Modeling Knowledge with Object-Process Methodology

Dov Dori

Technion, Israel Institute of Technology

and

Massachusetts Institute of Technology

1. Introduction

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) is an ontology- and systems theory-based vehicle for knowledge representation and management that perfectly meets the formality and intuition requirements through a unique combination of graphics and natural language. We start by a brief account of ontology and general systems theory, which form the basis for OPM.

Ontology is defined as a branch of philosophy that deals with modeling the real world (Wand and Weber, 1989). Ontology discusses the nature and relations of being, or the kinds of existence (Ontology Markup Language, 2001). More specifically, ontology is the study of the *categories of things* that exist or may exist in some domain (Sowa, 2001). 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. The traditional goal of ontological inquiry is to discover those fundamental categories or kinds that define the objects of the world. The natural and abstract worlds of pure science, however, do not exhaust the applicable domains of ontology. There are vast, human-designed and human-engineered systems such as manufacturing plants, businesses, military bases, and universities, in which ontological inquiry is just as relevant and equally important. In these human-created systems, ontological inquiry is primarily motivated by the need to understand, design, engineer, and manage such systems effectively. Consequently, it is useful to adapt the traditional techniques of ontological inquiry in the natural sciences to these domains as well (IDEF Family of Methods, 2001).

The types in the ontology represent the predicates, word senses, or concept and relation types of the language L when used to discuss topics in the domain D. An uninterpreted logic, such as predicate calculus, conceptual graphs, or Knowledge Interchange Format (2001), is ontologically neutral. It imposes no constraints on the subject matter or the way the subject may be characterized. By itself, logic says nothing about anything, but the combination of logic with ontology provides a language about the entities in the domain of interest and relationships among them.

An *informal ontology* may be specified by a catalog of types that are either undefined or defined only by statements in a natural language. In every domain, there are phenomena that the humans in that domain discriminate as (conceptual or physical) objects, processes, states, and relations. A *formal ontology* is specified by a collection of names for concept and relation types organized in a partial ordering by the Generalization-Specialization (also referred to as the type-subtype) relation. Formal ontologies are further distinguished by the way the subtypes are distinguished from their supertypes. An *axiomatized ontology* distinguishes subtypes by axioms and definitions stated in a formal language, such as logic; a *prototype-based ontology* distinguishes subtypes by a comparison with a typical member or prototype for each subtype. Examples of axiomatized ontologisms include formal theories in science and mathematics, the collections of rules and frames in an expert system, and specifications of conceptual schemas in languages

like SQL. OPM concepts and their type ordering are well defined, hence OPM belongs to the family of axiomatized ontology.

The IDEF5 method (IDEF Family of Methods, 2001) is designed to assist in creating, modifying, and maintaining ontologies. Ontological analysis is accomplished by examining the vocabulary that is used to discuss the characteristic objects and processes that compose the domain, developing definitions of the basic terms in that vocabulary, and characterizing the logical connections among those terms. The product of this analysis, an ontology, is a domain vocabulary complete with a set of precise definitions, or axioms, that constrain the meanings of the terms sufficiently to enable consistent interpretation of the data that use that vocabulary.

2. General Systems Theory

General systems theory (GST) argues that however complex or diverse the world that we experience is, we will be able to describe it by concepts and principles that are common to all systems and are independent of their domain of discourse. Following this argument, uncovering and establishing those commonalties would enable us to analyze and solve problems in any domain, pertaining to any type of system.

2.1 A Brief History of General Systems Theory

The dawn of General Systems Theory (GST) can be traced back to Aristotle, who articulated the basic synergy principle: *The whole is more than the sum of the parts.* Galileo, who emphasized the analytic approach, replaced this synthesis-oriented view. The analytic approach, which is based on experimentation and introspection, opened the door for modern scientific analysis. Descartes developed the scientific method to be able to analyze complex phenomena by breaking them into elementary particles (which we now call objects) and processes. OPM rests on these same premises: objects and process.

In modern times, efforts to construct a unifying theory that tackles complex systems in the various domains of human activity – natural, social and engineering sciences – dates to the early 1920s. Lotka (1956) articulated the principles of what would become modern systems theory, and applied them to biological phenomena, such as the circulation of elements and growth of organisms. Defay (1929) and Schrödinger (1967) utilized thermodynamic principles to explore biological systems and made it clear that an organism is an open system that exchanges matter and energy with its environment in order to remain stable.

Bertalanffy (1968) established GST principles on the basis of ideas he developed in the 1930's and published in 1955. GST reconciles competing concepts of cybernetics and system dynamics. Just as in economics, Keynes defined "whole" as the entire economic system, in biology Darwin defined "whole" as a system of nature. In spite of the remoteness between economics and biology, they share common system principles. Understanding the need for communication among domain experts to increase the overall knowledge of the operation of the system, Bertalanffy (1975) listed the following aims of GST:

- There is a general tendency towards integration in the various sciences, natural and social;
- Such integration seems to be centered in a general theory of systems; and
- This theory may be an important means for aiming at exact theory in the non-physical fields of science.
- Developing unifying principles that run through the universe of the individual sciences, this theory brings us nearer to the goal of the unity of science, and may lead to a much-needed integration in scientific education.

Bertalanffy viewed GST as being comprised of three elements:

Mathematical Systems Theory: The description of the system is provided in terms of a set of measures that define the states and transformations of the system at various points in time. Formal mathematics, such as a set of differential equations or a graph-theoretical description are employed for this purpose. The system is precisely described from an *internal* aspect, using attributes such as stability, wholeness and sum, growth mechanisms, competition and finality. Externally, the system is described in "black box" terms of inputs and outputs – we do not know what goes on inside – and in control theory terms, such as feedback and goal. This mode is useful for thinking about the system and its environment.

System technology: Society and its use of technology have become so complex, that they are no longer amenable to traditional analysis. GST allows one to effectively cope with this complexity. Ecosystems,

industrial complexes, education, urban and political environments, socioeconomic entities and a variety of organizations exhibit structure and behavior that lend them to analysis within the GST framework.

System philosophy: GST strives to be a fully articulated worldview that contrasts with the mechanistic framework of the traditional scientific approach. GST is a new paradigm, complete with ontology and epistemology, covering "real systems, conceptual systems and abstract systems." The coverage of real systems pertains to the scientific approach, while the conceptual and abstract ones may be new to traditional human thinking habits.

In 1956, Boulding (1956) identified the communication problems that can occur during systems integration: subsystem specialists (which we call domain experts, e.g., physicists, ecumenists, chemists, sociologists, etc.) have their own languages, but in order for successful integration to take place, all subsystem specialists must speak a common language, such as mathematics. The communication among specialists of various domains at some level in the hierarchy of systems discussed below contributes to the development of knowledge at higher levels.

Mathematics is so widely recognized for its generality and abstracting power, that it has been used in almost any domain of knowledge and human interest. Building on graph theory, OPM can be viewed as a field of mathematics for general systems modeling. Conversely, mathematics can be expressed in OPM terms, with variables being objects and operators, processes.

In their seminal work, Shannon and Weaver (1949) listed three stages in the development of scientific analysis: (1) Organized simplicity, which is the basis for classical mechanics and based on the assumption that the orderliness of the world is built up from simple units and relations; (2) unorganized complexity, the basis for statistical physics, which accounts for complexity arising from random occurrences; and (3) organized complexity, expressed by information theory, which accounts for complexity by identifying fundamental ordering relations. They claimed that the third stage is the model for science in the 20th century. In 1948, Wiener (1961) suggested that cybernetics draws on systems, information, and control theories. He analyzed feedback and goal-directed behavior and applied them to social, biological and mechanical systems. His envisaging of the central role of computers in industry and intellectual processes provided the impetus for systems dynamics theory and cognitive science.

Kerzner (1995) defined GST as "an approach that attempts to integrate and unify scientific information across many fields of knowledge." GST tries to solve problems by looking at the total picture, rather than through an analysis of the individual components. He goes on to note that an increasing number of scholars are recognizing the central role General Systems Theory should play in various, seemingly remote domains, such as project management.

2.2 The Hierarchy of System Levels

Boulding (1956) postulated that all areas of scientific interest could be categorized according to their level of development in the following universal hierarchy of system levels.

(1) The *level of frameworks* – the level of static structure, for example the anatomy of the universe;
(2) The *level of clockworks* – the level of simple dynamic systems with predetermined motion;
(3) The *thermostat level* – the level at which the system is self-regulating in maintaining equilibrium through control or cybernetic mechanisms;
(4) The *cell level* – the level of self-maintaining structure, at which life begins to differentiate from not-life;
(5) The *genetic-societal level*, which is typified by the plant and dominated by the empirical world of the botanist;
(6) The *animal system level*, which is characterized by mobility, teleological (purposeful) behavior and self-awareness;
(7) The *human level*, at which the human being is considered as a system with self-awareness and the ability to utilize language and symbolism;
(8) The *social system level*, which exhibits human organizations, complete with value systems, communication and education abilities, emotions and history recording capabilities; and finally
(9) *Transcendental systems level*, which feature the ultimate and absolutes, and the inescapable and unknowable things that nevertheless exhibit systemic structure and relationship.

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 system's function is enabled by its architecture – the combination of structure and behavior. The system's 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 – raw materials, 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.

Modeling of complex systems should conveniently combine structure and behavior in a single model. Motivated by this observation, OPM (Dori 1995, 2002) 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. 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.

Rather than requiring that the modeler views each of the system's aspects in isolation and struggle to mentally integrate the various views, OPM offers an approach that is orthogonal to customary practices. According to this approach, various system aspects can be inspected in tandem for better comprehension. Complexity is managed via the ability to create and navigate via possibly multiple detail levels, which are generated and traversed through by several abstraction/refinement mechanisms.

Due to its structure-behavior integration, OPM provides a solid basis for representing and managing knowledge about complex systems, regardless of their domain. This chapter provides an overview of OPM, its ontology, semantics, and symbols. It then describes applications of OPM in various domains.

3. The OPM Ontology

The elements of the OPM ontology, shown in Table 1, are divided into three groups: entities, structural relations, and procedural links.

3.1 Entities

Entities, the basic building blocks of any system modeled in OPM, are of three types: stateful objects, namely *objects* with *states*, and *processes*. As defined below, processes transform objects by (1) creating them, (2) destroying them, or (3) changing their state. The symbols for these three entities are respectively shown as the first group of symbols at the left hand side of Figure 1, which is the symbols in the toolset available as part of the GUI of OPCAT 2 (Dori, Reinhartz-Berger et al. 2003).

Figure 1. The three groups of OPM symbols in the toolset of OPCAT 2

3.2 OPM Things: Objects and Processes

Objects are (physical or informatical) things that exist, while *processes* are things that transform (create, destroy, or change the state of) objects. Following is a set of basic definitions that build on top of each other.

*An **object** is a thing that exists.*

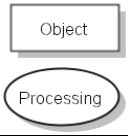
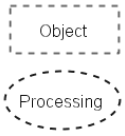
Objects are the things that are being transformed in the system.

> ***Transformation*** *is generation (creation) or consumption (destruction) of an object, or a change of its state.*

Processes are the things that transform objects in the system.

> *A* ***process*** *is a thing that represents a pattern of object transformation.*

Table 1. Things of the OPM ontology and their basic attributes

Thing / Attribute	Symbol	Description / OPL sentence
Object		A thing (entity) that has the potential of stable, unconditional physical or mental existence.
		Object Name is an object**.**
Process		A thing representing a pattern of transformation that objects undergo.
		Processing is a process**.**
Essence		An attribute that determines whether the thing (object or process) is physical (shaded) or informational.
		Processing is physical**.**
Affiliation		An attribute that determines whether the thing is environmental (external to the system, dashed contour) or systemic.
		Processing is environmental**.**

In OPL, **bold Arial font** denotes non-reserved phrases, while non-bold Arial font denotes reserved phrases. In OPCAT, various OPM elements are colored with the same color as their graphic counterparts (by default, objects are green, processes are blue, and states are brown).

Objects and processes are collectively called *things*. The first two lines of Table 1 show the symbol and a description of the two types of OPM things. The next two lines show two basic attributes that things can have: essence and affiliation.

> ***Essence*** *is an attribute that determines whether the thing is physical or informational.*

The default essence is informatical. A thing whose essence is physical is symbolized by a shaded shape.

> ***Affiliation*** *is an attribute that determines whether the thing is environmental (external to the system) or systemic.*

The default affiliation is systemic. A thing whose affiliation is environmental is symbolized by a dashed contour.

3.3 OPM States

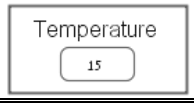
Objects can be stateful, i.e., they may have one or more states.

*A **state** is a situation at which an object can exist at certain points during its lifetime or a value it can assume.*

Stateful objects can be affected, i.e., their states can change.

***Effect** is a change in the state of an object.*

Table 2. States and values

	Symbol	Description / OPL sentence
Stateful object with two states		A situation at which an object can exist. **Website** can be **reachable** or **unreachable.**
Value		A value that an object can assume. **Temperature** is **15.**
Stateful object with three states: initial, default, and final		A state can be initial, default, or final**.** **Car** can be **new,** which is **initial, used,** which is **default,** or **junk,** which is **final.**

4. OPM Structure Modeling

Structural relations express static, time-independent relations between pairs of entities, most often between two objects. Structural relations, shown as the middle group of six symbols in Figure 1, are of two types: fundamental and tagged.

4.1 The four fundamental structural relations

Fundamental structural relations are a set of four structural relations that are used frequently to denote relations between things in the system. Due to their prevalence and usefulness, and in order to prevent too much text from cluttering the diagram, these relations are designated by the four distinct triangular symbols shown in Figure 1.

The four fundamental structural relations are:

(1) aggregation-participation, a solid triangle, ▲, which denotes the relation between a whole thing and its parts,

(2) generalization-specialization, a blank triangle, △, which denotes the relation between a general thing and its specializations, giving rise to inheritance,

(3) exhibition-characterization, a solid inside blank triangle, ◬, which denotes the relation between an exhibitor – a thing exhibiting a one or more features (attributes and/or operations) – and the things that characterize the exhibitor, and

(4) classification-instantiation, a solid circle inside a blank triangle, ◬, which denotes the relation between a class of things and an instance of that class.

Table 3. The fundamental structural relation names, OPD symbols, and OPL sentences

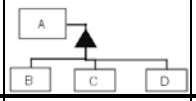
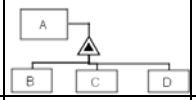
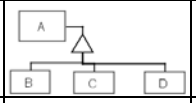
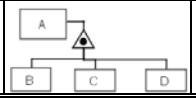
Structural Relation Name		Root Refineables	OPD with 3 refineables	OPL Sentences with 1, 2, and 3 refineables
Forward	Backward			
Aggregation	Participation	Whole Parts		**A** consists of **B.** **A** consists of **B** and **C.** **A** consists of **B, C,** and **D.**
Exhibition	Characterization	Exhibitor Features		**A** exhibits **B.** **A** exhibits **B** and **C.** **A** exhibits **B, C,** and **D.**
Generalization	Specialization	General Specializations		**B** is an **A.** **B** and **C** are **As.** **B, C,** and **D** are **As.**
Classification	Instantiation	Class Instances		**B** is an instance of **A.** **B** and **C** are instances of **A.** **B, C,** and **D** are instances of **A.**

Table 3 lists the four fundamental structural relations and their respective OPDs and OPL sentences. The name of each such relation consists of a pair of dash-separated words. The first word is the forward relation name, i.e., the name of the relation as seen from the viewpoint of the thing up in the hierarchy. The second word is the backward (or reverse) relation name, i.e., the name of the relation as seen from the viewpoint of the thing down in the hierarchy of that relation.

Each fundamental structural relation has a default, preferred direction, which was determined by how natural the sentence sounds. In Table 3, the preferred shorthand name for each relation is underlined. As Table 3 shows, each one of the four fundamental structural relations is characterized by the hierarchy it induces between the root—the thing attached to the tip of the triangle and the leaves—the thing(s) attached to the base of the triangle, as follows.

(1) In aggregation-participation, the tip of the solid triangle, ▲, is attached to the whole thing, while the base—to the parts.

(2) In generalization-specialization, the tip of the blank triangle, △, is attached to the general thing, while the base—to the specializations.

(3) In exhibition-characterization, the tip of the solid inside blank triangle, ◬, is attached to the exhibitor (the thing which exhibits the features), while the base is attached to the features (attributes and operations).

(4) In classification-instantiation, the tip of the solid circle inside a blank triangle, ◬, is attached to the thing class, while the base—to the thing instances.

The things which are the leaves of the hierarchy three, namely the parts, features, specializations, and instances, are collectively referred to as *refineables*, since they refine the ancestor, the root of the tree.

Refineable *is a generalization of part, feature, specialization, and instance.*

The third column in Table 3 lists for each fundamental structural relations the name of the root (whole, exhibitor, general, class) and the corresponding refineables (parts, features, specializations, and instances). The next column contains an OPD with three refineables, while the rightmost column lists the syntax of three OPL sentences for each fundamental structural relation, with one, two, and three refineables, respectively.

Having presented the common features of the four fundamental structural relations, in the next four subsections we provide a small example for each one of them separately.

4.2 aggregation-participation

Aggregation-participation denotes the relation between a whole and it comprising parts or components. Consider, for example, the excerpt taken from Section 2.2 of the RDF Primer (Manola and Miller 2003):

… each statement consists of a subject, a predicate, and an object.

This is a clear case of whole-part, or aggregation-participation relation. The OPM model of this statement, which consists of both the OPD and the corresponding OPL, is shown in Figure 2. Note that the OPL sentence, "**RDF Statement** consists of **Subject, Predicate,** and **Object.**" which was generated by OPCAT automatically from the graphic input, is almost identical to the one cited from the RDF Primer. The same OPD exactly (disregarding the graphical layout) can be produced by inputting the text of the OPL sentence above. This is a manifestation of the OPM graphics-text equivalence principle.

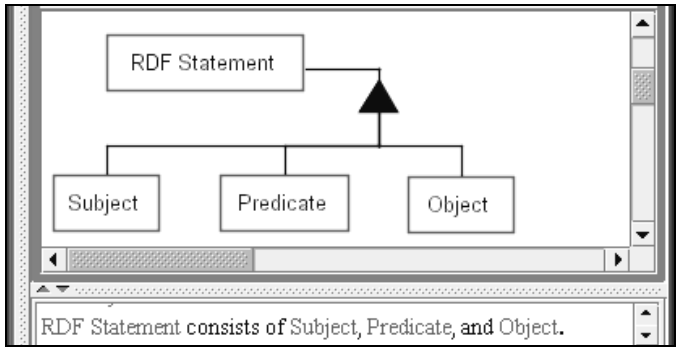


Figure 2. OPD of the sentence "RDF Statement consists of Subject, Predicate, and Object."

4.3 Generalization-specialization

Generalization-specialization is a fundamental structural relationship between a general thing and one or more of its specializations. Continuing our example from the RDF Primer (Manola and Miller 2003), consider the very first sentence from the abstract:

The Resource Description Framework (RDF) is a language for representing information about resources in the World Wide Web.

Let us take the main message of this sentence, which is that *RDF is a language*. This is exactly in line with the OPL syntax, so we can input the OPL sentence "**RDF** is a **Language.**" into OPCAT and see what we get.

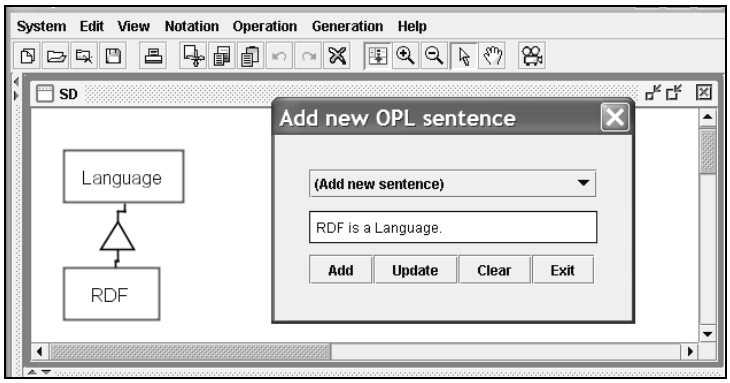


Figure 3. The OPD obtained by inputting into OPCAT the OPL sentence "**RDF** is a **Language.**"

The result, without any diagram editing, is shown in Figure 3, along with the conversation window titled "Add new OPL sentence," in which this sentence was typed prior to the OPD creation.

4.4 Exhibition-characterization

We continue to scan the RDF Primer (Manola and Miller 2003), where is Section 2.2.1 we find the sentence

RDF has a simple data model.

To model this statement, we need to rephrase this sentence into the following three sentences:

1. RDF is characterized by a data model.
2. The data model of RDF is characterized by a complexity attribute.
3. The value of this complexity attribute is "simple."

These three sentences are further rephrased to conform to the OPL syntax as follows:

1. **RDF** exhibits **Data Model.**
2. **Data Model** exhibits **Complexity.**
3. **Complexity** is **simple.**

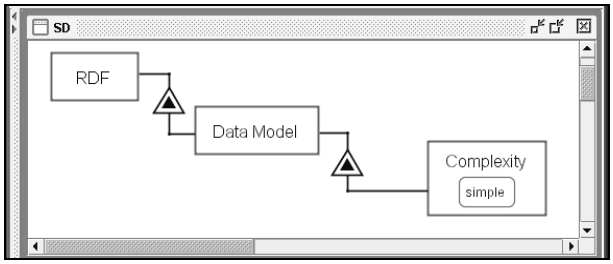


Figure 4. The OPD representing the sentence "*RDF has a simple data model.*"

4.5 Classification-instantiation

Reading through the RDF Primer, we find in Section 3.3 on datatypes the sentence:

Datatypes are used by RDF in the representation of values, such as integers, floating point numbers, and dates.

…

RDF predefines just one datatype, `rdf:XMLLiteral`*, used for embedding XML in RDF.*

An OPL interpretation of these two sentences, respectively, is:

1. **RDF** exhibits many **Datatypes.**
2. **XMLLiteral** is an instance of **Datatype.**

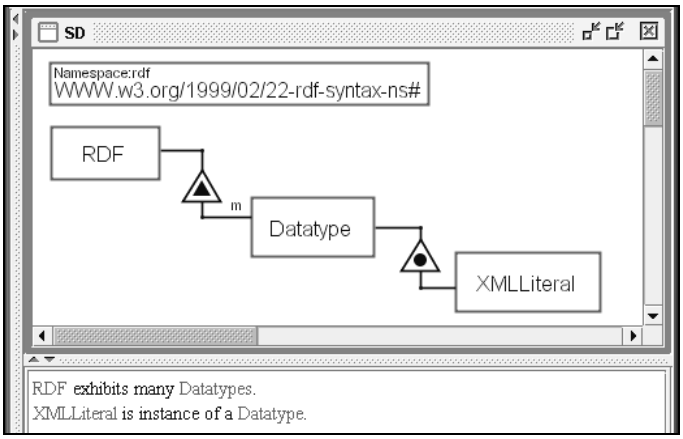


Figure 5. The OPM model of XMLLiteral, an instance of a Datatype of RDF

Figure 5 is the OPM model of **XMLLiteral**, an instance of a **Datatype** of **RDF**.

5. OPM Behavior Modeling

Procedural links connect entities (objects, processes, and states) to express dynamic, time-dependent behavior of the system. Behavior, the dynamic aspect of a system, can be manifested in OPM in three ways:

(1) A process can *transform* (generate, consume, or change the state of) objects,

(2) An object can *enable* a process without being transformed by it, and

(3) An object or a process can *trigger* an event that might, in turn, invoke a process if some conditions are met.

Accordingly, a procedural link can be a transformation link, an enabling link, or an event link.

In order to be able to talk about object transformation, we need to first define state and demonstrate how states are used.

5.1 Object states

In Figure 6 we added to the object **Check** two states: The initial state **uncashed** and the final state **cashed**. This causes the addition of the following OPL sentence to the OPL paragraph:

Check can be **uncashed,** which is initial**,** or **cashed,** which is final**.**

5.2 Transformation links

A *transformation link* expresses how a process transforms one or more objects. The transformation of an object can be its consumption, generation, or state change. The transforming process is the *transformer*, while object that is being transformed is called *transformee*.

5.3 Input and output links

Having added the states to the object **Check**, we can now show how the process **Cashing** affects **Check** by changing its state. In Figure 7, **Cashing** was added and linked to the two states of **Check**: An input link leads from the initial **uncashed** state to **Cashing**, while an output link leads from **Cashing** to the final state **cashed**.

The OPL sentence generated automatically by OPCAT as a result of adding these input and output links is:

Cashing changes **Check** from **uncased** to **cashed.**

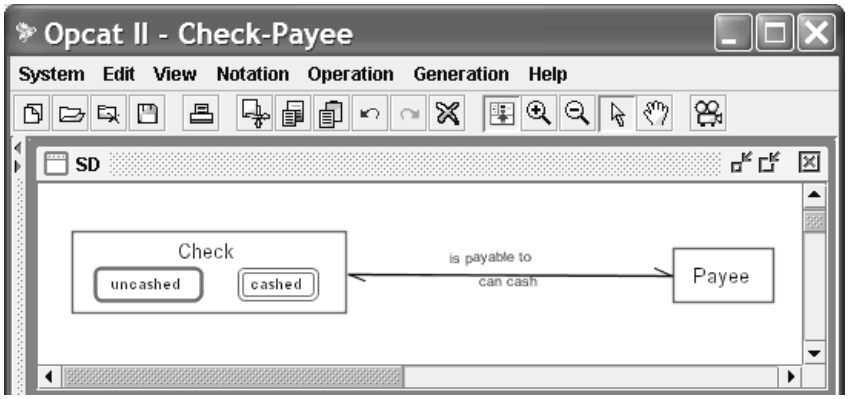


Figure 6. Adding states to Check

Table 4. OPD and OPL syntax for objects with one, two, and three or more states, and optional time designator attributes

Number of states or timeline	OPD	OPL
Single state	Stateful Object [singular]	**Stateful Object** is **singular.**
Two states	Stateful Object [singular] [plural]	**Stateful Object** can be **singular** or **plural.**
Three states or more	Stateful Object [first] [second] [third] [fourth]	**Stateful Object** can be **first, second, third,** or **fourth.**
Three states or more	Stateful Object [first] [second] [third] [fourth]	**Stateful Object** can be **first,** which is initial**, second, third,** which is default**,** or **fourth,** which is final**.**

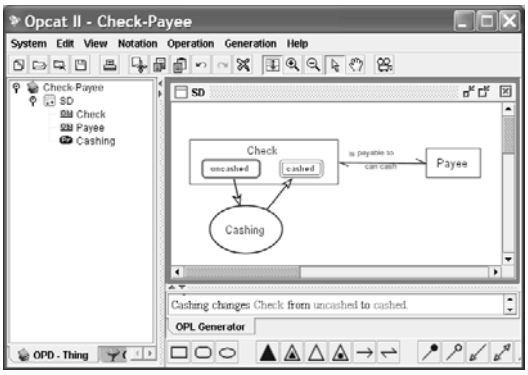


Figure 7. The **Cashing** process changes the state of **Check** from the **uncashed** to **cashed**.

5.4 Effect link

Sometimes we may not be interested in specifying the states of an object but still show that a process does affect an object by changing its state from some unspecified input state to another unspecified output state. To express this, we suppress (hide) the input and output states of the object, so the edges of the input and output links "migrate" to the contour of the object and coincide, yielding the effect link shown in Figure 8.

The OPL sentence that represents this graphic construct is:

Cashing affects **Check.**

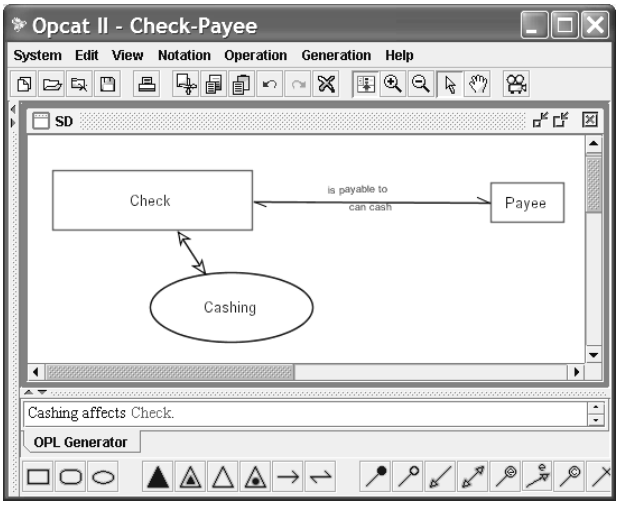


Figure 8. Suppressing the input and output states of **Check** cause the two link edges to migrate to the contour of **Check** and coincide, yielding the single bidirectional effect link between **Check** and **Cashing**.

5.5 Result and consumption links

We have seen that one type of object transformation is effect, in which a process changes the state of an object from some input state to another output state. When these two states are expressed (i.e., explicitly shown), then we can use the pair of input and output links to specify the source and destination states of the transformation. When the states are suppressed, we express the state change by the effect link, a more general and less informative transformation link.

State change is the least drastic transformation that an object can undergo. Two more extreme transformations are generation and consumption, denoted respectively by the result and consumption links.

Generation is a transformation which causes an object, which had not existed prior to the process execution, to become existent. For example, **Check** is born as a result of a **Check Making** process.

As Figure 9 shows, the object **Check** is generated as a result of executing the process **Check Making**. The result link is the arrow originating from the generating process and leading to the generated object. The OPL sentence that represents this graphic construct (shown also in Figure 9) is:

Check Making yields **Check.**

In contrast to generation, consumption is a transformation which causes an object, which had existed prior to the process execution, to become non-existent. For example, **Check** is consumed as a result of a **Destroying** process.

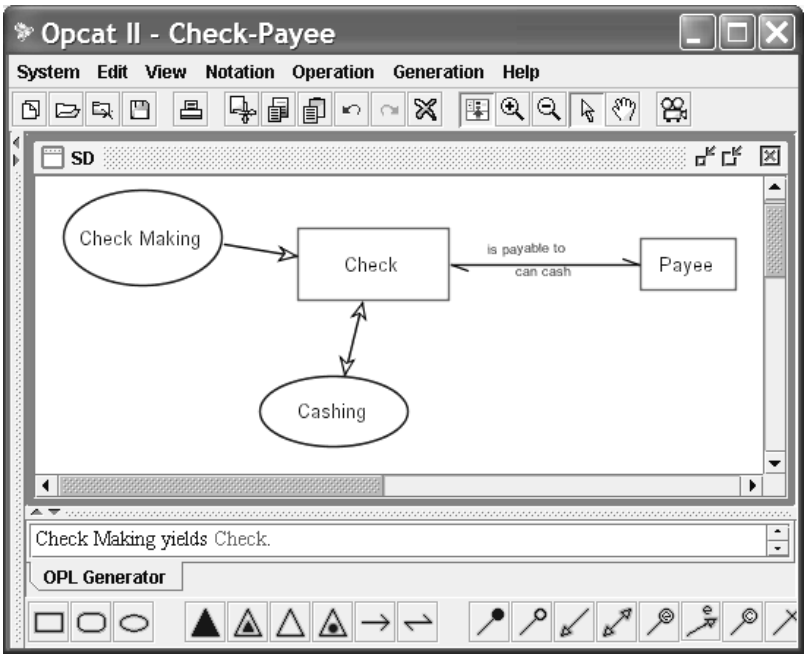


Figure 9. The object **Check** is generated as a result of executing the **Check Making** process.

As Figure 10 shows, the object **Check** is consumed as a result of executing the process **Destroying**. The consumption link is the arrow originating from the consumed object and leading to the consuming process. The OPL sentence that represents this graphic construct (shown also in Figure 10) is:

Destroying consumes **Check.**

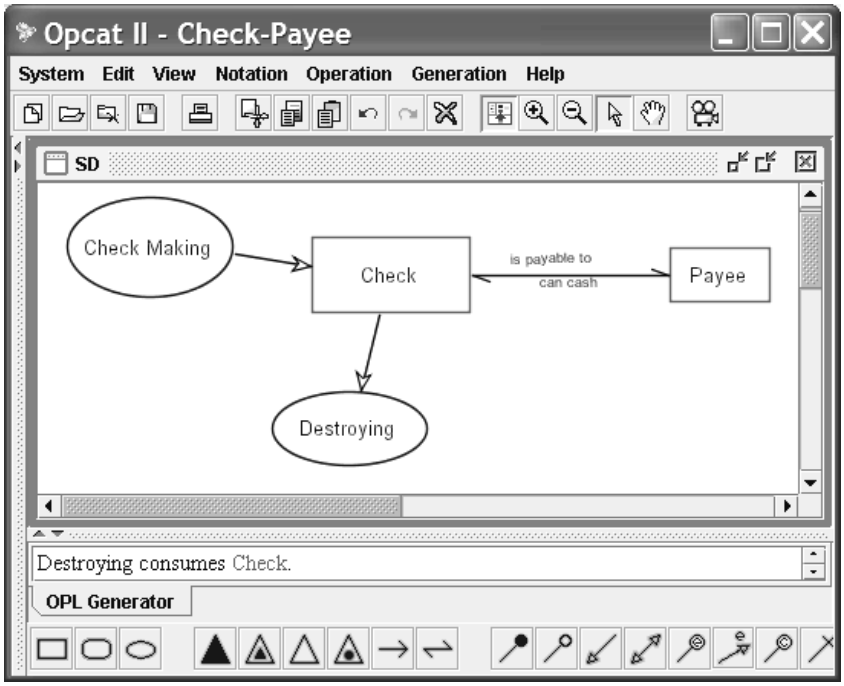


Figure 10. The object **Check** is consumed as a result of executing the **Destroying** process.

5.6 State-specified result and consumption links

We sometimes wish to be specific and state not only that an object is generated by a process, but also at what state that object is generated. Some other times, we might wish to be able to state not only that an object is consumed by a process, but also at what state that object has to be in order for it to be consumed by the process. As Figure 9 shows, the object **Check** is generated in its **unendorsed** state as a result of executing the process **Check Making**.

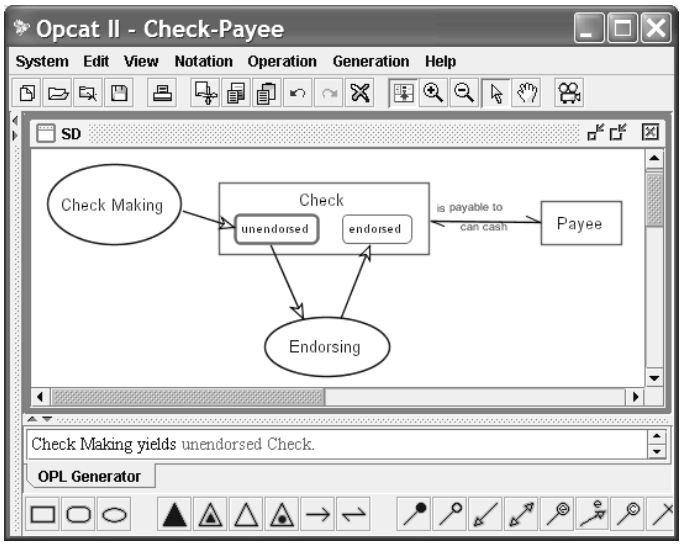


Figure 11. The object **Check** is generated in its **unendorsed** state as a result of executing the **Check Making** process.

The OPL sentence that represents this state-specified result link graphic construct (shown also in Figure 11) is:

Check Making yields **unendorsed Check.**

In comparison, the "regular," non-state-specified result link is the same, except that the (initial) state is not specified:

Check Making yields **Check.**

The difference is the addition of the state name (**unendorsed** in our case) before the name of the object (**Check**) that owns that state.

Analogously, a state-specified consumption link leads from a (final) state to the consuming process. For example, assuming a check can only be destroyed if it is cashed, Figure 12 shows the state-specified consumption link leading from the final state cashed of **Check** to the consuming process **Destroying**.

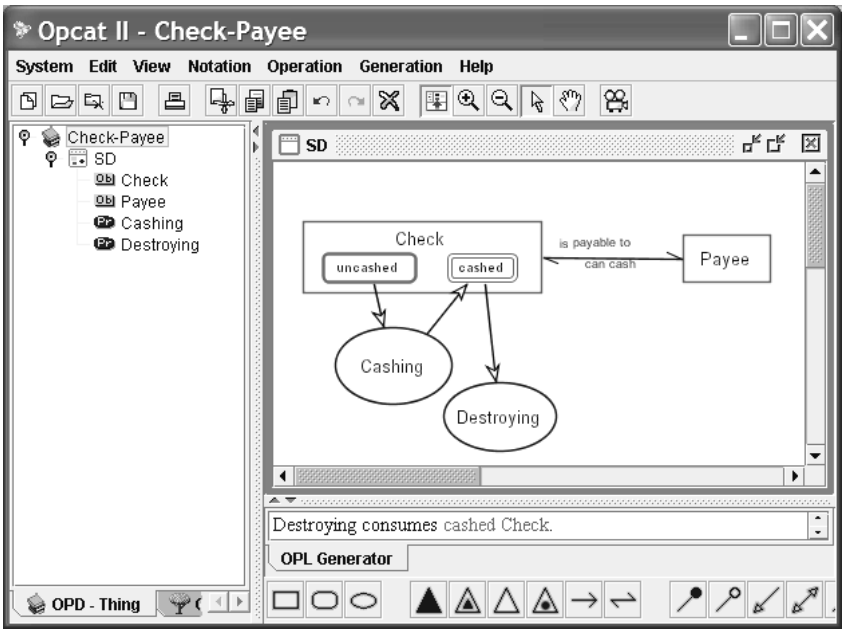


Figure 12. The object **Check** is consumed in its **cashed** state as a result of executing the **Destroying** process.

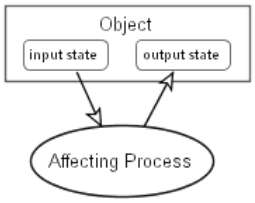
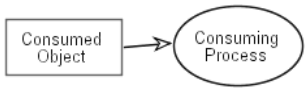
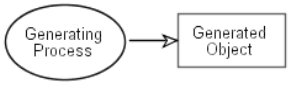
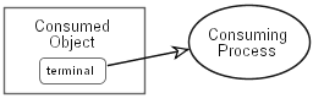
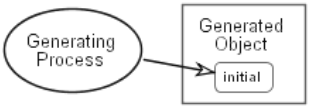
The OPL sentence that represents this state-specified consumption link graphic construct (shown also in Figure 12) is:

Destroying consumes **cashed Check.**

5.7 Summary of procedural links between processes and objects

Table 5 provides a summary of the six procedural links between a process and a (possibly stateful) object. They are divided into three pairs: Input and output links, which always come as a pair, consumption and result links, and state-specified consumption and result links.

Table 5. Summary of the procedural links between a process and an object or its state

Transformation Link Type	Source Entity	Destination Entity	OPD	OPL
Input link	Input state	Affecting process		**Affecting Process** changes **Object** from **input state** to **output state.**
Output link	Affecting process	Output state		
Consumption link	Consumed object	Consuming process		**Consuming Process** consumes **Consumed Object.**
Result link	Generating process	Resulting object		**Generating Process** yields **Generated Object.**
State-specified Consumption link	Final state of the consumed object	Consuming process		**Consuming Process** consumes **terminal Consumed Object.**
State-specified Result link	Generating process	Initial state of the resulting object		**Generating Process** yields **initial Generated Object.**

5.8 Enablers and enabling links

An *enabler* is an object that is required for a process to happen, but is not transformed by the process. An *enabling link* expresses the need for a (possibly state-specified) object to be present in order for the enabled process to occur. The enabled process does not transform the enabling object. Enablers are divided into instruments and conditioners, each of which can be stateless or stateful.

6. Applications of OPM and Summary

OPM has been applied in many domains, including education (Dori & Dori, 1996), computer integrated manufacturing (Dori, 1996A; Dori, Gal & Etzion, 1996), The R&D universe and its feedback cycles (Myersdorf & Dori, 1997), real-time systems (Peleg & Dori, 2000), banking (Dori, 2001), requirements engineering (Soffer, Golany, Dori & Wand, 2001), Web applications development (Reinhartz-Berger, Dori, & Katz, 2002), ERP modeling (Soffer, Golany & Dori, 2001), axiomatic design (Soderborg, Crawley & Dori, 2002), computational synthesis (Dori & Crawley, 2003), software reuse (Reinhartz-Berger, Dori, & Katz, 2002), systems architecture (Soderborg, Crawley, & Dori, 2003), and Web Service Composition (Yin, Wenyin, & Chan. 2004).

This chapter has presented an overview of Object-Process Methodology and its applications in a variety of domains. There are a number of important OPM-related issues that could not be discussed in detail in this chapter due to space limitations. One such topic is complexity management. Complexity is managed in

OPM via in-zooming, unfolding, and state-expression, which provide for looking at any complex system at any desired level of granularity without loosing the context and the "big picture." Another issue is the systems development and evolution methodology with OPM, for which a comprehensive reflective metamodel (which uses OPM) has been developed. These issues and others are treated in detail in the OPM book (Dori, 2002).

The domain-independent nature of OPM makes it suitable as a general, comprehensive, and multidisciplinary framework for knowledge representation and reasoning that emerge from conceptual modeling, analysis, design, implementation, and lifecycle management. The ability of OPM to provide comprehensive lifecycle support of systems of all kinds and complexity levels is due to its foundational ontology that builds on a most minimal set of stateful objects and processes that transform them. Another significant uniqueness of OPM is its unification of system knowledge from both the structural and behavioral aspects in a single diagram – OPD. It is hard to think of a significant domain of discourse and a system in it, in which structure and behavior are not interdependent and intertwined. A third unique feature of OPM is its dual knowledge representation in graphics and text and the capability to automatically switch between these two modalities. Due to its single model, expressed in both graphics and text, OPM lends itself naturally for representing and managing knowledge, as it is uniquely poised to cater to the tight interconnections between structure and behavior that are so hard to separate.

OPM and its supporting tool OPCAT continue to evolve. The site www.ObjectProcess.org is a rich, continuously updated resource of OPM-related articles, free software downloads, and more.

References

1. Dori, D. (1994). Automated Understanding of Engineering Drawings: An Object-Oriented Analysis. Journal of Object Oriented Programming, 35-43, Sept.
2. Dori, D. (1995). Object-Process Analysis: Maintaining the Balance between System Structure and Behavior. Journal of Logic and Computation, 5(2), 227-249.
3. Dori, D. (1996A). Object-Process Analysis of Computer Integrated Manufacturing Documentation and Inspection. International Journal of Computer Integrated Manufacturing, 9(5), 39-353.
4. Dori, D. (2001). Object-Process Methodology Applied to Modeling Credit Card Transactions. Journal of Database Management, 12(1), 2-12.
5. Dori, D. (2002). Object-Process Methodology - A Holistic Systems Paradigm, Springer Verlag, Berlin, Heidelberg, New York.
6. Dori, D., & Crawley, E. (2003). Towards a Common Computational Synthesis Framework with Object-Process Methodology. 2003 AAAI Spring Symposium Series: Computational Synthesis: From Basic Building Blocks to High Level Functionality, Stanford University, Stanford, CA, Technical Report SS-03-02.
7. Dori, D., Gal, A., & Etzion, O. (1996). A Temporal Database with Data Dependencies: a Key to Computer Integrated Manufacturing. International Journal of Computer Integrated Manufacturing, 9(2), 89-104.
8. Dori, D., & Dori, Y.J. (1996). Object-Process Analysis of a Hypertext Organic Chemistry Module. Journal of Computers in Mathematics and Science Teaching, 15(1/2), 65-84.
9. IDEF Family of Methods. A Structured Approach to Enterprise Modeling and Analysis, 2001. www.idef.com
10. Knowledge Interchange Format, 2001. http://logic.stanford.edu/kif/
11. Myersdorf, D.. & Dori, D.(1997). The R&D Universe and Its Feedback Cycles: an Object-Process Analysis. R&D Management, 27, 4, 333-344.
12. Ontology Markup Language, 2001. http://wave.eecs.wsu.edu/CKRMI/OML.html
13. Peleg M., & Dori, D. (2000). The Model Multiplicity Problem: Experimenting with Real-Time Specification Methods. IEEE Transaction on Software Engineering, 26(8), 742-759.
14. Reinhartz-Berger, I., Dori, D., & Katz, S. (2002). OPM/Web – Object-Process Methodology for

Developing Web Applications. Annals of Software Engineering. 13, 141–161.

15. Reinhartz-Berger, I., Dori, D., & Katz, S. (2002). Open Reuse of Component Designs in OPM/Web. Proc. IEEE 26th Annual International Computer Software and Applications Conference, 19-26.
16. Soderborg, N., Crawley E., & Dori D. (2002). System Definition for Axiomatic Design Aided by Object-Process Methodology. Proc. 2nd International Conference on Axiomatic Design, Cambridge, MA, USA, 134-140.
17. Soderborg, N., Crawley E., & Dori D. (2003). OPM-Based System Function and Architecture: Definitions and Operational Templates. Communications of the ACM, 46(10), 67-72.
18. Soffer, P., Golany, B., & Dori, D. (2003). ERP Modeling: A Comprehensive Approach. Information Systems 28(6), 673-690.
19. Soffer, P., Golany, B., Dori, D., & Wand Y. (2001). Modeling Off-the-Shelf Information Systems Requirements: An Ontological Approach. Requirements Engineering, 6, 183-199.
20. Sowa, J.F. Principles of Ontology, 2001. http://www-ksl.stanford.edu/onto-std/mailarchive/0136.html
21. Wand, Y. and Weber, R. An Ontological Evaluation of Systems Analysis and Design Methods. In Information System Concepts: An In-Depth Analysis. In Falkenberg, E.D. and Lindgreen, P. (Eds.). Elsevier Science Publishers B.V. (North Holland), IFIP, pp. 145–172, 1989.
22. Wenyin L., & Dori, D. (1998). A Generic Integrated Line Detection Algorithm and its Object-Process Specification. Computer Vision – Image Understanding (CVIU), 70(3), 420-437.
23. Wenyin L., & Dori, D. (1998A). Genericity in Graphics Recognition Algorithms. In Graphics Recognition – Algorithms and Systems, Lecture Notes in Computer Science, K. Tombre and A. K. Chhabra (Eds.), 1389, 9-18.
24. Wenyin L., & Dori, D. (1999). Object-Process Based Graphics Recognition Class Library: Principles and Applications. Software - Practice and Experience, 29, 15, 1355-1378.
25. Yin, L., Wenyin, L., & Changjun, J. (2004). Object-Process Diagrams as Explicit Graphic Tool for Web Service Composition", Journal of Integrated Design & Process Science: Transactions of the SDPS, 8(1), 113-127.

Key Terms

Object-Process Methodology – A generic methodology for systems modeling and knowledge representation based on a single model expressed in graphics and text, in which stateful objects and processes are the basic building blocks.

Object – A thing that exists.

State – A situation at which an object can exist at certain points during its lifetime or a value it can assume.

Transformation – Generation (creation) or consumption (destruction) of an object, or a change of its state.

Process – A thing that represents a pattern of object transformation.

Essence – An attribute that determines whether the thing is informatical (the default) or physical. The

Affiliation – An attribute that determines whether the thing is systemic (the default) or environmental (external to the system).

Structural relation – A relation between objects that holds in the system regardless of time.

Procedural link – A link between an object and a process that expresses the behavior of the system.