CONCEPTUAL MODELING WITH THE OBJECT-PROCESS METHODOLOGY IN SOFTWARE ARCHITECTURE*

Hong Liu
David P. Gluch
Embry-Riddle Aeronautical University
600, S. Clyde Morris Blvd
Daytona Beach, Fl 32114
liuho@erau.edu
gluchd@erau.edu

ABSTRACT

The ultimate goal of software design is to transform real world problems into software solutions. Architectural design is the earliest phase of this process. It is a phase in which conceptual modeling plays an important role. In conceptual modeling, designers pay more attention to accurately describing real world problems than making detailed design decisions. Object-Oriented Analysis and Design is the prevailing software development methodology. Using object-orientation for conceptual modeling is often difficult, especially because of the encapsulation of processes within objects. An alternative approach is the Object-Process Methodology (OPM). OPM is a systems engineering approach that, while recognizing the duality of objects and processes, establishes a peer relationship among them. This peer relationship enables conceptual modelers to describe real world problems more naturally. Using OPM and its support tool, designers can develop integrated conceptual models that faithfully capture the characteristics and interactions of real-world entities. This paper briefly introduces OPM and its support tools and uses case studies to show the advantages of OPM for conceptual modeling in software architecture design.

Keywords: Object-Process Methodology, Conceptual Modeling, Software Architecture

INTRODUCTION

An ideal software architectural design should yield a software blueprint that optimizes the often-conflicting functional and nonfunctional requirements of real world problems. The large scale and complexity of modern software projects result in extremely difficult challenges for software architectural designers. Accompanying these challenges are technology advancements that result in new approaches and support tools for the design process. Object-Oriented Analysis and Design (OOA/D) is the prevailing methodology employed in the software industry; and in the form of the UML methodology ([1], [5], [6]) has become the de facto industry standard [4].

While object-orientation has been a successful software design approach, the world is not inherently object-oriented. Michael Jackson makes this point in [14]: "But the idea of an *object* is a programming idea, and it doesn't fit most of the individuals in the world very well." Rather, objects and the processes that modify the states of an object are distinct, i.e. processes are not internalized in objects. Concepts in the real world are described by natural language that contains nouns (objects) and at least one verb (process) in each sentence. Dov Dori recognized the duality of objects and processes and proposed the Object-Process Methodology (OPM [2]) as a systems engineering approach to model real world problems more naturally. In addition to OPM, he also developed the Opcat, tool that supports Object-Process Diagrams (OPD) and an equivalent textual representation, Object-Process Language (OPL). Both OPD and OPL are part of the Methodology.

Transforming problems into software designs, preferably object-oriented for programming efficiency, relies heavily on the ingenuity and experience of designers. Nevertheless, a sound engineering process, plus a systematic methodology and proper tool support can make a significant difference in effectiveness and productivity. The Siemens methodology defined in the book Applied Software Architecture (ASA) [8] is an architectural design approach, drawing heavily from object-orientation.

In this paper, we propose using the OPM and its CASE tool as an effective approach for developing conceptual models, comparable to the conceptual architecture view (model) of the ASA approach. The application of OPM to conceptual modeling of ASA may help designers use natural language to organize their thoughts and may reduce the burden of bridging the gap between real world concepts and object-oriented design constructs.

This paper first defines the role of conceptual models in software architecture and raises some issues associated with current conceptual modeling practices. Next, we introduce OPM, OPD, OPL, and OpCat-the CASE tool for supporting OPM. Then, we address the main objective of paper and present the rationale for applying OPM in conceptual modeling. In

addition, we use case study examples to illustrate the advantages of OPM for conceptual modeling. Finally, we conclude with a summary of our work.

CONCEPTUAL MODELS IN SOFTWARE ARCHITECTURE

This section we first introduce the process and the conceptual view of the ASA approach. Then we discuss some common criteria for conceptual models, recognizing that there is no consensus on these since different domains emphasize different aspects of product quality [11]. Finally, we raise some issues in conceptual modeling that challenge novice and seasoned architectural designers alike.

An Introduction to Conceptual Models in Software Architecture

Separating concerns is a main strategy to control complexity. In order to study software architectures for large, complex industrial-scale systems, the ASA design process involves the construction of four views [8]. Each view deals with a different engineering concern. The views are the conceptual view, module view, execution view and code view. These are notionally developed in this order as design progresses. However, the engineering process is incremental and iterative. This paper addresses the conceptual view and its influence on the other three views. The significance of the conceptual view is described in the following direct quotation:

In a few systems, the conceptual view plays a primary role. The module, execution and code views are defined indirectly via rules or conventions, and the conceptual view is used to specify the software architecture of a system, perhaps with some attributes to guide the mapping of other views [8].

In 1995, Kructhen defined 4 plus 1 views for architectural design. These are the logical view, process view, development view, physical view, and plus-one view [10]. The logical view primarily supports the behavioral requirements and the services the system should provide to users. The plus 1 view embodies a set of scenarios, instances of use cases, showing how the elements of the four views should work together. Although the 4 plus 1 views separate concerns with slightly different perspectives, there are similarities between the 4 plus 1 approach, ASA views, and OPM. The logical view is close to the conceptual view of ASA approach. The plus 1 view is similar to the integrated view of OPM.

One of OPM's contributions is that its syntax and semantics integrate the structural, functional, and behavioral considerations into one diagram that usually includes many hierarchically organized sub-diagrams. Structural considerations address what is in the system. In UML, the structural considerations are included in the class, object, component, and deployment diagrams.

The functional considerations address what the system does. In UML, the functional considerations are captured in use case and sequence diagrams. Behavioral considerations focus on how does the system fulfill its functionalities. In UML, behavior is addressed in activity, statechart, and collaboration diagrams. OPM provides the expressability that enables designers to describe objects, processes, states, and their relations at a variety of desired

granularity levels. The relevant question is: Which engineering concerns should a conceptual model address and what level of detail it should specify?

In the ASA approach, the conceptual view is closest to the application domain and furthest from the target platform. The inputs for constructing a conceptual model include global analysis documents, factors, and issues relating to functional requirements, and technological, business, and organizational constraints. A conceptual model maps functionalities onto abstract components. The interaction of software with its environment and major controlling mechanisms between components is specified in connectors. Specifically, a conceptual view provides answers to the following questions: Does the system fulfill the requirements? Are commercial off-the-shelf components (COTS) to be integrated and how do they interact with the rest of the system? Is domain-specific hardware/software incorporated into the system? Is functionality partitioned into product releases? Does the system support legacy components and future extensions? Are product lines supported [8]?

Criteria for Sound Conceptual Models

We can identify criteria for a sound conceptual model using the definition of a conceptual view and the expectations articulated above.

- The view should be similar to an analogical model [9] to simulate the implementation of the real world problems. Its main purpose is to capture and facilitate analysis of the system in the context of its environment, incorporating key product features, requirements, and essential domain knowledge.
- It should include information about the structural, behavioral, and functional characteristics of the product, not the software solution. It should be distinct from design models to avoid premature design decisions.
- It should specify the major components and their interactions with the environment so that designers can identify conflicting requirements and nonfunctional constraints (e.g. COTS, embedded legacy systems, domain critical hardware etc.).
- It should enable the following traceability checks: each component in a conceptual model can be mapped to an environmental actor, a physical component, a COTS component, a legacy system, a domain concept, or a perceptible environmental action; all fundamental functionalities are included and responsibilities are assigned to proper components; and behaviors of the model can be traced to use case scenarios that match major features of the requirements.
- It should be modeled in a clear hierarchical structure with each level properly abstracted. This should be constructed such that the relationships of upper level components and lower level components are identifiable and traceable.

Some Issues in Conceptual Modeling

Some issues in conceptual modeling include:

- The multiplicity of representations (e.g. multiple UML diagrams) causes traceability problems [7]. Cross inspection of multiple diagrams is tedious and unreliable.
- Components and connector diagrams rely heavily on textual documentation to specify the interactions of the components and the functionalities of the connectors. It can be difficult for designers to obtain intuitive perception from the diagram alone.
- A conceptual model that faithfully represents a system without making premature modularization decisions may help designers to identify and address reusability and maintainability issues in the early phases of development.
- Suppressing the role of processes in real world problems (e.g. encapsulation in objected-oriented requirements specification approaches) can make it awkward for designers to capture the relationships between objects and identify the triggering processes of state transitions.

OBJECT-PROCESS METHODOLOGY AND ITS TOOL SUPPORT

In this section, we introduce the OPM, its support tools, and the graphical and textual representation of OPM models. We also present a simple comparison to well-known OOA/D. An example is used to illustrate the interface of OpCat, one of the OPM CASE tools.

Introduction to Object-Process Methodology (OPM)

Object-Process Design uses three entities as building blocks. They are objects, processes, and states. Objects are things that exist. Processes are things that transform objects by changing their states or creating or consuming objects. Since OOA/D is a well-known technique, one can understand OPM easily by giving a comparison between the two methodologies. OOA/D emphasizes identifying objects and encapsulating processes into objects.

OPM promotes processes as peers of Objects. Therefore, objects may contain processes and processes may contain objects. States are lower level entities in OPM meta-models since states are expressed inside of objects. The scope and focus of OPM are different from that of OOA/D. OOA/D is a software industry practice where as OPM is a systems engineering approach. OOA/D evolved from programming to design and analysis, whereas OPM is a top down representation of a system without the constraints of programming languages.

Object-Process Language (OPL), Diagram (OPD) and Support Tools

Object-Process Diagrams (OPD) and accompanying language (OPL) are used in OPM modeling. OpCat (included in [2]) and Systematica are two CASE tools that support OPD and OPL. OPM combines the ease of use of natural language with the exactness and reduced ambiguity of formalism.

The syntax of OPM is simpler than that of UML in that OPM has fewer entities and entity interaction patterns. Objects, processes, and states are symbolized by rectangles, ellipses, and

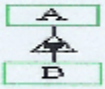
Structural Relation Name		OPD Symbol	OPL Sentence	
Backward	*Forward*		Backward	Forward
Participa-tion	*Aggrega-*tion		B is part of A.	A consists of B.
Characteria-tion	Exhibition		B characterizes A.	A exhibits B
Specializ-ation	Generaliz-ation		B is an A.	A generalizes B.
Instantia-tion	Classifi-cation		B is an instance of A	A classifies B.

Table 1

rounded rectangles, respectively (see figure 1). Besides these three entities, there are two types of links in OPD/OPL. One type is a structure link that captures the relationships between objects or between processes. This type cannot link an object to process. The other type is a procedure link that captures interactions between an object and a process. This type cannot link two objects. Tables 1 and 2 summarize the four fundamental structure links and nine procedure links, respectively [2]. Besides the four fundamental structure links, there are tagged structure links that are identical to that of UML.

Complexity Management in OPM

Separation of concerns in the form of aspect-based decomposition is used as a principal strategy to conquer complexity in OOA/D. This strategy results in the multiplicity of UML models. Using one model, OPM controls the complexity through granularity levels, refinement, and abstraction similar to zooming in and zooming out in a digital map. There are three scaling modes with respect to the three entities of OPM. Visibility scaling zooms in or zooms out of a process; hierarchy scaling unfolds or folds an object, and manifestation scaling expresses or suppresses a state.

An Example Diagram from a Student Registration System

Figure 1 is the *Registering* process zoomed in from the root diagram of a Class Registration System. This diagram provides a glimpse to the features of OpCat, OPD, and OPL and the integration of OPD and OPL (graphics and text) within the tool. OMG employs a similar integration concept, incorporating OCL (Object Constrain Language) as its textual documentation of constraints within UML diagrams [13].

OPM FOR CONCEPTUAL MODELING

The important features of OPM for conceptual modeling are its integrated views, concise formalism interaction patterns, and CASE support tools that combine graphic and textual interfaces. In addition, the integrated view of OPM can relieve designers of the need for cross-model inspections to ensure consistency among multiple UML models and ASA views.

Integrated Views: As noted in the section above, the integrated views of OPM combine the structural, functional, and behavioral characters into one model. In this integration, the responsibility and roles of all entities are clearly specified and self-documented. The three

Type	Name	Semantics	Symbols	Source	Destination
Trans-forming	Consump-tion	The process consumes the object		Object	Process
Trans-forming	Result	The Process generates the object		Process	Object
Trans-forming	Input	The Process changes to an input state		States	Process
Trans-forming	Output	The Process changes to an output state		Process	States
Trans-forming	Effect	The Process changes the object from input state to output state		Object	Process
Enabling	Agent	The human agent enables the process		Object	Process
Enabling	Instrument	The process requires the instrument		Object	Process
Condition	Agent condition	The agent at state enables the process		States	Process
Condition	Condition	The instrument at state enables the process		States	Process

Table 2

building blocks can be identified from their responsibilities presented in requirements documents and their semantic roles in OPM. Key questions relating to identifying the roles of the building blocks in OPM include:

- Process: Which objects instrument and initiate the process and which of the three functionalities does it fulfill: create, destroy, or modify the states of an object?
- Object: What are the states of the object; which processes create, destroy, and modify the object; and which processes does it instrument?

- State: What object does it belong to; which process triggered the change of the state; and what are the source and destination states?

Concise Formalism of Interaction Patterns: OPM provides a template of interaction patterns with four fundamental structural links and nine procedural links (see table 1 & 2). The four structural links mainly specify the relationship between objects. Procedural links give the relationship between objects and processes. Unlike the ASA conceptual diagram, which only shows the existence of an interaction with the environment or other subsystems, the semantics of the OPM interaction links provides insight into the nature of the interactions. For example, in the registration system above, both graphics and the context show that the *Student* is an agent for initiating the *Login* process and the *Login process* creates the *Access Permit* object.

OPM interaction patterns focus on how to naturally and faithfully capture the relationships between entities and environments, rather than making premature decisions about the encapsulation of processes into objects. The OPM interaction patterns (13 in total) are fewer than that of UML. UML has 8 kinds of diagrams (use case, class, component, deployment, statechart, activity, sequence, and collaboration). Each of these kinds has a slightly different set of interactions among the elements that comprise the diagrams [4]. The conciseness of OPM interaction patterns helps to simplify the modeling effort.

Tool Support: The combined textual and graphical interface of OpCat is designed to take advantage of both our natural language abilities and graphically-oriented intuition. An OpCat user can sketch his or her idea in the OPD and then check whether the textual specifications of the OPL match his or her intent. The syntax of OPD and OPL can help focus a designer's thought on relevant issues and provide him or her instruments for creative expression. The zooming feature allows designers to choose the granularity level and comprehend the OPM designs from high-level abstractions down to detailed level presentations.

CASE STUDIES

This section presents cases studies for the use of OPM.

Cruise Control System: Our first example is a conceptual model of a Cruise Control (CC) system. In [12], Mary Shaw specifies the architectural design of a cruise control system using a feedback structure diagram, state transition diagram, and event table. She mentions the idea of an integrated view but does not provide one in [12]. The OPM diagrams in Figures 2 to 5 represent various levels of an integrated conceptual view based on the CC design presented in [12]. Figure 2 is the root diagram. It specifies the top-level processes and states. We distinguish the *controller actuating* from the *controlling process*. The OPL of figure 2 shows that the *Speed Control System* consists of the *Speed Controller* and *Vehicle Speed* objects and the *Speed Controlling* process. This is a very natural sequencing, i.e. the speed controller is controlling the vehicle speed. The driver handles the *Starting* process, which changes the state of the *Engine* from *off* to *on*, etc. By right clicking the *Speed Control System* in the tool and selecting the unfolding feature, we obtain Figure 3. Right clicking the corresponding processes in Figure 3, we zoom in to Figures 4 and 5, the *Controller Actuating* and *Speed Controlling* processes, respectively.

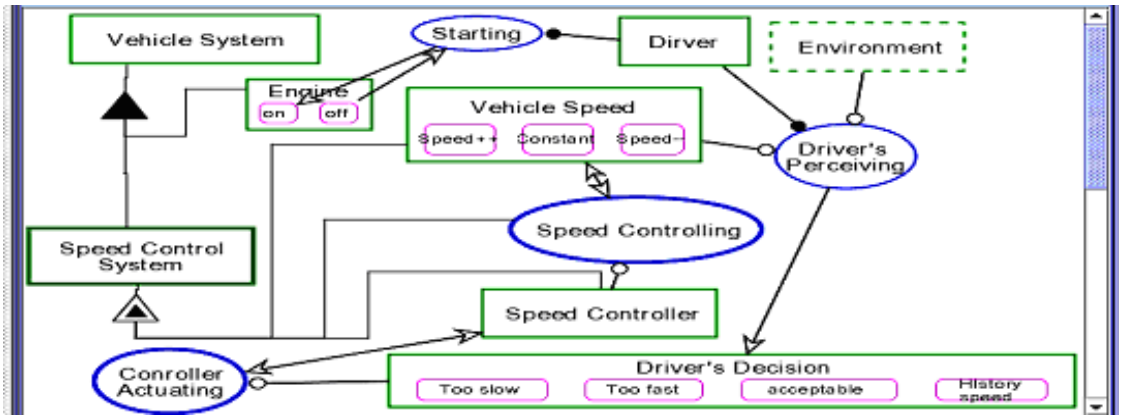


Figure 2

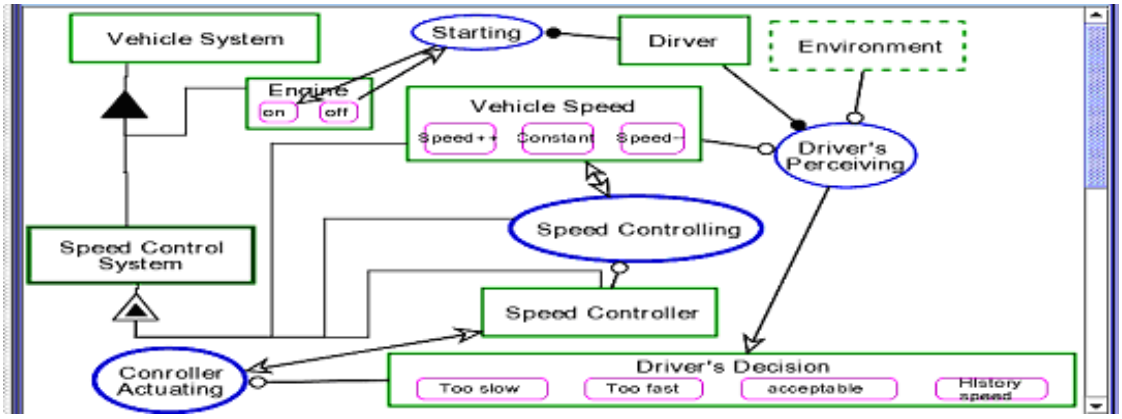


Figure 3

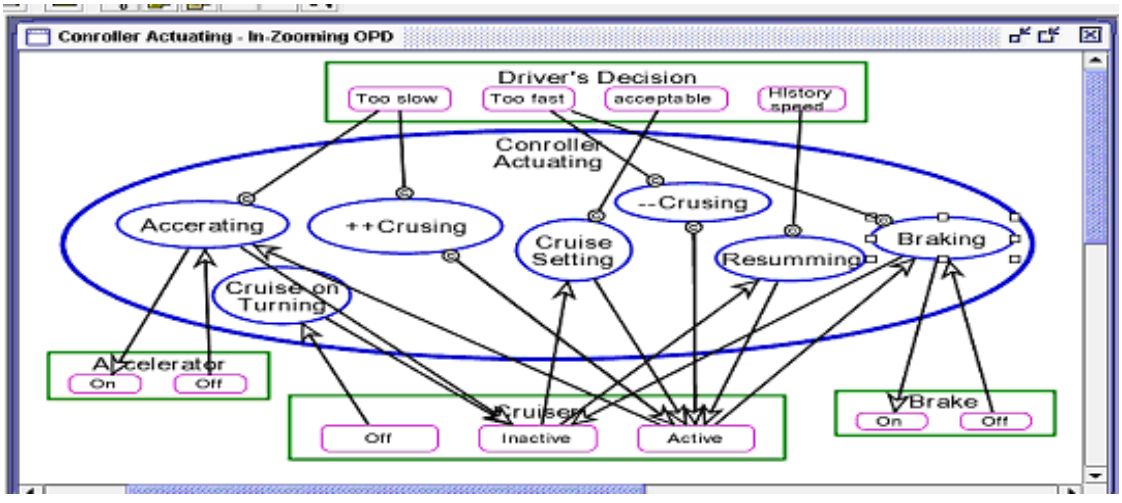


Figure 4

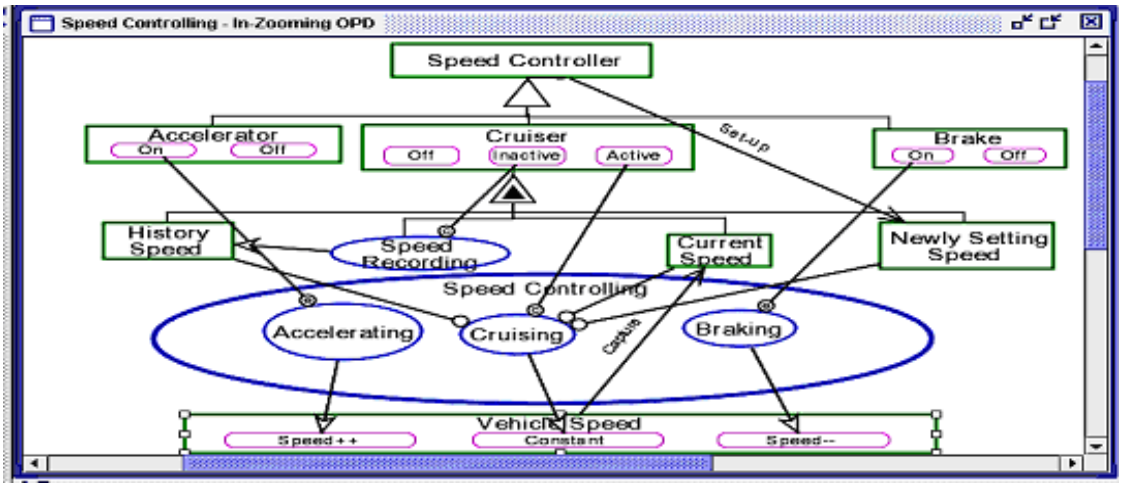


Figure 5

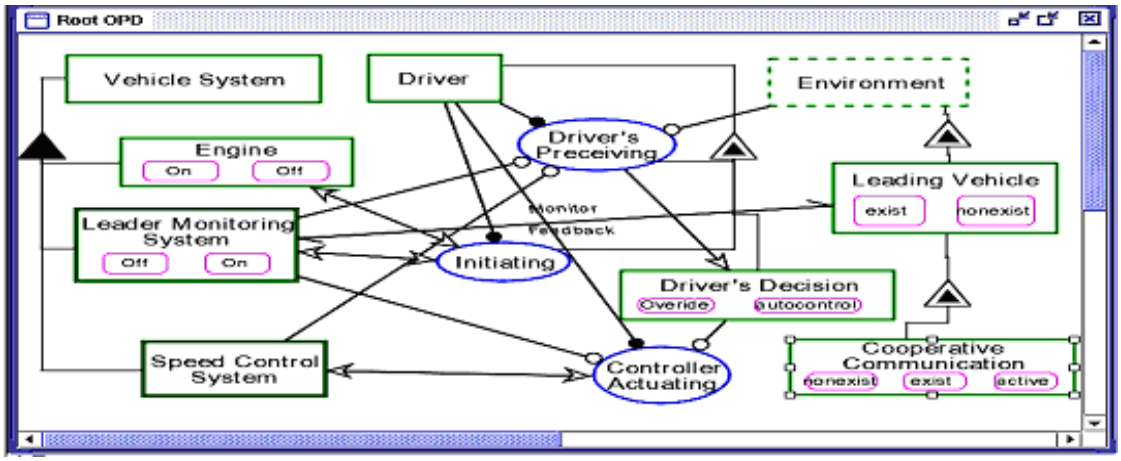


Figure 6

Adaptive Cooperative Cruise Control and Collision Warning Systems: We extend the CC model into an Adaptive Cooperative Cruise Control and Collision Warning System (ACCC), a more complicated cruise control system [3]. Conceptually the ACCC simply adds a monitor system as an alternative to the driver for controlling the CC. Figures 6 - 8 are a subset of sub-diagrams of an OPM conceptual model for the ACCC system. Conceptually, ACCC adds a Forward Collision Warning (*FCW*) radar and wireless communication (*Cooperative FCW*) system to monitor the leading vehicle and uses the information from the leading vehicle to control the speed of a following vehicle. Hence, figure 6 simply adds a *Leader Monitoring System* to the *Vehicle System* and a *Leading Vehicle* to the *Environment*. Figure 7 and 8 further specify the *Leading Monitoring System* and *Communicating* processes. The Speed Controlling Mechanism of the ACCC is similar to the CC system except that the monitoring system can be controlled by the ACCC as well as by the driver.

SUMMARY

The Object-Process Methodology (OPM) is a new systems engineering approach that recognizes the duality of objects and processes in describing real world problems. It provides a highly integrated representation of a system, built-in interaction patterns, and support tools that combine graphical and textual interfaces. OPM's integrated approach contrasts with the multiple views associated with object-oriented techniques. What impact this methodology may bring to architectural modeling within the software industry is not yet clear. This paper presents the results of investigations into the application of OPM to conceptual modeling, using the ASA approach [8] as a comparative benchmark.

ACKNOWLEDGEMENT

We would like to thank the referees for their helpful suggests toward improving this paper.

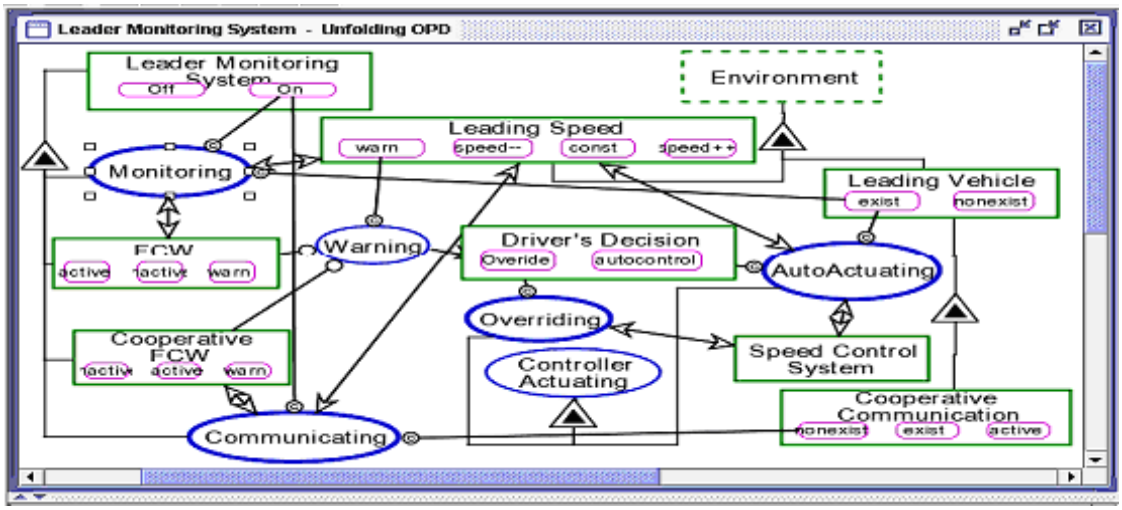


Figure 7

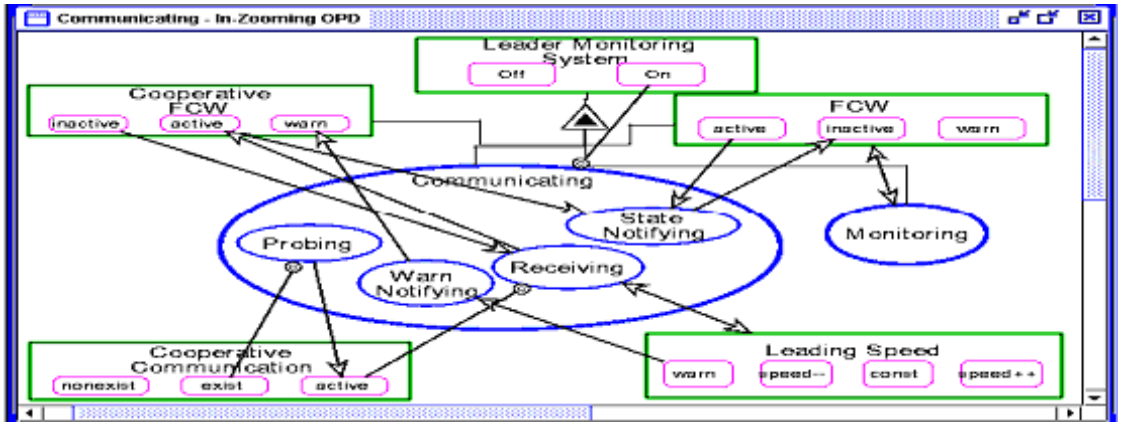


Figure 8

REFERENCES

[1] Grady Booch, Object-Oriented Analysis and Design with Applications, Benjamin & Cummings, 2nd Ed. 1994.

[2] Dov Dori, Object-Process Methodology, Springer, 2002. www.ObjectProcess.org

[3] Anouck Renée Girard, João Borges de Sousa, James A. Misener and J. Karl Hedrick, A "Control Architecture for Integrated Cooperative Cruise Control and Collision Warning Systems", path.berkeley.edu/~anouck/papers/cdc01inv3102.pdf

[4] Fowler, M. UML Distilled, 2nd edition, Addison-Wesley, Reading, MA 1999.

[5] Harel, D. "Statecharts: A Visual Formalism for Complex Systems," Science of Computing 8, pp. 231-274, 1987.

[6] Craig Larman, Applying UML and Patterns, an Introduction to Object-Oriented Analysis and Design, Prentice Hall, 2nd Ed. 2000

[7] B. Hnatkowska, Z. Huzar, J. Magott, "Consistency Checking in UML Models," www.fit.vutbr.cz/events/ism/2001/pdf/hnatkowska.pdf

[8] C. Hofmeister, R. Nord and D. Soni, Applied Software Architecture, Addison-Wesley, 2000.

[9] Jackson, Michael, Problem Frames: analyzing and structuring software development problems, Addison-Wesley, 2001.

[10] Kructhen, P. "The 4 + 1 View Model of Architecture," IEEE Software 12, 6: 42-50, 1995.

[11] Shaw, M. and Garlan, D. Software Architecture: Perspectives on an Emerging Discipline, Prentice Hall, 1996.

[12] Shaw, Mary, "Comparing Architectural Design Styles", IEEE Software Special Issue on Architecture, Vol. 12, Num.6, page 27-41, Nov. 1995

[13] Warmer, J. and Klppe, A. The Object Constrain Language: Modeling with UML, Addison-Wesley, Reading, MA, 1999.

[14] Jackson, M. Software Requirements & Specifications: a lexicon of practices, principles, and prejudices, Addison-Wesley (1995).