NIST-RCS and Object-Oriented Methodologies of Software Engineering: A Conceptual Comparison

Hui-Min Huang and Elena Messina
Intelligent Systems Division
National Institute of Standards and Technology
Gaithersburg, MD 20899
{huang, messina}@cme.nist.gov

Abstract

The hierarchical real-time control systems (RCS) reference model architecture that is under development at the National Institute of Standards and Technology aims at designing and developing intelligent control for large and complex systems. A methodology being developed to create RCS-based systems exhibits many characteristics that are similar to object-oriented paradigms. The authors compare certain key attributes of object-oriented approaches to the RCS methodology. They find many similarities and suggest that RCS provides unique multiple resolution, behavior-oriented features which can be considered to go beyond most object-oriented paradigms. Examples from a recent RCS testbed for a manufacturing inspection system are detailed for clarification.

Keywords: hierarchical systems, intelligent control, methodology, object-oriented

1. Introduction

The Intelligent Systems Division (ISD) of the National Institute of Standards and Technology (NIST) has been researching and developing a reference model architecture for hierarchical real-time control systems. The reference model is called the NIST Real-time Control System (RCS) [ALB96-1]. ISD is also describing a methodology to create RCS-based systems. RCS aims at designing and developing intelligent control for large and complex systems. RCS brings forth a distinctive behavior-oriented paradigm to manage system complexity.

One of the most recent applications of RCS is the NIST National Advanced Manufacturing Testbed (NAMT) project. The NAMT is established to allow scientists and engineers from industry, NIST, other government agencies, and academia to work together to solve measurement and standards issues in information-based manufacturing. NAMT also develops the needed tests and test methods for industry that are part of NIST's mission. In this paper, the authors use the NAMT inspection subsystem to illustrate their research findings.

Object-oriented (OO) paradigms are emerging as a major methodology for system development. Although there are skeptics, it is generally believed that these object-oriented paradigms provide several significant advantages over some traditional methods, such as functional decomposition, structured design and analysis, and information engineering in that OO paradigms focus on the depiction of the real-world. Object-oriented methodologies typically refer to three distinct activities which are often, although not necessarily, interrelated: Object-oriented analysis, object-oriented design, and object-oriented programming.

The authors attempted to compare RCS and OO. They discovered that, although both methodologies emphasize depiction of the real world, RCS, additionally, describes a reference model and a process for engineering and structuring real world entities to achieve user-defined goals. The authors describe their findings in this paper. The comparison reveals that RCS is an object-oriented methodology which focuses on behavioral abstraction. Note that due to the existence of the variety of OO methods and the wide span of system life cycle these OO paradigms cover, the comparison and example given in this paper may reference only part of the general OO concepts or particular OO methods. In particular, the authors compared RCS with the inheritance, abstraction, association, aggregation, and encapsulation concepts of the OO paradigms.

2. Object-Oriented Concepts

For in-depth descriptions of object-oriented methodologies, the reader is referred to texts like [BOO94], [COA91], [RUM91], or [SHL92]. A very brief definition of the OO concepts discussed in this paper is included for the reader's convenience.

Inheritance is "the sharing of attributes and operations among classes based on a hierarchical relationship" [RUM91]. For example, lathes and milling machines are subclasses of machine tools. They would inherit the properties of the class "machine tools," such as having multiple axes, a tool holder and tool changer, or the ability to remove material.

Inheritance may take the forms of *specialization* or *augmentation*. A class derivation may involve tailoring the features defined in the base class to the needs of the derived class. A derived class may also contain features in addition to the base class [DEW89].

Abstraction is eliminating nonessential information when considering an object's properties and behavior. What is considered essential about an object may vary according to the development stage or to the application in which the object is to be used.

Associations establish the relationships among objects or classes. The relationships are described at *Links* that connect objects or classes [RUM91].

Aggregation is a form of association where a relationship is established between objects that represent components of an assembly and an object representing the entire assembly [RUM91]. The assembly subsumes the components via the aggregation. The assembly may derive its properties from the component properties. For example, an inspection system consists of a table, a three-axis arm, a probe, and a control box, with monitor, CPU, and other components. Because the components of the control box are spelled out in this example, this is a multi-level aggregation. When convenient, the inspection system can be referred to by its aggregate identity. Yet the option of referencing its constituent elements is still available. Properties may apply to the aggregate. For example, a system can be modeled such that its total mass is a sum of all of its component's masses.

Encapsulation is used to hide object information that should not matter to the external objects. The internal representation of information and other implementation details which are irrelevant to the access of and communication with an object should not be exposed.

3. The RCS reference model

3.1 The intelligent machine system model

RCS originates with an intelligent machine system model (IMS), as shown in Figure 1. The model contains a behavior generation (BG) function that makes decisions based on the received task commands and on the current state of the world. BG is supported by world modeling (WM), value judging (VJ), and sensory processing (SP) functions [ALB96-1], [BAR84].

BG employs a task planning function that generates a set of possible schedules. WM simulates these candidates and generates the predicted results. VJ uses a set of cost functions to judge the simulation results and provides the values of these schedule candidates for the BG planning function to determine a final schedule for execution.

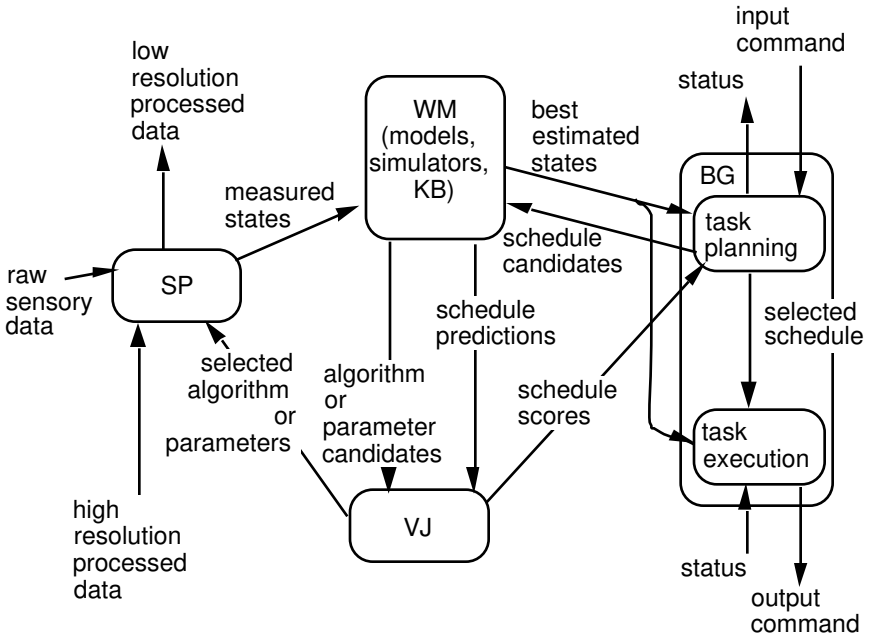

Figure 1: RCS IMS reference model

SP processes sensory data and generates measured states for the system. Different processing algorithms or algorithm gain values may need to be determined for particular data sets. This process may involve WM and VJ, as the left side of Figure 1 shows. Figure 1 also shows that SP handles data with multiple resolutions. The remainder of this paper describes this multiple resolution concept.

3.2 Multiple resolutions

RCS is a hierarchical architecture that was developed through spatial and temporal decompositions. Higher levels operate in a state space with larger spa-

tial and temporal span but coarser resolution. Rules which guide the hierarchical decomposition with respect to resolution of space and time are based on control theory and biological evidence. In his "Outline for a Theory of Intelligence," [ALB91], Albus proposed a theorem which states,

> "In a hierarchically structured, goal-driven, sensory interactive, intelligent control system architecture:
> 1. control bandwidth decreases about an order of magnitude at each higher level,
> 2. perceptual resolution of spatial and temporal patterns decreases about an order of magnitude at each higher level,
> 3. goals expand in scope and planning horizons expand in space and time about an order of magnitude at each higher level, and
> 4. models of the world and memories of events decrease in resolution and expand in spatial and temporal range by about an order of magnitude at each higher level."

By following these guidelines, RCS-based systems have a sound basis for decomposing a system at its constituent resolution levels. Depending on the scope of individual problems, RCS prescribes up to six types of levels that form a smooth transition of spatial and temporal resolution from the highest to the lowest levels. These levels are application domain, group, equipment, emove (kinematic), primitive (dynamic), and servo. Each level may have zero or multiple control nodes that are modeled after the IMS (except for the highest level which has one node). At each level of the hierarchy, the control for individual and collective physical entities, such as robots, vehicles, workstations, manufacturing shops, and robotic motors, are modeled. Note that if the problem is for the control of a machining center only, then the workstation and even upper level control become not applicable.

These guidelines also help operators to understand and anticipate the way a high-level goal can be decomposed to low-level and detailed behavior.

3.3 Behavior and behavior generation

Albus and Meystel defined behavior as "an ordered set of consecutive or concurrent changes among the states registered at the output of the system or subsystems [ALB96-2]." In a complex system, agents or subsystems are capable of performing certain behaviors. A mechanism is required to coordinate the individual behaviors to form system behavior. This mechanism does not necessarily have to be external to the agents, but can be embedded within the agents themselves and manifest itself as "cooperation."

In RCS, nodes generate behavior through planning and execution processes. The objective of the generated behavior is to command and coordinate the sub-behavior of all the nodes' subordinates. The system behavior may be initiated when a user enters a high-level goal that results in the actuation of the hardware components to achieve the goal within a given tolerance. In the NAMT Next Generation Inspection Workstation (NGIS) testbed, a system goal could be "Inspect_part." The performance of this goal can involve a series of sub-behaviors such as "Inspect_feature." The performance of inspecting a specific feature can involve a series of even lower level "Go_to_point" sub-behaviors to be conducted by the inspection arm controller. These sub-behaviors are further decomposed until individual motor behavior is generated.

Note that OO also uses the term behavior either generally or specifically, but in a different context [BOO94]]. In this paper, the term behavior refers to the definition given in this section.

3.4 Command authority

The combination of the multiple resolution (described in section 3.1) and the behavior generation concepts forms a relationship of command authority between the superior and the subordinate control nodes. The superiors command the subordinates. The subordinates report back the status of command execution.

4. Assessing the behavioral aspect within the two paradigms

4.1 Behavior as a principle of abstraction

Abstraction is defined as denoting the essential characteristics of an object that distinguish it from other objects, while suppressing other details. Different types of abstractions are described in various OO paradigms, including object, class, data, function, and process [BOO94]. The abstraction relationship among objects within a system leads to system hierarchies. Functional abstraction leads to a hierarchy with functional decomposition.

RCS views goals and behavior as the most important aspects of an intelligent system. Therefore, RCS uses behavior abstraction to derive control system hierarchies. In other words, the tasks that each node can perform, which are limited in amount and are compliant with the resolution requirement, may be referred to as the "task abstraction" or "behavior abstraction."

4.2 Analysis of a system with respect to its behavior

RCS provides for an analysis phase during which the available system is methodically dissected to glean the basic components, hierarchy, and information or commands required [QUI93]. These aspects bear a strong resemblance to Object-Oriented Design and Object-Oriented Analysis.

All of the RCS applications thus far have controlled either physical systems or their simulations, such as machine tools, robots, submarines, or autonomous vehicles. However, at the beginning of the analysis and design phase, physical entities may or may not be totally available. The developers may need to specify and acquire certain sensors and communication systems. The developers must closely examine the available physical entities in terms of their sensing, actuation, and communication capabilities. The developers must then closely interleave the physical system analysis and the task analysis to construct a hierarchical control system. This control system must manifest the multiple resolution behavior generation capability. At the same time, it is advantageous to have a software node configuration that is as close to the existent physical entities as feasible. This would preserve the control system's depiction of the physical world. The behavior oriented analysis may also dictate the additional physical entity requirements as part of the design effort. Operational scenarios are typically developed to help analyze the system behavior, explain the system operational characteristics and the control flow, and validate system requirements. The developer must reflect the system requirement specifications in these scenarios. Such a process may help bring to light unspecified, unclear, or unattainable requirements.

4.3 Behavior orientation and object orientation

Behavior is considered part of an object's model in most of the object-oriented methods. For example, Coad and Yourdon describe parameters such as frequency and location which control a radar system's behavior [COA91]. Shlaer and Mellor describe real-world things as having stages in their behavior patterns [SHL92]. However, there is a subtle distinction between "behavior as a part of objects" in the object-oriented methods and the "behavior-oriented" methodology in RCS. From a global perspective, the system behavior determines an RCS hierarchy and the underlined control flow. In RCS, developers may choose to analyze and identify system behavior before designing control nodes, or "objects," to perform the behavior. This concept has not been emphasized in most object-oriented methods. The authors believe that this behavior orientation could significantly strengthen the general object-oriented methods, particularly in the large complex real-time control problems that the RCS architecture excels.

5. Assessing the object and inheritance aspects

A careful study shows that RCS contains many object-oriented concepts. However, these OO concepts are refined and given explicit meanings in RCS.

5.1 Control node objects with a generic processing model

An RCS hierarchy is composed of the control nodes. The control nodes have a generic functional model, IMS, as described in Section 3.1. In a simplest case, one can view IMS as a type of object. This OO representation leads to the creation of an IMS based OO base class. The base class serves as a single building block, or progenitor, for the design of RCS systems.

The IMS based base class also contains basic message passing and processing functionality. Typical operations that must occur every control cycle may include processing input buffers and writing to output buffers (commands, status, and world modeling updates) according to prescribed rules. This approach has been used successfully in several major RCS implementations [ALB96-1][HUA96].

A general OO representation of IMS still is under investigation. One possibility is to model each of the functional modules or submodules (SP, planning,

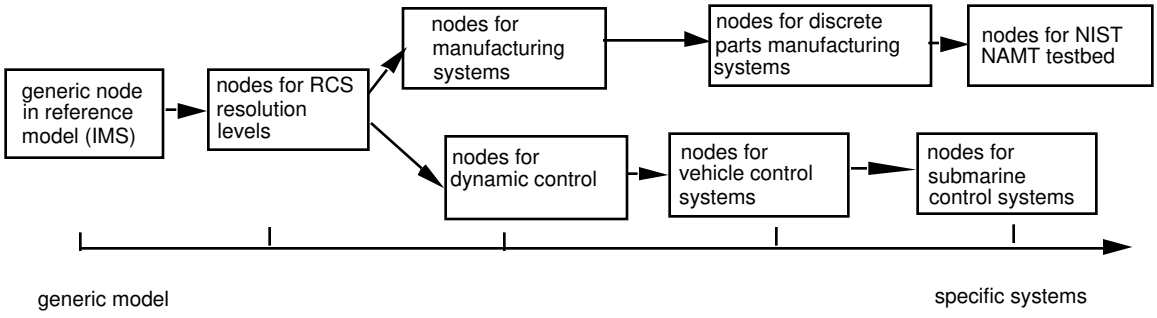

Figure 2: The inheritance in RCS

etc.) as a type of objects. These object types constitute a set of generic templates for RCS development. Applications will then use specific planning or sensory processing algorithms.

5.2 Multiple layers of inheritance

Figure 2 conceptualizes multiple layers of inheritance. RCS control node types (classes), shown as boxes, inherit the desired functionality of the node types to the left, when applicable [HUA95]. The arrowhead in the figure highlights this inheritance relationship. RCS, being a reference model architecture, implies that the properties of the intelligent machine model are inherited by any class of applications that use the architecture. Manufacturing control systems can be considered one class of applications. A discrete parts manufacturing control system inherits properties developed for the generic manufacturing control system RCS. This inheritance relationship can extend to many layers. The mechanism of inheritance may be either specialization or augmentation. The inherited properties can be in the forms of software libraries or data sets.

The authors anticipate that, when the RCS environment is fully developed, a node class will contain common behavioral, modeling, or information features for problem classes such as manufacturing processing or vehicle mobility. This contributes to the richness of the RCS development environment.

5.3 Mapping behavior to a node hierarchy as a means of encapsulation

Encapsulation publicizes the interface of a model and hides the implementation (procedures and data). In a class declaration within an object-oriented model, the interface is in the public portion, and implementation is in the private portion of the model [BOO94]. In RCS, the task structure is integrated with the node hierarchy. Each node has a set of tasks that it is capable of performing. An input command set corresponding to the task set is used by the node's superior to command the node's behavior. In the same manner, the node's behavior results in the generation of sub-commands for its subordinates.

In this way, in RCS, the nodes' behavioral capabilities are encapsulated via the task set that it is capable of processing. A given node within an RCS hierarchy needs to be aware only of its immediate subordinates' interfaces. It does not need to be concerned about how the subtasks the node passes down are carried out or if the subtasks are further decomposed.

6. Assessing the concept of control hierarchy

6.1 The contribution of multiple resolutions

A major distinction between RCS and the prevailing OO methods is the concept of multiple resolutions. One premise of the OO paradigm is for the object models to reflect the physical world. In this sense, large physical entities can require complicated object models. OO paradigms may apply the concept of abstraction to handle system complexity by focusing on the important and ignoring the secondary characteristics of the objects.

To this effect, the authors propose that designers may use the RCS multiple resolution principle as an explicit guideline to systematically determine the significance of the object characteristics. This, in turn, will determine whether to focus or ignore the characteristics. In other words, the designers model the complex physical entities as an object hierarchy. In this hierarchy, the objects handle the characteristics that are of appropriate levels of temporal and spatial resolutions. Information or behavior that is

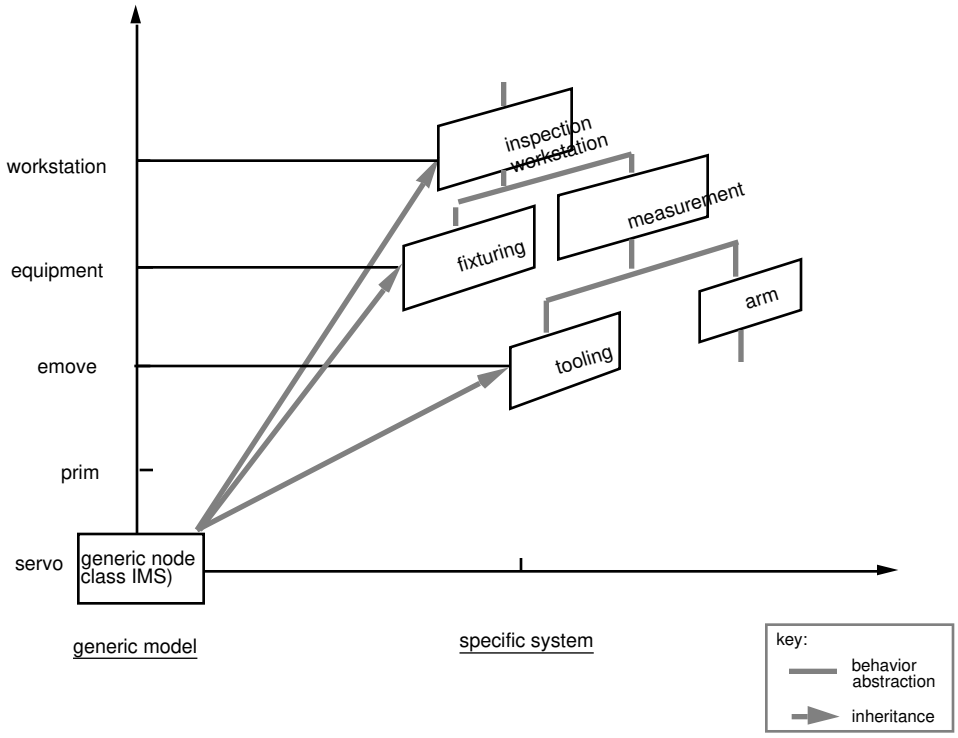

Figure 3: An implementation of RCS-enhanced OO methodology

too detailed or too coarse in detail should be left for the objects at other levels with the appropriate levels of resolution.

6.2 RCS hierarchies as viewed from various OO perspectives

An RCS control hierarchy, such as the one shown in Figure 3, may be compared to various OO concepts, as the following describes:

The RCS superior and subordinate nodes form an association relationship. The links between these two types of nodes are the <send_command> and <report_status> pairs (see Figure 1). In the Figure 3 example, the inspection workstation issues an "inspect_feature" command to the measurement node. This relationship is necessary for the hierarchy to perform tasks.

The RCS command authority concept might be compared to the OO aggregation relationship. One can argue that physically the measurement controller is not a part of the workstation controller. Therefore, this fact would not warrant an aggregation relationship. However, logically, measurement control is a part of the workstation subsystem. This does seem to warrant an aggregation relationship, to this particular workstation and measurement pair, as well as to RCS superior and subordinate pairings in general.

Similarly, from a hierarchical behavior generation viewpoint, a node at a level is concerned with behavior at this level of resolution. One may regard the behavior at the higher levels of resolution as hidden. Therefore, this encapsulation relationship in RCS hierarchies states that, with respect to a node at a given level, all the subordinates are encapsulated.

This explicit command and status relationship helps to focus the analysis and design effort. In designing an automobile system, the initial steps for an RCS developer would be to consider the types of tasks that the driver can ask of the car: to drive to hospital in a snowy day, to haul a trailer through hilly roads, or to enter a race. These behavioral, or command and response, requirements would be used to derive the methods for an OO car class. The derivation process would also yield the required data such as tire size and traction, engine type and size, and transmission type to specifically support the command execution. In this sense, RCS focuses on the intelligence in agents. On the contrary, a data modeling based OO approach might start with describing a car as an aggregation of wheels, axles, body, engine type and size, etc. Each has knowledge of its own capabilities encoded within itself as a "method." While the RCS approach is concordant with this approach and this approach may produce comparable class structures, there may be concern that the data modeling approach may specify information that is with excessive or insufficient level of details for a given task within a certain level of a system. In this regard, the RCS design approach stresses the command hierarchy (and behavior) in a system as opposed to data modeling based hierarchies.

7. The RCS enhanced object-oriented methodology: current implementation

The authors studied the current RCS implementation on the NIST National Advanced Manufacturing Testbed (NAMT) inspection workstation. A base class RCS node was developed and was inherited by all the control nodes in the hierarchy. The control hierarchy implemented has several levels of resolution, as shown in Figure 3. Control nodes populate the entire hierarchy. The nodes are abstracted according to their specific behavior and according to their "natural resolution." The hierarchy emerges through analysis of command (task) flow downward and status reporting flowing upward. The inspection workstation controller receives an "inspect_part" command. The controller issues a "load_part" command for the fixturing control node and an "inspect_feature" command for the measurement node.

Further generalization of this implementation toward a construct as described in Figure 2 may allow the node classes to be inherited and reused by broader classes of applications. For example, the inspection task set may be substituted with one appropriate to machining operations, including "drill_hole." Such work is underway at NIST.

8. Summary

The case study reveals that RCS shares many characteristics with object-oriented methodologies. RCS strengthens the general object-oriented concepts with a focus on behavioral abstraction and multiple resolution. RCS emphasizes that behavior is the central ingredient of an object. This behavior orientation brings forth guidelines which facilitate the analysis and design processes for object-oriented control hierarchies. The multiple resolution concept of RCS brings forth guidelines for organizing the object hierarchies that are designed for complex system intelligent control.

References

ALB96-1 J. S. Albus and A. Meystel, “A Reference Model Architecture for Design and Implementation of Intelligent Control in Large and Complex Systems,” to appear in the International Journal of Intelligent Control and Systems, 1996.

ALB96-2 J. S. Albus and A. Meystel, “Behavior Generation: The Architectural Issues.” *Draft*

ALB91 J. Albus, “Outline for a Theory of Intelligence.” IEEE Transactions on Systems, Man, and Cybernetics, Vol. 21, No. 3, May/June 1991.

BAR84 A. J. Barbera, J. S. Albus, M.L. Fitzgerald, and L.S. Haynes, "RCS: The NBS Real-Time Control System," Robots 8 Conference and Exposition, Detroit, MI, June 1984.

BOO94 G. Booch, Object-Oriented Analysis and Design, The Benjamin/Cummins Publishing Company, Inc., 1994.

COA91 P. Coad, Yourdan, E., Object-Oriented Analysis, Yourdan Press, 1991.

DEW89 S. C. Dewhurst and K. T. Stark, Programming in C++, Prentice Hall Software Series, 1989.

HUA96 H. Huang, “An Architecture and Methodology for Intelligent Control,” IEEE Expert, April 1996.

HUA95 H. Huang, J. Michaloski, N. Tarnoff, M. Nashman, “An Open Architecture Based Framework for Automation and Intelligent System Control,” Proceedings of the IEEE International Industrial Automation and Control Conference, Taipei, Taiwan, May, 1995.

QUI93 R. Quintero, Barbera, A., “A Software Template Approach to Building Complex Large-Scale Intelligent Control Systems,” in Proceedings of the 8th IEEE International Symposium of Intelligent Control, Chicago, Illinois, 1993.

RUM91 J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, W. Lorensen, Object-Oriented Modeling and Design, Prentice-Hall, 1991.

SHL92 S. Shlaer, S. Mellor, Object Lifecycles: Modeling the World in States, Prentice-Hall, 1992.