Chapter 5
Object-Process Methodology for Structure-Behavior Co-Design

Dov Dori

Abstract Function, structure, and behavior are the three major facets of any system. Structure and behavior are two inseparable system aspects, as no system can be faithfully modeled without considering both in tandem. Object-Process Methodology (OPM) is a systems paradigm and language that combines structure-behavior co-design requirements with cognitive considerations. Based on formal mathematical foundations of graph grammars and a subset of natural language, OPM caters to human intuition in a bimodal way via graphics and auto-generated natural language text. In a nutshell, OPM processes transform objects by creating them, consuming them, or changing their states. The concurrent representation of structure and behavior in the same, single diagram type is balanced, creating synergy whereby each aspect helps understanding the other. This chapter defines and demonstrates the principles and elements of OPM, showing its benefits in facilitating structure-behavior co-design and achieving formal, semantically-sound, and humanly accessible conceptual models of complex systems in a host of domains and at virtually any complexity level.

5.1 The Cognitive Assumptions and OPM's Design

Text and graphics are two complementary modalities that our brains process interchangeably. Conceptual modeling, which is recognized as a critical step in architecting and designing systems, is an intellectual activity that can greatly benefit from concurrent utilization of the verbal and visual channels of human cognition. A conceptual modeling framework that employs graphics and text can alleviate cognitive loads (Dori, 2008). OPM is a bimodal

Dov Dori
Technion, Israel Institute of Technology and Massachusetts Institute of Technology e-mail: dori@ie.technion.ac.il

graphics-text conceptual modeling framework that caters to these human cognitive needs. This section argues for the value of the OPM holistic approach in addressing the dual-channel, limited channel capacity, and active processing assumptions. Bimodality, complexity management via hierarchical decomposition, and animated simulation respectively address these cognitive needs.

5.1.1 Mayers Three Cognitive Assumptions

Humans assimilate data and information, converting them into meaningful knowledge and understanding of systems via simultaneously use of words and pictures. During eons of human evolution, the human brain has been trained to capture and analyze images, so it can escape predators and capture food. In contrast, processing of spoken words, let alone text, is a product of a relative very recent glimpse in the history of mankind. As our brains are hard-wired to process imagery, graphics appeal to the brain more immediately than words. However, words can express ideas and assertions that are way too complex or even impossible to express graphically (try graphing this sentence to get a feeling for the validity of this claim...). So while a picture is worth a thousand words, as the saying goes, there are cases where a word, or a sentence, is worth a thousand pictures. A problem with the richness of natural languages is the potential ambiguity that arises from their use. This certainly does not imply that pictures cannot be ambiguous as well, but graphic ambiguity can be greatly reduced, or even eliminated, by assigning formal semantics to pictorial symbols of things and relations among them. Since diagrams usually have fewer interpretations than free text, they are more tractable than unconstrained textual notations.

Mayer [May03] found that when corresponding words and pictures are presented near each other, learners can better hold corresponding words and pictures in working memory at the same time, enabling the integration of visual and verbal models. A main contribution of diagrams may be that they reduce the cognitive load of assigning abstract data to appropriate spatial and temporal dimensions. For example, Glenberg and Langston [GL92] found that where information about temporal ordering is only implicit in text, a flow diagram reduces errors in answering questions about that ordering.

Mayer [May03] and Mayer and Moreno [MM03] proposed a theory of multimedia learning that is based on the following three main research-supported cognitive assumptions.

1. Dual-channel – humans possess separate systems for processing visual and verbal representations [CP91, Bad92].
2. Limited capacity – the amount of processing that can take place within each information processing channel is extremely limited [Mil56, CS91, Bad92].

3. Active processing – meaningful learning occurs during active cognitive processing, paying attention to words and pictures, mentally organizing and integrating them into coherent representations. The active processing assumption is a manifestation of the constructivist theory in education, which puts the construction of knowledge by ones own mind as the centrepiece of the educational effort [vG87]. According to this theory, in order for learning to be meaningful, learners must engage in constructing their own knowledge.

5.1.2 Meeting the Verbal-Visual Challenge

As the literature suggests, there is great value in designing a modeling approach and supporting tool that meet the challenges posed by the three cognitive assumptions. While Mayer and Moreno [MM03] used these assumptions to suggest ways to reduce cognitive overload while designing multimedia instruction, the same assumptions can provide a basis for designing an effective conceptual modeling framework. Indeed, conceptual modeling can be viewed primarily as the active cognitive effort of concurrent diagramming and verbalization of one's thoughts. The resulting diagrams and text together constitute the system's model. A model that is based on a compact set of the most primitive and generic elements is general enough to be applicable to a host of domains and simple enough to express the most complex systems. A sufficiently expressive model can be simulated for detecting design-level errors, reasoning, predicting, and effectively communicate ones design to other stakeholders.

Such a modeling environment would help our brains to take advantage of the verbal and visual channels and to relieve cognitive loads while actively designing, modeling, and communicating complex systems to stakeholders. These were key motivations in the design of OPM—Object-Process Methodology [Dor02]. The OPM modeling environment implementation by OPCAT[1] [DRBS03] embodies these assumptions. Stateful objects—things that exist at some state, and processes—things that transform objects by changing their state or by creating or destroying them, are the generic building blocks of OPM. Structural and procedural links express static and dynamic relations among entities (objects, object states, and processes) in the system, and a number of refinement/abstraction mechanisms are built into OPM for complexity management.

[1] An academic individual version of OPCAT for modeling small systems can be obtained from opcat.com.

5.1.3 Dual-Channel Processing and the Bimodality of OPM

Considering the dual-channel assumption, an effective use of the brain is to simultaneously engage the visual and the verbal channels (and hence probably also the two brain hemispheres) for conveying ideas regarding the system's architecture. Indeed, OPM represents knowledge about the system's structure and behavior both pictorially and verbally in a single unifying model. When the user expresses a piece of knowledge in one modality, either graphics or text, the complementary one is automatically updated so the two remain coherent at all times.

In order to show how the cognitive assumptions have been accounted for, we follow a stepwise example of bread baking. Figure 5.1 depicts OPCAT's graphic user interface, which displays simultaneously the graphic (top) and textual (bottom) modalities for exploiting humans' dual channel processing. The top pane presents the model graphically in an Object-Process Diagram—OPD, while the one below it lists the same model textually in Object-Process Language—OPL. OPCAT recognizes OPD constructs (symbol patterns) and generates their OPL textual counterparts. OPL is a subset of natural English. Each OPD construct has a textual OPL equivalent sentence or phrase. For example, **Baking**, the central system's process, is the blue ellipse in Figure 5.1. The remaining five things are objects (the rectangles) which enable or are transformed by **Baking**. **Baker** and **Equipment** are the enablers of **Baking**, while **Ingredients Set**, **Energy**, and **Bread** are its *transformees*—the objects that were transformed by **Baking**. As the direction of the arrows indicate, **Ingredients Set** and textbfEnergy are the *consumees*—they are consumed by **Baking**, while is the resultee—the object created as a result of **Baking**. As soon as the modeler starts depicting and joining things on the graphics screen, OPL sentences start being created in response to these inputs. They accumulate in the OPL pane at the bottom of Figure 5.1, creating the corresponding OPL paragraph, which tells in text the exact same story that the OPD does graphically.

As the example shows, the OPL syntax is designed to generate sentences in plain natural, albeit restricted English, with sentences like "**Baking yields Bread.**" This sentence is the bottom line in Figure 5.1. An English subset, OPL is accessible to non-technical stakeholders, and other languages can serve as the target OPL. Unlike programming languages, OPL names can be phrases like **Ingredients Set**.

To enhance the text-graphics linkage, the text colors of the process and object names in the OPL match their colors in the OPD. Since graphics is more immediately amenable to cognitive processing than text, modelers favor modeling the system graphically in the OPD pane, while the textual interpretation is continuously updated in the OPL pane.

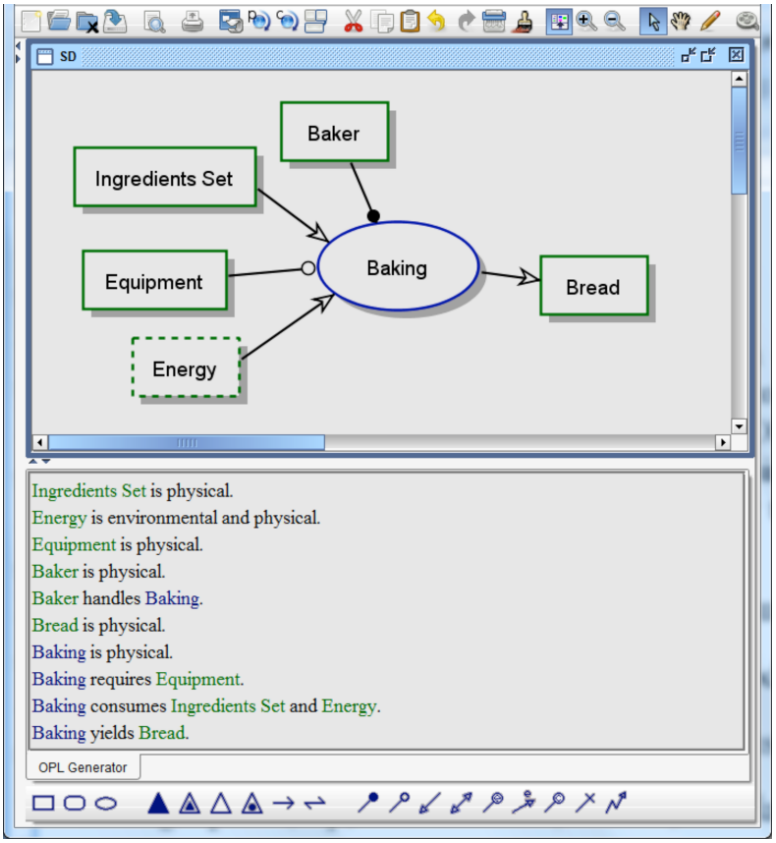

Fig. 5.1 The GUI of OPCAT showing the System Diagram (SD, the top-level diagram) of the Baking system. Top: Object-Process Diagram (OPD). Bottom: The corresponding, automatically-generated Object-Process Language (OPL) paragraph.

The OPL sentences that are constructed or modified automatically in response to linking graphic symbols on the screen provide the modeler and her/his audience with immediate feedback. This real-time human-like response "tells" the modeler what the modeling environment "thinks" he or she meant to express in the most recent graphic editing operation. When the text does not match the intention of the modeler, a corrective action can be taken promptly. Such immediate feedback is indispensable in spotting and correcting logical design errors at an early stage of the system lifecycle, before they had a chance to propagate and incur costly damage. Any correction of the graphics changes the OPL script, and changes can be applied iteratively until a result that is satisfactory to all the stakeholders from both the customer and the supplier sides is obtained. While generating text from graphics is the prevalent working mode, OPCAT can also generate graphics from text.

The System Diagram (SD, the top-level OPD), is constructed such that it contains a central process, which is the one that carries out the main function of the system, the one which delivers the main value to the beneficiary, for whom the system is built. In our case, **Baking** is the process that provides the value—the **Bread**. This is an example of the application of the ***Function as a seed*** OPM principle, which states that *modeling of any system starts with specifying the function of the system* as the main process in SD. Then, objects involved as enablers or transformees are added and both the function and the objects are refined, as described in the sequel. In this system, all the things—objects and processes—are physical, denoted by their shading. **Energy** is also environmental—it is not part of the system as it is supplied by an external source but is consumed by the system. This is denoted by the dashed contour of **Energy**.

5.1.4 Limited Capacity and the Refinement Mechanisms of OPM

Figure 5.1 is the System Diagram, the bird eye's view model of the Baking system. This OPD already contains six entities and five links, approaching the limit of the human capacity determined by the "magic number seven plus or minus two" of Miller (1956). However, we have not even started to specify the subprocesses comprising the **Baking** process or the members (parts) of **Ingredients Set**. To cater to humans' limited capacity, OPM advocates keeping each OPD simple enough to enable the diagram reader to quickly grasp the essence of the system by inspecting the OPDs without being intimidated by overly complicated layout. Overloading the SD with more artifacts will put its comprehension at risk, so showing additional details is deferred to lower-level OPDs, which can be more detailed, as the reader is already familiar with some of the things in them from upper-level OPDs.

To manage the system's inherent complexity, when an OPD approaches humans' "comprehensibility limit", the model is refined. Refinement in OPD entails primarily the application of the in-zooming refinement mechanisms on a process. Figure 5.2 shows a new OPD which resulted from zooming into the **Baking** process in Figure 5.1.

This OPD, which is automatically given the name SD1 (**Baking in-zoomed**), elaborates on SD in several regards. First, **Baking** is inflated, showing within it the four subprocesses **Mixing**, **Forming**, **Heating**, and **Cooling**, as well as the interim objects **Dough** and **Loaf**. **Bread** is created in the initial state **hot**, and after **Cooling** ends, it is **edible**.

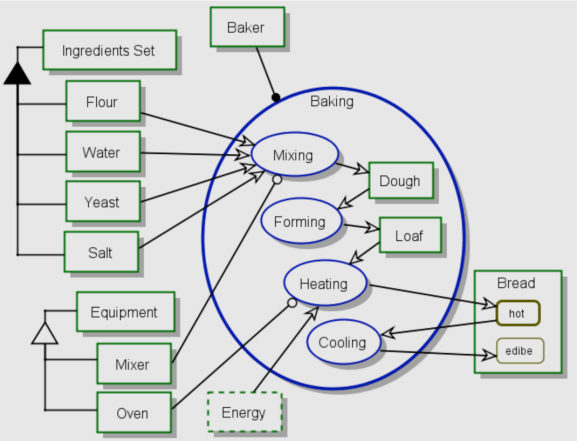

Fig. 5.2 The new OPD which resulted from zooming into the **Baking** process in Figure 5.1.

5.1.5 Active Processing and the Animated Simulation of OPM

The active processing assumption is tacitly accounted for in that each and every modeling step requires complete engagement of the user—the system architect or modeler who carries out the conceptual modeling activity. While modeling, the modeler is active in creating the model, putting and rearranging elements—entities and links—on the screen. While engaged in modeling, the modeler can inspect the OPL textual interpretation that is continuously created in response to each new graphic input. At times, he or she needs to rearrange the graphic layout to make it more comprehensible by such actions as grouping entities and moving links to avoid crossings. When the current OPD becomes too busy it is approaching the limited channel capacity, in which case a new OPD needs to be created via in-zooming or unfolding.

Another aspect of active processing that is unique to OPM is its animated simulation. Humans have been observed to mentally animate mechanical diagrams in order to understand them. Using a gaze tracking procedure, Hegarty [Heg92] found that inferences were made about a diagram of ropes and pulleys by imagining the motion of the rope along a causal chain. Similarly, an active processing aspect of OPCAT is its ability to simulate the system by animating it. The animation enables the modeler to simulate the system and see it "in action" at each point in time during the design. Like a program debugger, the modeler can carry out "design-time debugging" by running the animation stepwise or continuously, back and forth, inserting breakpoints where necessary.

Figure 5.3 is a snapshot of the animated simulation of the **Baking** system, showing it at the point in time when **Heating** just ended, yielding Bread at

its hot state. Currently, as the red dots to and from **Cooling** indicate, **Cooling** is happening, as indicated by its dark blue filling. The timeline within an in-zoomed process is from top down, so **Cooling** is the last subprocess of **Baking**, which is therefore light blue. Objects in green exist at this time point, while white ones (like **Ingredients Set**) are already consumed (or not yet created, but in Figure 5.3 no such objects exist). The active participation of the modeler in inspecting the system behavior and advancing it step-by-step has proven highly valuable in communicating action and pinpointing logical design errors, which are corrected early on, saving precious time and avoiding costly troubles downstream.

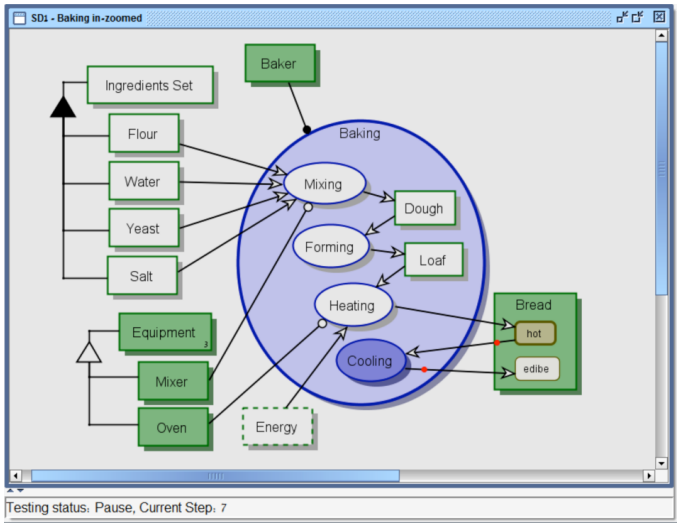

Fig. 5.3 The new OPD which resulted from zooming into the **Baking** process in Figure 5.1.

The ability to animate the system in a simple and understandable manner is yet another benefit of the structure-behavior integration of the OPM model in one type of diagram—the OPD. Splitting the single model into several structural views and several other behavioral views would unnecessarily complicate and obscure the model, preventing it from being amenable to such clear, eye-opening animated simulation.

Having introduced OPM via this simple baking system example, we now define and discuss basic concepts underlying OPM as both a language and a methodology.

5.2 Function, Structure, and Behavior: The Three Major System Aspects

All systems are characterized by three major aspects: function, structure, and behavior. The **function** of an artificial system is its value-providing process, as perceived by the beneficiary, i.e., the person or group of people who gain value from using the system. For example, the function of the organization called hospital is patients' *health level improving*. Each patient is a beneficiary of this system, the customer may be a government or a private entity, and the medical staff constitutes the group of users.

Function, structure, and behavior are the three main aspects that systems exhibit. Function is the top-level utility that the system provides its beneficiaries who use it or are affected by it, either directly or indirectly. The systems function is enabled by its architecture—the combination of structure and behavior. The systems architecture is what enables it to function so as to benefit its users.

Most interesting, useful, and challenging systems are those in which structure and behavior are highly intertwined and hard to separate. For example, in a manufacturing system, the manufacturing process cannot be contemplated in isolation from its inputs—the raw materials, the model, machines, and operators—and its output—the resulting product. The inputs and the output are objects, some of which are transformed by the manufacturing process, while others just enable it. Due to the intimate relation between structure and behavior, it only makes sense to model them concurrently rather than try to construct separate models for structure and behavior, which is the common practice of current modeling languages like UML and SysML. The observation that there is great benefit in concurrently modeling the systems structure and behavior in a single model is a major principle of Object-process Methodology—OPM.

Structure of a system is its form—the assembly of its physical and logical components along with the persistent, long-lasting relations among them. Structure is the static, time-independent aspect of the system. **Behavior** of the system is its varying, time-dependent aspect, its dynamics—the way the system changes over time by transforming objects. In this context, transforming means creating (generating, yielding) a new object, consuming (destructing, eliminating) an existing object, or changing the state of an existing object.

With the understanding of what structure and behavior are, we can define a system's architecture.

Architecture *of a system is the combination of the system's structure and behavior which enables it to perform its* ***function****.*

Following this definition, it becomes crystal clear why co-design of the systems structure and behavior is imperative: They go hand in hand, as a certain structure provides for a corresponding set of system behaviors, and this, in turn, is what enables the system to function and provide value. Therefore, any attempt to separate the design of a system, and hence its conceptual modeling, into distinct structure and behavior models is bound to hamper the effort to get close to an optimal design. One cannot design the system to behave in a certain way and execute its anticipated function unless the ensemble of its interacting parts of the system—its structure—is such that the expected behavior is made possible and deliver the desired value to the beneficiary.

It might be interesting to compare our definition of architecture to the one used by the U.S. DoD Architecture Framework (DoDAF 2007), which is based on IEEE STD 610.12:

> ***Architecture****: the structure of components, their relationships, and the principles and guidelines governing their design and evolution over time.*

The common element in both definitions is the system's structure. However, the DoDAF definition lacks the integration of the structure with the behavior to provide the function. On the other hand, the DoDAF definition includes "the principles and guidelines governing the design and evolution of the system's component over time". However, these do not seem to be part of the system's architecture. Rather, principles and guidelines govern the architecting *process*, which culminates in the system's architecture.

5.2.1 Function vs. Behavior

The above definitions lead to the conclusion that the function of a system is identical with its top-level process. Moreover, the architecture of the system, namely its structure-behavior combination, is what enables the system to execute its top-level process, thereby to perform its function and deliver value to its beneficiary.

The value of the function to the beneficiary is often implicit; it is expressed in process terms, which emphasize what happens, rather than the *purpose* for which the top-level process happens. This implicit function statement can explain why the function of a system is often confused with the behavior or dynamics of the system. However, it is critical to clearly and unambiguously distinguish between function and behavior: Function is the value of the system to its beneficiaries. The function is provided by operating the system, which, due to its architecture—structure-behavior combination—functions to attain

this value. Function explicitly coincides with behavior only at the top-most level. At lower levels, behaviors manifested by subprocesses indirectly serve the function. For example, the electric motor of a water pump in an internal combustion car functions to circulate water, which in turn cools the engine, which in turn drives the car. The function of the car is driving, but processes at various levels of granularity are required to make this happen.

This distinction between function and behavior is of utmost importance since in many cases a system's function can be achieved by different architectures, i.e., different combinations of processes (system behavior) and objects (system structure). Consider, for example, a system for enabling humans to cross a river with their vehicles. Two obvious architectures are ferry and bridge. While the two systems' function and top-level process—river crossing—are identical, they differ dramatically in their structure and behavior. Similarly, a time keeping system can be a mechanical clock, an electronic watch, or a sundial, to name a few possible system architectures, each with its set of "ilities" (availability, maintainability, precision...). Failure to recognize this difference between function and behavior may lead to a premature choice of a sub-optimal architecture. In the river-crossing system example above, this may amount to making a decision to build a bridge without considering the ferry option altogether.

Capturing the knowledge about existing systems and analysis and design of conceived systems require an adequate methodology, which should be both formal and intuitive. Formality is required to maintain a coherent representation of the system under study, while the requirement that the methodology be intuitive stems from the fact that humans are the ultimate consumers of the knowledge. Object-Process Methodology (OPM; [Dor95, Dor02]) is an ontology- and systems theory-based vehicle for co-design via conceptual modeling, as well as for knowledge representation and management, which perfectly meets the formality and intuition requirements through a unique combination of graphics and natural language.

Modeling of complex systems should conveniently combine structure and behavior in a single model. Motivated by this observation, OPM is a comprehensive, holistic approach to modeling, study, development, engineering, evolution, and lifecycle support of systems. Employing a combination of graphics and a subset of English, the OPM paradigm integrates the object-oriented, process-oriented, and state transition approaches into a single frame of reference that is expressed in both graphics and text. Structure and behavior coexist in the same OPM model without highlighting one at the expense of suppressing the other to enhance the comprehension of the system as a whole.

A systems modeling methodology and language must be based on a solid, generic ontology. The next section discusses this term as an introduction to presenting the OPM ontology that follows.

5.2.2 Ontology

Ontology is defined as a branch of philosophy that deals with modeling the real world [WSW99]. Ontology discusses the nature and relations of being, or the kinds of existence [Ont10]. More specifically, ontology is the study of the categories of things that exist or may exist in some domain [Sow00]. The product of such a study, called ontology, is a catalog of the types of things that are assumed to exist in a domain of interest from the perspective of a person who uses a specific language, for the purpose of talking about that domain.

References

[Bad92] A. Baddeley. Working memory. *Science*, 255:556–559, 1992.

[CP91] J.M. Clark and A. Paivio. Dual coding theory and education. *Educational Psychology Review*, 3:149–210, 1991.

[CS91] P. Chandler and J. Sweller. Cognitive load theory and the format of instruction. *Cognition and Instruction*, 8:293–332, 1991.

[Dor95] D. Dori. Object-process analysis: Maintaining the balance between system structure and behavior. *Journal of Logic and Computation*, 5(2):227–249, 1995.

[Dor02] D. Dori. *Object-Process Methodology: A Holistic Systems Paradigm*. Springer Verlag, Berlin, Germany, 2002.

[DRBS03] D. Dori, I. Reinhartz-Berger, and A. Sturm. Developing complex systems with object-process methodology using OPCAT. In *Proceedings of ER 2003, Lecture Notes in Computer Science (2813)*, pages 570–572, 2003.

[GL92] A.M. Glenberg and W.E. Langston. Comprehension of illustrated text: Pictures help to build mental models. *Journal of Memory and Language*, 31:129–151, 1992.

[Heg92] M. Hegarty. Mental animation: Inferring motion from static displays of mechanical systems. *Journal of Experimental Psychology: Learning, Memory and Cognition*, 18:1084–1102, 1992.

[May03] R.E. Mayer. The promise of multimedia learning: Using the same instructional design methods across different media. *Learning and Instruction*, 13:125–139, 2003.

[Mil56] G.A. Miller. The magical number seven, plus or minus two: Some limits on our capacity for processing information. *Psychological Review*, 63:81–97, 1956.

[MM03] R.E. Mayer and R. Moreno. Nine ways to reduce cognitive load in multimedia learning. *Educational Psychologist*, 38(1):43–52, 2003.

[Ont10] Ontologymarkuplanguage version 3.0. http://www.ontologos.org/OML/OML

[Sow00] J.F. Sowa. *Knowledge Representation: Logical, Philosophical, and Computational Foundations*. Brooks Cole Publishing Co., Pacific Grove, California, 2000.

[vG87] E. von Glaserfeld. *The Construction of Knowledge: Contributions to Conceptual Semantics*. Intersystems Publications, Seaside, California, USA, 1987.

[WSW99] Y. Wand, V.C. Storey, and R. Weber. An ontological analysis of the relationship construct in conceptual modeling. *ACM Transactions on Database Systems*, 24(4):494–528, 1999.