionally equivalent code. Generating ActiveX components, that is to say, *binary components*, the user will never re-implement it for customizing. Nevertheless, the user could modify the client code for these purposes; but it will be under his responsibility for maintaining the consistency of the model and its associated database.

The second important decision is related to the way the user interacts with the object environment that the final prototype implements. In a first step, strictly guided by the Execution Model, we thought that the appropriate way to interact was to choose the class we wanted to work with. Thereafter, choosing the event that we wanted to execute, a visual form requesting parameters was displayed (also asking for the object identifier) and, once accepted, the rest of the Execution Model was accomplished. Contrasting this point of view with the final users, we found that in some cases it is more ergonomic to select firstly the object and, then, to choose the event to execute. The experience revealed that there are some tasks that are more suitable for the first case, and others more suitable for the second. The point is that the second way is far more complicated to implement automatically, but that is not a user's problem.

The third design decision is related to the Data Base Management System (DBMS). The prototypes that we automatically generate use an standard Relational DBMS approach, due to its great industrial acceptance. Even though we rely on Relational DBMS, it is important to comment that the code that is automatically generated can be organized in a way that allows its portability if a degree of independence is considered on its code design. The alternative to this independence is to encapsulate the access to the database in the components that represents the classes modeled in the Conceptual Model. This is the solution finally chosen because it is more convenient for three-tier software architectures.

We are aware of the importance acquired by the Relational DBMS that will make them, probably, last for a long time before other kind of DBMS became so popular and developed as the relational data base technology is now at the market. But, it cannot be denied that from our point of view, using an object-oriented DBMS would make easier the process of automatic translation because it implies to work with a homogeneous paradigm and not changing from the object-oriented model to the relational model.

Using the capabilities of a Relational DBMS we can take profit of the security mechanisms that they provide establishing a security model for the objects created when the specification is animated by the prototype. For instance, when the user of the prototype creates an object instance of a class that is able to execute an event in the system modeled, a call to a stored procedure is executed in order to register this new object as a possible -active- user of the database. From this moment, the object can connect itself to the prototyped system with its own database user, and therefore, every action executed its registered in the log for possible audit purposes. If we work with a particular Relational DB not supporting the definition of a security model, we must generate an alternative one by generating code to deal with this issue.

Another point that is advisable to use from the Relational DBMS is the possibility of implementing the control of cardinalities, derived from the implementation of the aggregation concept in OASIS, using stored procedures of the database [8]. They make the implementation more efficient reducing the overload of traffic on the net using three-tier architectures. It is important to bear in mind that any work that the server side can do, surely, it will be carried out more efficiently by the DBMS. This rule has implications also in how the trigger concept in OASIS can be implemented using the trigger concept supplied by the database.

Summing up the design decisions, we can classify them as follows:

- About the system architecture:
  Choosing a particular architecture for the automatically generated prototype. Namely, it can be three-tier, client/server, or other type of architectures.
- About the interaction with the user:
  Selecting what action we want to do and then choose the target object that is going to provide it. Or the alternative way, select an object and tell what action you want to do.
- About the Data Base Management System:

Dealing with persistence issues, security issues and efficiency issues.

Furthermore, there are some crucial points that any person who wants to implement a code generator for the CASE tool must be aware of:

- The importance of establishing all the differences between the object model of target environment (concept of class, inheritance, etc) and the OASIS object model.
- The importance of proposing an analysis and design for the generated prototype in order to determine what the code generator must generate in the chosen architecture. This task must be done in advance.
- Applying metrics to the generated prototype for obtaining a better quality product.

4. The OO-Method/CASE Tool

This point revises the software production process focusing on the OO-Method/CASE tool. In this way we introduce to the reader in how the tool is structured and which items are the results of using it for resolving a specific problem.

As an information system, the OO-Method/CASE tool -implemented in Visual C++ 5- will receive an input that will process in order to obtain an output. Its aim is to build up another independent information system (a software product). This product, which is not tied to the tool in any way, is functionally equivalent to the requirements captured for the problem to solve (using the different models provided by the tool).

Nowadays, we are especially interested in generating products for rapid application environments, object-oriented and using web

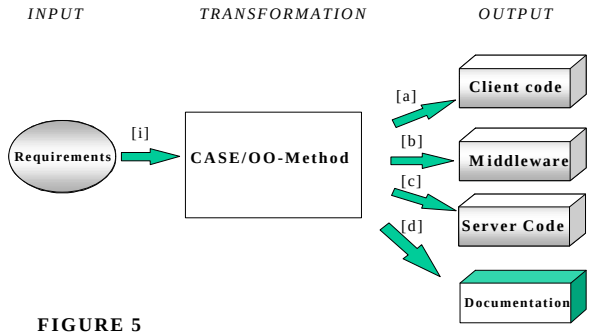


FIGURE 5

technologies. Our prototypes (products) can be generated for three-tier architectures and transactional environments. This is why the output can be divided into: client code, middleware, server code and documentation (see figure 5).

Next, a brief description for this components is presented:

a) Client Code

The client code is a set of ASCII-format files that determine the interface part of a program. It is the source code for a compiler of a programming language. Once compiled, it will invoke the compiled middleware (other step) in order to serve the client requests. Depending on the target environment, we can have Java code, Visual C++ code, Delphi code, Visual Basic code or even HTML pages that no need to be compiled (they are interpreted by a Internet browser). Currently, the most refined generator we have developed is constructed for Visual Basic 5. But, the problem is that we have not gathered at the conceptual level so much information for generating a good quality interface. This point is commented in the future work on this report. For instance, a basic interface automatically generated with Delphi for our example is:

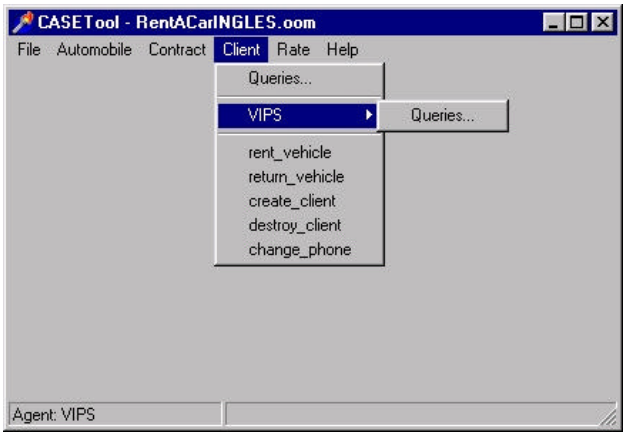


Figure 6

b) Middleware

As we have just mentioned, the middleware has the mission of solve the functionality requested by the interface (client code). This middleware is also a set of ASCII-format files that are compiled by an environment that produces an ActiveX component as a result of the process. In this case, the Visual Basic generator is our most developed code generator. Compiling this middleware ASCII files we obtain an ActiveX Dynamic Link Library that can be incorporated to a transactional server; currently, the Microsoft Transaction Server. As a very simple example we show an excerpt of a component that implements the valuation fulfillment

section of the Change_Phone event in the Client class specified in our example:

```
procedure TClient.change_phone;
begin
 Client.phone:=nphone;
end;
```

c) Server Code

The server code is set of scripts in an ASCII format that contains the Standard Query Language (SQL) sentences for constructing a Database, which will be the repository for the information system that we want to solve. These scripts can also contain sentences or stored procedures for specific Data Base Management System. At the moment, the CASE tool allows to generate scripts for SqlServer 7, Informix 7.2, Oracle 7, Access 7 as well as standard SQL server code. Moreover, the CASE tool can execute the SQL sentences through Open DataBase Connectivity (ODBC). Now, the middleware part of a Visual Basic generated application works with SQLServer 7. In the following picture we can see the database automatically generated in our example displayed with the browser that Access97 provides:

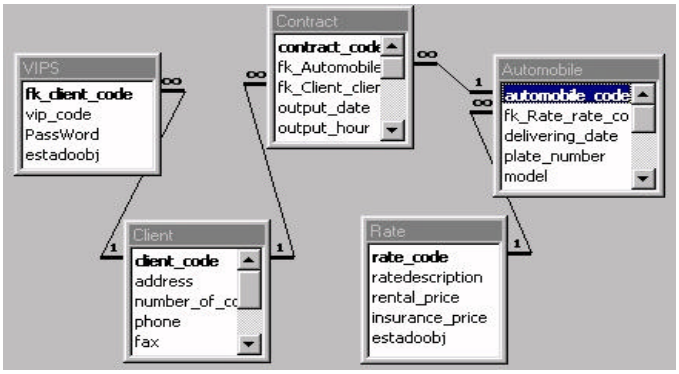


Figure 7

d) Documentation

It is the last ASCII file that is left to comment. It contains the text of the OASIS specification corresponding to the problem to solve modeled using the tool. It reflects directly the requirements of the problem in our formal and object-oriented specification language that can be useful as documentation.

Taking a look inside at the box named CASE in the Figure 5, some interesting issues arise.

Figure 8 is representing the process of code generation strategy for three-tier architectures. In other types of architectures, for instance

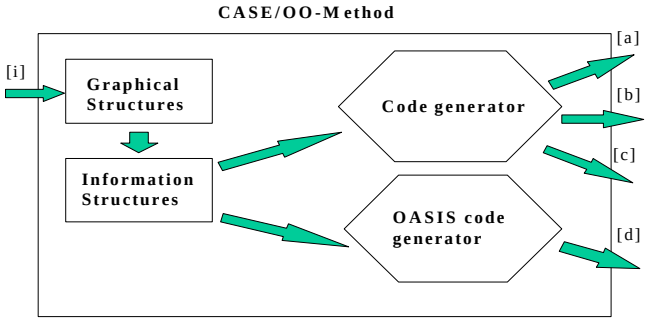


FIGURE 8

client/server, the output from the specific code generator (for a particular environment), can have another set of output (typically client and server code without a middleware). Each target environment or programming language will have its own particularized code generator.

The requirements of the problem are introduced in the tool drawing OASIS concepts in a UML notation (by means of the Conceptual Model diagrams) that are directly validated, transformed and stored in the information structures (OASIS repository of the CASE tool). Thereafter, the information gathered about the requirements is ready to be globally validated and used by the different code generators which include the CASE tool.

The code generator, depending on the design decisions commented in this report, will generate the source code of an application that will execute the Execution Model steps in a particular environment.

It is remarkable that both the code generators and the automatically generated prototype have their own analysis; as well as the CASE tool has its own different one.

5.Conclusions

Due to the experience acquired while putting our methodology into practice, a special emphasis is added on this report for taking into account several design decisions.

One of the most important characteristics in OO-Method has become its independence between itself and any fashionable rapid development environment. Moreover, using this method the analyst is never chained to neither an imperative nor a declarative environment. But, when we have the determination of generating code for a specific environment in a particular programming language, we must take this kind of decisions. They are crucial in order to generate efficient products automatically from a conceptual model in OO-Method. The reason

is quite simple, we are automatically implementing a program, and generic representations must be concretized.

The biggest problem of the methodology, at the moment, is to capture interface information at the conceptual level. This issue is our present and future work in both the academic and practice fields. The more information we have collected in the conceptual model, the more quality will have the automatically generated prototype. In that way, we talk about a product and not about a simple prototype which includes functionality.

Acknowledgments

We want to thank all the people that have made feasible the implementation of the OO-Method/CASE Tool, especially Jose Barbera and Jose Merseguer who actively collaborated in the elaboration of the present paper.

References:

[1] Balzer, R. et al. Software Technology in the 1990s: Using a New Paradigm. IEEE Computer, Nov. 1983.

[2] Microsoft COM Technologies [online]. February 1999. From World Wide Web: < http://microsoft.com/com/default.asp>

[3] Object Management Group Home Page [online]. February 1999. From World Wide Web: < http://www.omg.org>

[4] Hartmann T.,Saake,G.,Jungclaus,R.,Hartel,P.,Kusch,J. *Revised Version of the Modeling Language Troll (Troll version 2.0).* Technische Universitat Braunschweig, Informatik-Berichte, 94-03 April 1994.

[5] Pastor, O.;Hayes,F.;Bear,S. *OASIS:An OO Specification Language*. Proc. of CAiSE-92 Conference, Lncs (593), Springer-Verlag 1992, pags: 348-363.

[6] Pastor,O., Merseguer, J., Romero, J., Barberá, J.M. : *The CASE OO-METHOD graphic environment.* Technical Report, DSIC-UPV, 1996.

[7] Pastor, O.; Romero, J.; Pelechano, V.;Insfran, E.;Merseguer,J.: *OO-METHOD: An OO Software Production Environment Combining Conventional and Formal Methods*. 9th Conference on Advanced Information Systems Engineering (CAiSE'97). Barcelona, Spain. June 1997. LNCS (1250), pages 145-159. Springer-Verlag 1997. ISBN: 3-540-63107-0.

[8] Pastor, O.; Romero, J.; Barberá J. *Mapping aggregation from modeling to object-oriented programming*. Proceedings of the 3rd International Conference "The White Object Oriented Nights' (WOON'98). Editors Alexander V.Smolyaninov,Alexei S.Shestialtynov.St Petesburg (Russia)

[9] Romero, J.; Pastor O. *Diseño de ambientes software orientados al objeto de prototipación automática*. IV Jornadas de tecnología de objetos. University of Deusto, Bilbao, España

[10] Rumbaugh J.,Blaha M., Permerlani W., Eddy F.,Lorensen W. *Object Oriented Modeling and Design*. Englewood Cliffs, Nj. Prentice-Hall 1991.

[11] Jungclaus, R., Saake, G., and Sernadas, C. Formal Specification of Object Systems. In Abramsky, S. and Maibaum, T., editors, Proc. TAPSOFT'91, Brighton, pages 60--82. Springer, Berlin, LNCS 494

[12] Booch,G.;Rumbaugh,J.;Jacobson,I. *Unified Modeling Language (UML summary). Version 1.0 January 1997.* Rational Software Corporation.