By Nathan R. Soderborg, Edward F. Crawley, and Dov Dori

SYSTEM FUNCTION AND ARCHITECTURE: OPM-BASED DEFINITIONS AND OPERATIONAL TEMPLATES

Designing a system's architecture involves creating system models that help the architect transform a concept for achieving some goal into a detailed specification of an architecture that attains that goal. A model should specify the system's building blocks—entities, including objects, their states, and the processes that transform the objects, along with the structural and procedural relationships among them. Minimizing subjectivity and ambiguity in a model requires rigor in conceptually formulating and decomposing a system. However, maintaining such rigor is challenging, as it may conflict with the creative nature of the modeling activity. This challenge can be compounded by arbitrary diagramming techniques and too-liberal use of natural language, which is highly flexible but prone to ambiguity.

This framework for modeling and conceptually representing a system's specification identifies essential objects and processes while minimizing ambiguity in functional and architectural requirements.

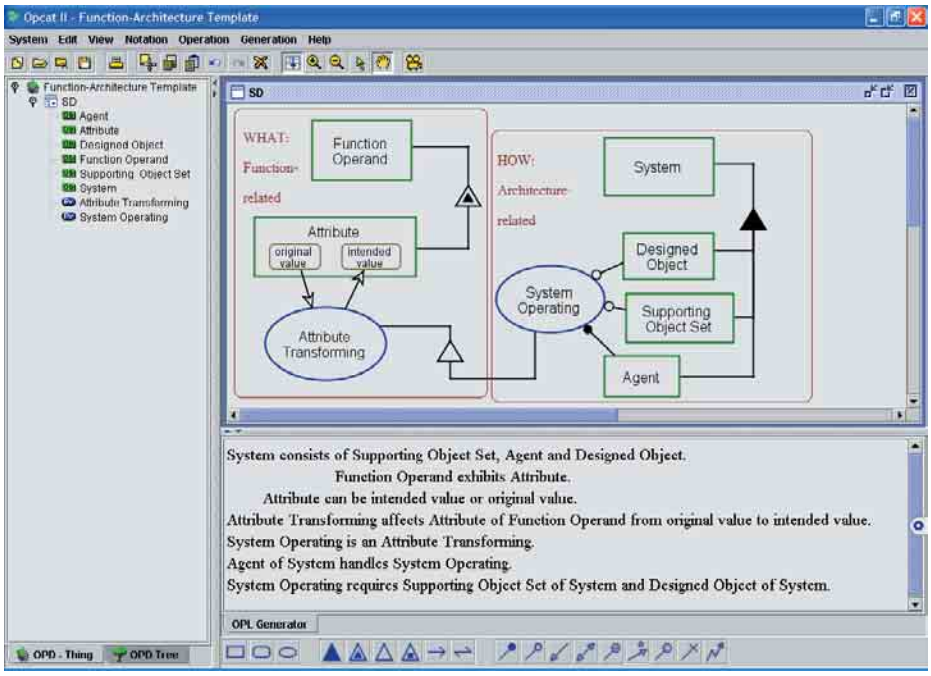

Figure 1. OPCAT screenshot of a generic OPM template for whats and hows, showing the OPD (top) and its corresponding automatically generated OPL paragraph (bottom). Dotted lines separating the What from the How parts of the template and the italic text labels are annotations, not part of the OPD.

One method for introducing rigor into system conceptual development is the *what-how* decomposition, which requires a clear delineation of *what* is desired and *how* this desire is achieved at each level of system detail [7]. Though this method is helpful, its rigor is inevitably compromised without clear rules for formulating and communicating whats and hows. Here, we propose an Object-Process Methodology (OPM)-based framework for modeling systems that enables a rigorous what-how decomposition. We offer OPM templates that provide operational definitions for whats and hows, enabling architects to answer a repeatable, ordered set of questions to define architectures at increasing levels of granularity with minimal ambiguity.

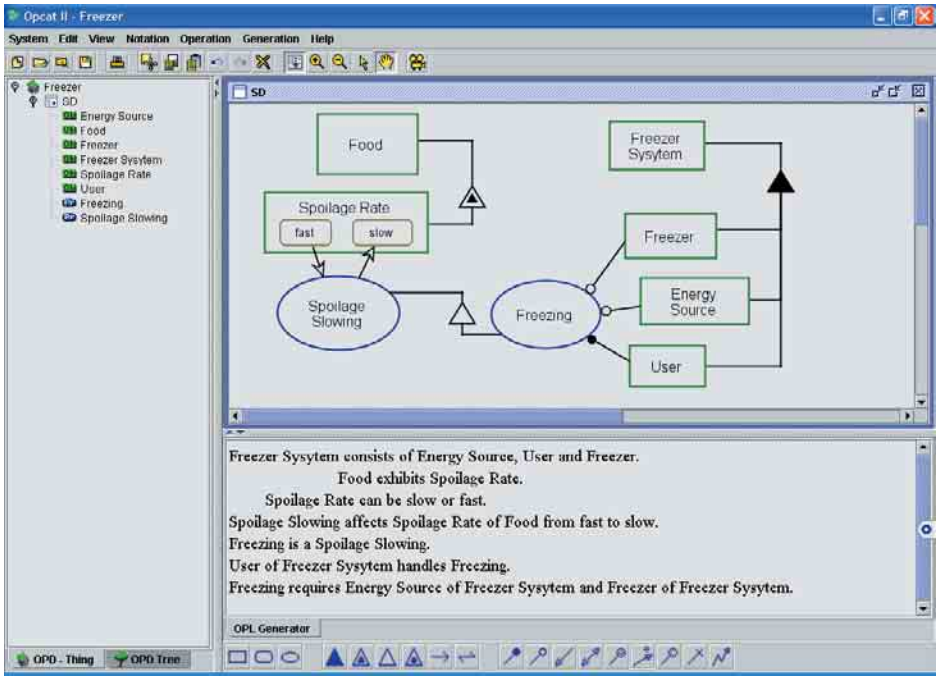

Figure 2. The Freezer System OPD and corresponding OPL paragraph (an instance of the template in Figure 1).

Whats and Hows

What? is a functional, problem-oriented question that refers to an objective, the achievement of which is desired. How? is a design (solution-oriented) question that refers to how the objective is met. Formal system decomposition methods adopting the what-how paradigm include Quality Function Deployment (QFD) [4], which identifies customer needs as the first set of whats [2], and Axiomatic Design [7], which labels whats as "functional requirements" and hows as "design parameters."

Object-process pairs characterize both whats and hows. OPM provides a framework for formally defining whats and hows in terms of objects and processes [2]. The OPM premise is that objects and processes are two equally important classes of things. Objects and processes together faithfully describe a system's function, structure, and behavior in a single, coherent, domain-neutral model. OPM represents systems

in two ways: graphically, through Object-Process Diagrams (OPDs), and textually, through Object-Process Language (OPL), a subset of natural English. Each OPD has a unique textual equivalent expression as an OPL paragraph consisting of OPL sentences. Each OPD and its corresponding OPL paragraph is uniquely and faithfully reconstructable from the other. A set of OPDs or its corresponding OPL script completely specifies a system.

A natural answer to the question "What?" is an object, typically expressed as a noun. A natural answer to the question "How?" is a process, typically expressed as a verb. However, for axiomatic design, the opposite is recommended: "Functional requirements are stated as imperatives, starting with verbs, whereas design parameters are usually stated with nouns" [6]. This apparent contradiction arises naturally through the flexibility of language to use the same words to describe different perspectives. Each is valid but incomplete because a combination of objects and processes is necessary to fully express functional whats and their corresponding hows.

Consider, for example, the development of a system to "provide a comfortable atmosphere in a hot and humid environment" (the what). Full representation of this functional requirement includes an object with an attribute that is changed to a desired value and a process that changes the attribute value. Here, the process is "providing," the object is "atmosphere," and the atmosphere's attribute to be controlled is "comfort level," with values "uncomfortable" and "comfortable." The goal of the system, or its function, is to change the value of the atmosphere's human comfort attribute from "uncomfortable" to "comfortable."

Two possible system architectures can arise from two different answers to the How? design question. One is "move air" (such as by a fan); the other is "cool air" (such as by an air conditioner). For "move air," the process is "moving," the object is "air," the air's attribute is "velocity," and the velocity values might be "zero," "low," "medium," or "high." The architect's objective of changing the human comfort level from uncomfortable to comfortable is achieved in this case by increasing the air velocity from zero to low, medium, or high. For "cool air," the process is "cooling," the object is again "air," but this time the air's controllable attribute is "temperature," and the temperature values are measured in Celsius or Fahrenheit degrees or stated less precisely as "hot," "warm," or "cool." The function (high-level objective) of changing the human comfort level is achieved by changing

OBJECTS AND PROCESSES TOGETHER FAITHFULLY DESCRIBE A SYSTEM'S FUNCTION, STRUCTURE, AND BEHAVIOR IN A SINGLE, COHERENT, DOMAIN-NEUTRAL MODEL.

SUCH PRINCIPLES AND APPROACHES CUT ACROSS DISCIPLINARY BOUNDARIES TO REALIZE THE VISION THAT SOFTWARE SYSTEMS ARE SUBSYSTEMS OF LARGER ENGINEERING SYSTEMS.

the air temperature from hot or warm to cool.

This example is typical of the process required to define whats and hows; their formulation involves an object, an attribute of that object (the control of which is desired), and a process that affects the object by changing the value or state of its attribute. As an architect decomposes a system into further levels of detail, each how becomes a what for the next level down in the system's function hierarchy. For example, "cool air" is the answer to the How? question associated with the "provide comfort" what. At the next level of detail, "cool air" becomes the what, and the associated how question is "How can air be cooled?" The answer might be to condense it, in which case "condense air" becomes the how to the "cool air" what. The process at this point is "condensing," and the object is "air." The relevant air attribute is pressure, and the transformation of air is obtained by changing its pressure value from "low" to "high."

Software engineers might think the what-how paradigm applies only to physical systems, but they should realize it also applies to software and information systems. For example, a "sort list" what can be achieved by repeating a "find minimum" how.

System theory principles (including the what-how paradigm) apply to software-intensive systems in a way that is not principally different from any other system. Moreover, with the ultimate objective of improving human conditions, systems increasingly combine mechanical, electrical, electronic, and optical hardware with software, further blurring the distinction between hardware and software systems. In our example of "providing a comfortable atmosphere," this trend might be achieved by an architect conceiving a sophisticated real-time software-driven system to monitor and control the combination of air temperature and humidity. In general, this trend underscores the importance of defining general principles and developing generic approaches to embedded system architecting. Such principles and approaches cut across disciplinary boundaries to realize the vision that software systems are subsystems of larger engineering systems.

Whats convey function. OPM defines function as an attribute of an object that describes what the object does, the phenomena it exhibits, the service it supports, or what it is used for. This definition focuses on what and is not concerned with how. Defining function in this way enables a clear distinction between the often blurred notions of a system's function and its dynamics. Function concerns the intended effect of the system's operation on the ben-

eficiary, user, and environment; dynamics concern how an object operates to achieve the effect. Function is "process with intent," and since intent is a result of human thought, function is subjective; it is in the eye of the beholder.

In OPM, no process exists unless it transforms at least one object. A transformation includes generation of a new object, consumption of an existing object, or a change in an object's value or state. This specification of a function requires an object, the transformation of which fulfills intent. The object is known as a function operand, or in more generic OPM terms, a transformee, emphasizing its transformation by a process.

Hows convey architecture. Architecture can be defined as "the scheme by which the functional elements are arranged into physical chunks and by which the chunks interact" [8]. This definition identifies objects (physical chunks) and processes (interactions) as constituents of a system's architecture. This definition aligns with an OPM definition of system architecture that clearly recognizes its static and dynamic aspects. System architecture is the overall system's structure-behavior combination, enabling it to attain its function while embodying the architect's concept [2]. The OPM definition focuses on how and is not concerned with what. The dynamic elements of architecture are the operational elements of a system, represented in OPM by processes. The static aspect of architecture is structure, represented in OPM by objects and their structural relationships.

Generic OPM Function and Architecture Template

Figure 1 presents a standard OPM template for a system's what-how decomposition in terms of function and architecture. It formalizes the idea that whats and hows both include processes and objects. Showing the template as an OPD and a corresponding OPL script, the figure was generated using the Object-process CASE Tool, or OPCAT, a Technion-developed integrated systems engineering software environment supporting OPM-based development [1].

The function-related part of the template includes two objects: Operand and Attribute. (Following the OPL convention, object and process names are capitalized.) Attribute can take one of two values: Original Value and Intended Value. (In general, an object can take any number of values; only two are identified in the template for the sake of simplicity.) Intent is made explicit by identifying the desired change of values in Attribute from Original Value to Intended Value through the Attribute Transforming process. Thus, the template captures a process with intent, namely, a function.

In the architecture-related section of the template, the process is System Operating. The term System is applied broadly to include:

Agents. The humans operating or controlling the system;
Designed Object. The object designed by the architect to act within the system to fulfill the function; and
Supporting Objects. Additional objects in the system's environment (not designed by the architect) required for the system to fulfill the function.

The connection between the function and the architecture sections of the template is the specialization link (white triangle) connecting System Operating to Attribute Transforming. This link indicates that the process (operation) of the system being designed is a specialization of the desired attribute transformation. It corresponds to the final OPL sentence: "System Operating is Attribute Transforming," which communicates the architect's concept of how the system operates through its architecture to achieve the desired function (what) of attribute transforming.

What?		**How?**	
What result is desired?		**How** is the desired result achieved?	
Problem: Specify a **Function** A value-providing operand-result combination		**Solution:** Devise an **Architecture** A function-providing structure-behavior combination	
Object	**Process**	**Process**	**Object**
What should be transformed?	**What** transformation should occur?	**How** does the system operate?	**How** is the system structured?
The transformee (operand): the object whose transformation benefits the beneficiary.	The transformation: creation, destruction, or change in state/ attribute value experienced by the transformee.	System behavior: the way the system causes the transformation.	System structure (form): the objects and relations among them comprising the system.

Relations between Whats and Hows (in OPM terms).

Function and architecture questions. OPM deliberately aligns function with the question "What result is desired?" and architecture with the question "How is the desired result achieved?" Describing a particular function requires an object-process "operand-result" combination, and describing architecture requires an object-process "structure-behavior" combination. The table here organizes these conclusions into the categories of what and how.

The top two rows of the table group our conclusions in plain English. The bottom three rows use OPM concepts and terminology to form more rigorous descriptions. The table also shows how the initial what and how questions can be subdivided into more specific object- and process-related questions. The four sub-questions and their answers elicit the basic information required by the system's architect for a good what-how decomposition of a system.

Consider the hypothetical development of a freezer, the kind commonly found in a kitchen. The system architect begins by asking and answering the questions in the table, including:

What should be transformed? The customer or user needs long-term preservation of food; Food should be transformed.
What transformation should occur? (Food) Preserving should occur.

The template requires explicit identification of the desired change of state in Food. This change is to slow its spoilage rate, so Spoilage Rate is identified and defined as the attribute of Food that must be affected. The specific change required in the system is to decrease this rate from fast to slow, stating values that can be specified quantitatively, if desired. Continuing, the architect asks:

How does the system operate? By Freezing Food; this is the chosen process for slowing Spoilage Rate.
How is the system structured? The architect envisions a Freezer comprised of a Cabinet with a built-in Cooling System. People must place or remove food in the cabinet, so the Freezer System includes a human agent, or Operator.

Figure 2 applies the generic OPM template to the work on the Freezer system completed so far. The architect has defined function and begun specifying architecture. "Changing the Spoilage Rate of Food from fast to slow" is a structured way of stating the initial what. Freezing using a Freezer is the corresponding architectural concept, or how, for achieving the what. Further system decomposition requires the architect to identify next-level whats that define Freezing using a Freezer and developing architectures that fulfill each of them.

Conclusion

OPM facilitates a graphical and natural language system specification identifying essential objects and processes at each level of decomposition while minimizing ambiguity in functional and architectural requirements. Adopting such orderly architecting practice will improve the quality of resulting systems of all kinds, including software-intensive systems and embedded systems combining various hardware types with real-time performance software requirements. C

REFERENCES

1. Dori, D., Reinhartz-Berger, I., and Sturm, A. OPCAT: A bimodal CASE tool for object-process-based system development. In *Proceedings of the 5th International IEEE/ACM Conference on Enterprise Information Systems* (École Supérieure d' Électronique de l' Ouest, Angers, France, Apr. 23–26, 2003; see www.ObjectProcess.org.
2. Dori, D. *Object-Process Methodology: A Holistic Systems Paradigm.* Springer-Verlag, Berlin, 2002; see www.ObjectProcess.orgwww.sightcode.com.
3. Hauser, J. and Clausing, D. The house of quality. *Harv. Bus. Rev. 66,* 3 (May-June 1988), 63–73.
4. Mizuno, S. and Akao, Y. *QFD: The Customer-Driven Approach to Quality Planning and Deployment.* Productivity Press, Portland, OR, 1994.
5. Soderborg, N. *Representing Systems Through Object-Process Methodology and Axiomatic Design.* Master's Thesis, System Design and Management Program, Massachusetts Institute of Technology, Cambridge, MA, 2002.
6. Suh, N. *Axiomatic Design: Advances and Applications.* Oxford University Press, New York, 2001.
7. Suh, N. *The Principles of Design.* Oxford University Press, New York, 1990.
8. Ulrich, K. and Eppinger, S. *Product Design and Development, 2nd Ed.* McGraw-Hill, Boston, MA, 2000.

NATHAN R. SODERBORG (nsoderbo@ford.com) is a Design for Six Sigma Master Black Belt in Product Development Operations at Ford Motor Company, Dearborn, MI.
EDWARD F. CRAWLEY (crawley@mit.edu) is the executive director of the Cambridge MIT Institute and professor of aeronautics and astronautics at MIT, Cambridge, MA.
DOV DORI (dori@ie.technion.ac.il, dori@mit.edu) is an associate professor of information systems engineering at the Faculty of Industrial Engineering and Management, Technion, Israel Institute of Technology, Haifa, and a research affiliate at MIT, Cambridge, MA.