



**OO-Method, the methodological support
for the
Oliva Nova Model Execution System**

**From Object-Oriented Conceptual Modeling to Automated
Programming**

by

**Prof. Dr. Oscar Pastor, Valencia University of Technology,
President of CARE Technologies Scientific Advisory Board.**

**Emilio Insfrán, Valencia University of Technology
Research Team Leader at CARE Technologies.**

TRADEMARKS

OlivaNova is a trademark of CARE Technologies S.A.

Windows is a trademark of Microsoft Corp.

Java is a trademark of Sun Corp.

While the information in this publication is believed to be accurate, CARE Technologies makes no warranty of any kind to this material including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. CARE Technologies shall not be liable for errors contained herein, or for incidental or consequential damages in connection with the furnishing, performance or use of this material.

COPYRIGHT NOTICE

No part of this publication may be reproduced, stored in a retrieval system or transmitted, in any form or by any means, photocopying, recording or otherwise, without prior written consent of CARE Technologies. No third-party intellectual property right liability is assumed with respect to the use of the information contained herein. CARE Technologies assumes no responsibility for errors or omissions contained in this white paper. This publication and features described herein are subject to change without notice.

Copyright © 1999-2003 CARE Technologies. All rights reserved.

All products or services mentioned in this white paper are covered by the trademarks, service marks, or product names as designated by the companies that market those products.

EXECUTIVE SUMMARY

Current and future (conventional) notations used in Conceptual Modeling techniques should have a precise (formal) semantics to provide a well-defined software development process, with the purpose of going from specification to implementation in an automated way. To achieve this objective, the OO-Method approach to Information Systems Modeling introduced here tries to overcome the conventional (informal) / formal dichotomy by picking the best ideas from both approaches.

In the OO-Method a clear distinction is being made between the problem space (centered on what the system is) and the solution space (centered on how it is implemented as a software product). It provides a precise, conventional graphical notation to obtain a system description at the problem space level, but this notation is strictly based on a formal OO specification language that determines the conceptual modeling patterns needed to have the system specification. An abstract execution model determines how to obtain the software representations corresponding to these conceptual modeling patterns. In this way, the final software product can be obtained in an automated way.

Table of Contents

1	Introduction.....	4
2	OO-Method.....	6
2.1	Conceptual Model	6
2.1.1	Object Model	7
2.1.2	Dynamic Model	8
2.1.3	Functional Model	8
2.1.4	Presentation Model	9
2.2	Execution Model	11
3	Conclusions.....	12
4	References.....	14

1 Introduction

If the software production process is understood to be a true engineering activity, a comprehensive view of the involved set of phases must be introduced. This view must be global, in the sense that a clear relation between the different steps has to be defined.

We are concerned with how to improve the software development process, for the reasons detailed below:

- Moving out from the chaotic level associated with conventional methods, where contradictions, ambiguities, incompleteness and mixed levels of abstractions occur in a natural way.
- Improving understanding at the problem space level.
- Designing flexible methods and better CASE (Computer-Aided Software Engineering) / CARE (Computer-Aided Requirements Engineering) support, with the objective of reaching the solution space with a final software product which is compliant with the problem space description.
- Providing guidance in order to be able to deal with the entire software production process in a structured way.

To overcome these shortcomings, we should move from traditional systems engineering to advanced requirements engineering, where the distinction between the problem space (what the problem is) and the solution space (how it is to be implemented in a particular software development environment) must be clearly established. The conversion of the conceptual modeling patterns (which are included in the resultant conceptual model) into the final software components should also be clearly determined.

It is obvious that the problem space imposes strong requirements on the software production process. Taking into account that the Requirement Engineering process is complex, knowledge-intensive, experience-based, and requires highly intellectual and creative activity, it is essential to provide a clear way of capturing those requirements considered relevant in order to build a correct conceptual model. The first decision to make is to determine the model to be used for dealing with the conceptual modeling process. From our point of view, the object-oriented model is the best choice, due to its proximity to human cognitive mechanisms, due to the encapsulation of structural and behavioral aspects provided by the object notion and due to the fact that the transition from the problem space to the solution space is smooth. We can specify objects in the conceptual model and implement objects in the final software representation. The object-oriented

model introduces modularization in the problem space, making reuse a good practice in the conceptual modeling process.

If we look at the current approaches that deal with object-oriented conceptual modeling, we can distinguish two main categories:

1. Those conventional methods such as Wirfs-Brock [1], Rumbaugh [2], Jacobson [3], Booch [4], Coleman [5], usually UML-based from the notation point of view. They inherit what we could call traditional CASE (Computer-Aided Software Engineering) problems:
 - An excessive number of models that have overlapping semantics.
 - Difficult and lossy transformations between models, languages and tools.
 - Variable quality in system implementation under similar quality designs.
 - Lack of comprehensiveness due to needless complexity.
2. Methods that are created using a formal framework such as Kush [6], Clyde [7], Jackson [8], Liddle [9], Liu [10] to define concepts in a precise way, and where for every behavioral pattern captured in the conceptual modeling step (problem space), there is a corresponding construct in the software representation used at the solution space.

We argue that future developments in Software Engineering follow this second approach, but properly linked with the well-known standard notations provided by the methods of the first approach. The main objective is to use a notation to represent a precise set of conceptual primitives, all of them with their corresponding formal counterpart. In the last few years we have been working on integrating the two approaches resulting in the OO-Method.

The OO-Method is built on a formal object-oriented model (OASIS). Its main feature is that developers' efforts are focused on the conceptual modeling step, where system requirements are captured in accordance with a predefined, finite set of what we call conceptual modeling patterns because they are a representation of relevant concepts at the problem space level. The final implementation is obtained in an automated way by programming the corresponding mappings between conceptual model constructs and their representation in a particular software development environment.

This architecture gives the pattern for obtaining software components, which can be dynamically combined to build a

software product that is functionally equivalent to the specification collected through the conceptual model. Currently the Oliva Nova Transformation Engine supports COM+ and J2EE. However, it is important to note that this approach ensures that models are independent of any target development environment.

2 OO-Method

The OO-Method was created using OASIS, an object-oriented formal specification language for Information Systems [11]. Basically, we can distinguish two modeling components in the OO-Method: the conceptual model and the execution model. Both are supported by the Oliva Nova Modeler and the Oliva Nova Transformation Engine respectively.

2.1 Conceptual Model

Two main aspects must be clearly defined when introducing a conceptual modeling approach:

1. The conceptual modeling patterns provided by the method.
2. The notation that properly captures the above conceptual modeling patterns.

Detailing the specific set of modeling patterns and notations would be out of the scope of this report.

The OO-Method proposes the combination of formal specification techniques with conventional, widely used, OO modeling techniques. In other words: avoid the complexity that is traditionally associated with the use of formal methods, by making designers view the method as being compliant to industrial standards.

This is why we adopted the well-known OMT strategy of dividing the conceptual modeling process into the three complementary views: the object view, the dynamic view and the functional model view (later adding a fourth view to specify user interfaces). However, there is a big difference in respect to OMT: modeling with an OO-Method tool, such as the Oliva Nova Modeler, to specify a system implies capturing the formal specification of that system "on the fly", according to the OASIS formal specification language.

Thanks to this characteristic it is possible to introduce a well-defined expressiveness in the specification that conventional methodologies are lacking. The use of such a formal specification provides a suitable context to validate the system specification in the problem space, and to verify the system software implementation in the solution space.

The obtained software product is functionally equivalent to the specification. This is achieved by creating a model compiler that implements all the mappings specified between the conceptual patterns (problem space level) and their software representations (at the solution space level). Naturally, it is mandatory to introduce all the data that OASIS requires, using the mentioned OMT-like models (object model, dynamic model, functional model and, user interface model) through their corresponding diagrams. Nevertheless, this is always done keeping the analyst's view of the model compliant with widely accepted modeling techniques. For example, to define a class only the relevant information for creating a class in OASIS must be introduced. In this way, the arid formalism is hidden by the modeler when describing the system, making the analyst feel more comfortable.

With respect to the notation, conceptual modeling in the OO-Method uses UML-compliant diagrams. According to the previous arguments, one of the main design criteria of the OO-Method was to prevent the analysts from having to learn yet another graphical notation in order to model an information system. OASIS thereby determines the set of needed diagrams for the modeling environment.

The OO-Method collects the system requirements using four complementary models.

2.1.1 Object Model

The Object Model (OM) is a graphical model where system classes including attributes, services and class relationships (association and inheritance) are defined. Additionally, agent relationships are specified to state the services that objects of a class are allowed to activate. The corresponding UML base diagram is the class diagram, where the additional expressiveness could be introduced through the use of UML stereotypes.

Specifically, for every class the OM captures the information about attributes, services, derivations, constraints and relationships (aggregation and inheritance). More precisely, the additional information associated with aggregation and inheritance is the following:

- For associated classes, to specify whether we have an aggregation or a composition (following the UML characterization), and whether the association is static or dynamic;
- For inheritance hierarchies, to specify whether a specialization is permanent or temporal. In the former case, the corresponding condition on constant attributes must characterize the specialization relationship; in the

latter, a condition on variable attributes or the carrier service that activates the child role must be specified.

2.1.2 Dynamic Model

With the OM, the system class architecture has been specified. Additionally, basic features, such as which object life cycles can be considered valid, and which inter-object communication can be established, have to be introduced in the system specification. To do this, the OO-Method proposes a dynamic model. It uses two kinds of diagrams:

- **State Transition Diagrams** The state transition diagram (STD) is used to describe correct behavior by establishing valid object life cycles for every class. By valid life cycles, we mean an appropriate sequence of states that characterizes the correct behavior of the objects that belongs to a specific class.
- **Interaction Diagram** The interaction diagram (ID) specifies the inter-object communication. We define two basic interactions: triggers, which are object services that are activated in an automated way when a condition is satisfied, and global interactions, which are transactions involving services of different objects.

There is one STD diagram for every class, but only one ID for the whole system, where the previous interactions will be graphically specified.

2.1.3 Functional Model

A correct functional specification is a shortcut for many of the extended OO methods. Sometimes, the model used breaks the homogeneity of the OO models, as happened with the initial versions of OMT, which proposed using the structured DFDs (Data Flow Diagrams) as a functional model. The use of DFD techniques in an object-modeling context has been criticized for being imprecise, mainly because it offers a perspective of the system (the functional perspective), which differs from the other models (the object perspective). Other methods leave the specification of the system operations solely in the hands of the designer. The OO-Method functional model (FM) is quite different compared to these conventional approaches. In this model, the semantics associated to any change of an object state is captured as a consequence of a service occurrence. To accomplish this, it is declaratively specified how every service changes the object's state depending on the arguments of the involved service and the object's current state.

A clear and simple strategy is given for dealing with the introduction of the necessary information. The relevant contribution of this functional model is the concept of the categorization of attributes.

Three types are defined: push-pop, state-independent and discrete-domain based. Each type will specify the pattern of information required to define its functionality.

Push-pop attributes are those whose relevant events increase or decrease their value by a given quantity. Events that reset the attribute to a given value can also exist.

State-independent attributes have a value that depends only on the latest action that has occurred. Once a relevant action is activated, the new attribute value of the object involved is independent of the previous one. In such a case, we consider that the attribute remains in a given state, having a certain value for the corresponding attribute.

Discrete-domain valued attributes take their values from a limited domain. The different values of this domain model are the valid "situations" that are possible for objects of the class. Through the activation of carrier actions (which assign a given domain value to the attribute) the object reaches a specific "situation". The object abandons this "situation" when another event occurs (a "liberator" event).

2.1.4 Presentation Model

With the three previous models, object society structure, behavior and functionality are specified. The last step is to specify how users will interact with the system. This is done by the Presentation Model through the definition of a set of Presentation Patterns. The Presentation Patterns capture the information required to characterize what form the application will have, and how the user will see the problem modeled.

Those Presentation Patterns are organized in three levels:

- Level 1: The first level contains the Hierarchical Action Tree pattern, providing the access to the application.
- Level 2: The second level contains the Interaction Units. The user interface is then split into several scenarios to support user tasks.
- Level 3: Eventually, the third level is composed of patterns that add additional semantics for interaction units.

Again, the precise description of these patterns are out of the scope of this report, but Figure 1 globally shows the levels and the corresponding inter-dependencies among patterns.

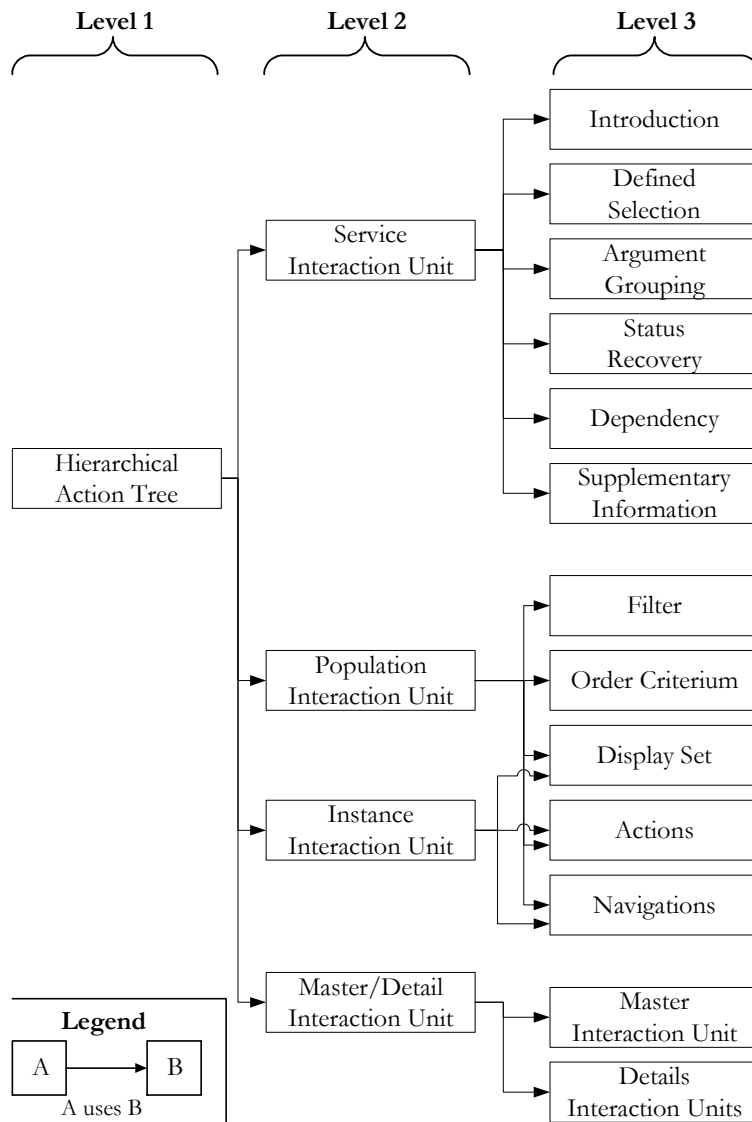


Figure 1. Pattern Language and usage.

2.2 Execution Model

The OASIS specification is the source for an execution model that must accurately state the implementation-dependent features associated with the selected object society machine representation. In order to easily implement and animate the specified system, we predefine a way in which users interact with system objects. The template used to achieve this behavior is:

1. Identify the user
2. Obtain the object system view
3. Service activation
 - 3.1 Identify the object server
 - 3.2 Introduce service arguments
 - 3.3 Send the message to object server
 - 3.4 Check state transition
 - 3.5 Check preconditions
 - 3.6 Valuations fulfillment
 - 3.7 Integrity constraint checking in the new state
 - 3.8 Trigger relationships test

The process starts by logging the user into the system (step 1) and providing an object system view (step 2), determining the set of object attributes and services that it can see or activate. After the user is connected and has a clear object system view, he can then activate any available service in his worldview. Among these services, we will have observations (object queries), local services and transactions served by other objects. Any service activation (step 3) has two steps: build the message and execute it (if possible). In order to build the message, the user has to provide information to identify the object server (step 3.1). Subsequently, he must introduce service arguments (step 3.2) of the service being activated (if necessary). Once the message is sent (step 3.3), the service execution is characterized by the occurrence of the following sequence of actions in the server object:

- Check state transition (step 3.4), which is the process of verifying that a valid transition exists in the OASIS specification for the selected service in the current object state.
- The precondition satisfaction (step 3.5) indicates that the precondition associated to the service must hold.

- If any of these actions do not hold, an exception will be raised and the message ignored. Otherwise, the process continues with:
- The valuation fulfillment (step 3.6) where the induced service modifications take place in the involved object state.
- To assure that the service execution leads the object to a valid state, the integrity constraints (step 3.7) are verified in the final state. If the constraint does not hold, an exception will be raised and the previous change of state ignored.
- After a valid change of state, the set of condition-action rules that represents the internal system activity is verified. If any of them hold, the specified service will be triggered (step 3.8).

The previous steps control the execution of any program to assure the functional equivalence between the object system specification collected in the conceptual model and its reification in an imperative programming environment. The Oliva Nova Transformation Engine provides a concrete execution model using a component-based architecture to deal with the peculiarities of component-based systems.

3 Conclusions

Conventional object-oriented methodologies have to provide a well-defined software development process by which the community of software engineers can properly derive executable software components from requirements in a systematic way. Jointly with CARE Technologies, we have addressed these problems in the context of the conceptual modeling approach supported by the OO-Method.

The OO-Method can be seen as a CARE environment (Computed-Aided Requirements Engineering) where in the context of the whole software production process, we focus on how to properly capture the system requirements. The resulting conceptual model specifies what the system is (problem space). The abstract execution model then guides the representation of these requirements in a specific software development environment (solution space, which is centered on how the system will be implemented).

The abstract execution model is based on the concept of Conceptual Modeling Patterns. The Oliva Nova Transformation Engine provides a well-defined software representation of these patterns in the solution space. The implementation of these mappings from problem space concepts to the solution

space opens the door to the automated generation of executable software components. Together these software components constitute a software product that is functionally equivalent to the requirements specification collected during conceptual modeling.

In summary:

- A methodological approach allowing analysts to concentrate on the problem/business space regardless of the implementation in the solution/technology space.
- The formalism in the specification allows the system to be completely defined, so it can be validated in the business space and later verified in the solution/technology space.
- An intuitive and graphical environment to the underlying formal language.
- A specific strategy to generate code based on a clear separation of component specification and component implementation to enable technology-independent application design.
- A well-defined component-based framework to execute the specification in the solution space
- The OO-Method is fully supported by Oliva Nova Model Execution software.

4 References

- [1] R. Wirfs-Brock, B. Wilkerson and L. Wiener. Designing Object Oriented Software. Prentice-Hall, 1990.
- [2] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy and W. Lorenzen. Object Oriented Modeling and Design. Prentice-Hall, 1991.
- [3] I. Jacobson, M. Christerson, P. Jonsson and G. Overgaard. OO Software Engineering: a Use Case Driven Approach. Addison-Wesley, 1992.
- [4] G. Booch. Object Oriented Analysis and Design with Applications. Second Edition. Addison-Wesley, 1994.
- [5] D. Coleman, P. Arnold, S. Bodo, S. Dollin, H. Gilchrist, F. Hayes and P. Jeremes. Object-Oriented Development: The Fusion Method. Prentice-Hall, 1994.
- [6] J. Kush, P. Hartel, T. Hartmann and G. Saake. Gaining a Uniform View of Different Integration Aspects in A Prototyping Environment. In Proceedings of DEXA'95 International Conference, volume 978 of LNCS, pages 35-42. 1992.
- [7] S. W. Clyde, D W. Embley and S N. Woodfield. Tunable Formalism in Object-Oriented Systems Analysis: Meeting the Needs of Both Theoreticians and Practitioners. In Proceedings of the OOPSLA'92 Conference on Object-oriented Programming Systems, Languages and Applications, pages 452-465, 1992.
- [8] R B. Jackson, D W. Embley and S N. Woodfield. Automated Support for the Development of Formal Object-Oriented Requirements Specification. In G. Wijers, S. Brinkkemper and T. Wasserman, editors, Proceedings of CAISE'94 International Conference, volume 811 of LNCS, pages 135-148. Springer-Verlag, 1994.
- [9] S. Liddle, D W. Embley and S N. Woodfield. Unifying Modeling and Programming Through an Active, Object-Oriented, Model-Equivalent Programming Language. In M. P. Papazoglou, editor, Proceedings of OOE'95 International Conference, volume 1021 of LNCS, pages 55-64. Springer-Verlag, 1995.
- [10] S. Liu, A. J. Offutt, C. Ho-Stuart, Y. Sun and M. Ohba. SOFL: A formal Engineering Methodology for Industrial Applications. IEEE Transactions on Software Engineering, 24(1):24-45, January 1998.
- [11] O. Pastor and I. Ramos. OASIS 2.1.1: A Class-Definition Language to Model Information Systems Using an Object-Oriented Approach. Servicio de Publicaciones. Universidad Politécnic de Valencia, 3rd edition, 1995.

CONTACT INFO

Spain:

CARE Technologies S.A.
Partida Madrigueres, 44.
03700 Denia, Alicante, SPAIN
care.technologies@care-t.com
Tel. +34 96 643 5555
Fax. +34 96 643 5554

USA:

SOSY Inc.
180 Montgomery Avenue, Suite 828
San Francisco, CA 94104
information@sosyinc.com
Tel. +1 415.362.4333
Fax. +1 415.362.4703

Germany:

CARE Technologies Deutschland GmbH
Wagmüllerstraße 18-20
80538 München
info@care-t.de
Tel. +49 89 21923774
Fax. +49 89 21923756

CARE Technologies white papers can be downloaded at <http://www.care-t.com>.