

# Semantic Object Language

Whitepaper

Jason Wells

Semantic Research Inc.

## Abstract

While UML is the accepted visual language for object-oriented system modeling, it lacks a common semantic foundation with a standard visual syntax. Furthermore, it does not cleanly mesh with external visual modeling languages such as entity relationship diagrams and relational data models. By restating UML and ER in terms of a common underlying semantic and syntactic modeling platform, a comprehensive, encapsulated, and truly unified software modeling technique becomes feasible.

## Expressing System Design: UML and ER

Object-oriented programming has become the dominant paradigm in software engineering and system design. Contemporary software designers employ the Unified Modeling Language (UML) to describe software systems. It is a graphical notation used to express requirements analysis and software design (Fowler, 2000). Briefly, UML consists of nine core types of diagrams:

- *Activity Diagram*. This shows behavior with control structure. It can show many objects over many uses, many objects in a single use case, or implementation of method.
- *Class Diagram*. This shows static structure of concepts, types, and classes. Concepts show how users think about the world; types show interfaces of software components; classes show implementation of software components.
- *Collaboration Diagram*. This shows how several objects interact in a single use case.
- *Deployment Diagram*. This shows physical layout of components on hardware nodes.
- *Object Diagram*. This shows a snapshot of the objects in a system at a single point in time.
- *Package Diagram*. This shows packages of classes and the dependencies among packages.
- *Sequence Diagram*. This shows how groups of objects collaborate in a chronological sequence.
- *State Diagram*. This shows how a single object behaves across many use cases.
- *Use Case Diagram*. This elicits requirements from users in meaningful chunks, and provides the basis for system testing.

These diagrams are used to express the design of various aspects of an object oriented software system. A typical system may require multiple instances of some or all of these nine fundamental types of diagrams to achieve the necessary descriptive completeness.

Similarly, entity relationship (ER) diagrams (Barker 1990) have been developed to address a closely related need: how to model the entities within a relational database schema. Object-oriented systems routinely rely on a relational database to persist object state. The visual syntax of ER diagrams is distinct from, and not compatible with, formal UML syntax. Many UML and ER tools impose rigid schematic structure on the system model.

UML and ER, by themselves, exhibit several notable drawbacks:

- They are not very easy to learn. Object-oriented design (OOD), by itself, is a full body of engineering technique requiring a great deal of time and effort to master. UML, which is primarily used to describe OOD, employs a specialized and jargonized visual syntax that must be learned. It is not self-evident what meaning is conveyed by various styles of boxes and arrows; the meaning cannot be deduced simply by studying the diagram.
- Because of its rigid framework, it is awkward to express unstructured design problems or the ‘general philosophy’ of an approach in a UML diagram. Typically the UML modeler is expected to force this sort of unstructured information into UML comments, which employs a totally separate syntax than the rest of the design. This creates a kind of design balkanization in which the formal and informal design elements are artificially segregated.
- There are nine different diagrams, each with different syntax rules and appearances. They are not easily combined, causing a cluttered and needlessly divided model.
- ER diagrams do not harmonize well with UML syntax, resulting in another kind of undesirable division in design.
- UML, having weak underlying semantics, requires repetition that would be unnecessary in a richer representation. For example, to type a class member in a UML class diagram, you must repeat the name of a class that also appears as the title of another UML class. There is no visual relationship connecting these two names; the model producers and consumers must make the connection in their own minds.
- Though graphical, not all UML diagrams convey meaning in terms of the structure of the diagram. They may not be subject to topological or network structure analysis.

## **A New Foundation: SOL**

A better approach, which addresses the problems identified above, is to express UML and ER diagrams in terms of an underlying semantic network knowledge representation. This is known as the Semantic Object Language (SOL). In this approach, one or more semantic networks are employed as a mechanism to capture the information typically captured by a UML modeling tool. To explore SOL in some detail, it is useful to examine how a particular UML diagram (in this case, a class diagram) maps to SOL. For example, consider the following class diagram (Figure 1) describing a simple factory pattern.

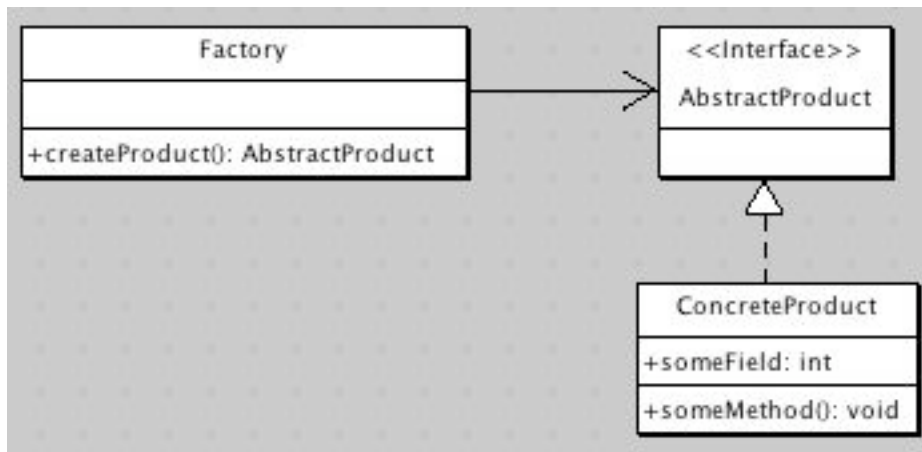


Figure 1. UML class diagram of a factory pattern.

A standard SOL ontology corresponding to a UML class diagram is provided as a starting point. From there, a modeler familiar with the basics of semantic networking (Staszak, 2002a) may create a new semantic network applying the ontology as a template. New concepts are created representing each class. To develop the structure of each class, concepts are created representing each field, method, and attribute of the class. The pre-defined relations contained within the SOL class ontology are used to relate these concepts to their class. Concepts are joined by named relations, leading to ever-growing definition. Class members are typed, and method signatures are defined. Associations are then established between classes. The equivalent of UML comments can be achieved through the use of additional concepts and relations beyond the core SOL ontologies. In this manner, the semantic network can be organically extended in flexible, coherent fashion.

At this point, a semantic network has been constructed containing the class structure of the object model. Figure 2 presents a slice of the full SOL network that describes AbstractProduct.

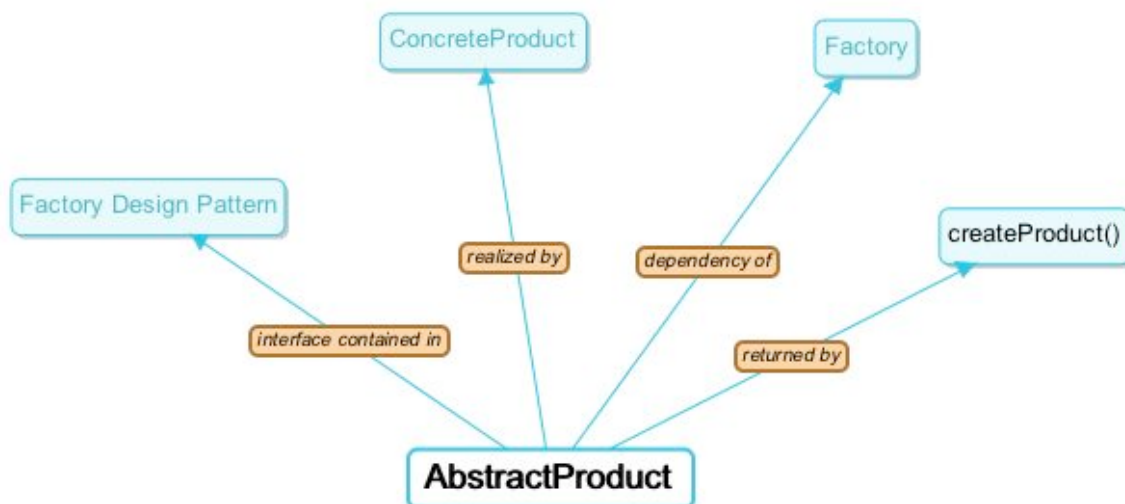


Figure 2. Graphical frame view of the structure of the AbstractProduct interface.

With the class structure now defined in terms of SOL, a variety of new capabilities emerge that address the weaknesses of traditional UML and ER modeling strategies.

Semantic networks, as presented in the Semantica graphical frame view, are easy to learn. UML classes are often a week or more in duration, but learning to interpret and create semantic networks can be achieved in one to two hours. UML is considerably harder to learn than semantic networks in part because of UML's non-intuitive visual jargon: for example, a dotted-line arrow with an open arrowhead signifies 'realization' – the notion of a class implementing an interface. In a semantic network, you would simply have a reflexive, asymmetric relation having one ray reading 'realizes' and the other reading 'realized by.' Unlike UML diagrams, the visualization of a semantic network in a graphical frame view (as in Figure 2) is self-evident. Thus this aspect of the design is comprehensible to anyone who can read and is familiar with OOD principles. In this manner, the sometimes-arcane UML visual jargon may be dispensed with, and the benefits of semantic network theory as applied to learning may be realized (Fisher & Hoffman, 2002). It becomes easier for software designers and business analysts to concentrate on the problem, rather than the tool, and it becomes possible for non-specialists to grasp the model.

Semantic networks are grammar-free and employ no rigid structure beyond the basic primitive elements: concepts, relations, instances, and so on. This means that the OOD modeler can represent unstructured design notions, using the exact same syntax as the rest of the design, resulting in a more elegant design expression. Additionally, this mechanism permits design problems to be expressed before the solution is evident – something that is cumbersome to do effectively in UML.

As SOL embodies a superset of the underlying semantics of UML, the nine core UML diagrams can be represented using one common visual syntax, rather than nine largely incompatible ones. This means that to render a SOL model as a UML class diagram is purely a matter of visual presentation. The underlying representation remains the same and is compatible, as long as the model employs the SOL ontology for OO class structure. Traversing back and forth between SOL and UML becomes straightforward.

While semantic networks have few primitives, their expressive power gives them a great deal of versatility; more than any single type of UML or ER diagram, and more than all ten diagrams combined. Employing SOL, it is straightforward to represent any of these diagrams as a semantic network. In effect, each of the ten diagrams becomes ten ontologies. OO systems routinely rely on RDBMS for persistence. These ontologies may be mixed and matched in a single semantic network. ER and UML representations may then share a common semantic base. This process of semantic unification delivers a single, comprehensive OOD model for the entire object-oriented software system.

Semantic networks resolve the verbose repetition of UML descriptors. In SOL, a concept representing a class participates in one or more distinct instances. This allows for a single expression of a class to be reused in multiple ways: to label a class, or to type a class member. This eliminates several problems associated with non-semantic representation: label repetition is no longer needed, and class may be understood both in terms of its

construction, and use, at once. Consequently, SOL is refreshingly minimalist and parsimonious, sparing the modeler from unwanted verbosity.

For example, consider the following example of the class Element. Element is declared to be the type of a class member e1, as seen in the graphical frame displayed in Figure 3 below. By centering e1 in the graphical frame view (Figure 4), the relation between e1 and Element is reversed, displaying the other ray (“has type”).

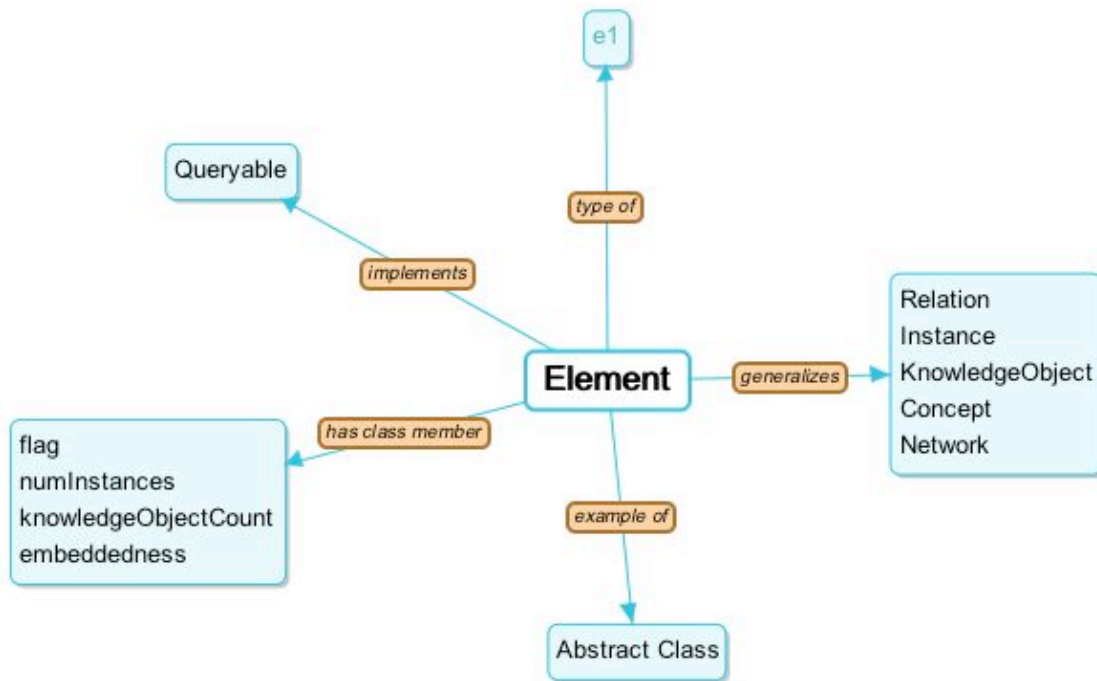


Figure 3. Graphical frame view of the structure of the Element abstract class.

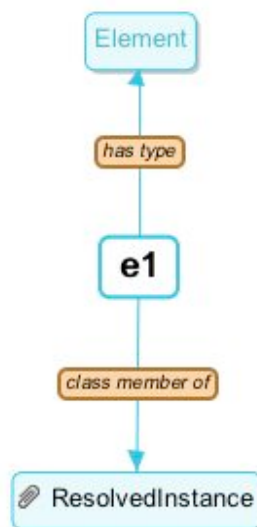


Figure 4. Graphical frame view of e1, a member of the class ResolvedInstance.

In Figure 4 we find that e1 happens to be a member of the class ResolvedInstance as well. Thus the concept representing the class member e1 functions within the context of both Element and ResolvedInstance. In short, the semantic richness of this model is demonstrated in that a class knows not only what it is, but what other things use it, all within a single expression.

SOL models are extensible beyond traditional UML and ER semantics. Knowledge objects, analogous to email attachments, may be attached to any element of a semantic network: concepts, instances, even the network itself. While no set of UML or ER diagrams can fully capture every aspect of a system in its project context, SOL provides a coherent vehicle for doing so. Project plans, requirements documents, statements of work, API documents, manuals, background theory, and related web sites can all be directly linked to the relevant elements of the SOL network, and thereby become part of the network. The complex relationships between these disparate documents can be represented explicitly in a coherent and unified way, rather than tacitly in the minds of the project managers, business analysts, and software engineers.

Investigation into extending traditional entity relationship models to capture more of the formal meaning of the data drove towards the notion of a semantic data model (Codd 1979). Over the years this was refined to a Semantic Object Model (Kroenke 1999), as it is now known. SOM introduces a richer ontological classification scheme for entities, properties, and associations, and reformulates entities as semantic objects. It models both data and relationships among data, encapsulates structure, and provides for object inheritance. While it goes beyond classical ER representation, no obstacles prevent a semantic object model from being represented in terms of SOL. Once the SOM is characterized as a generalized SOL ontology, the modeler creates the SOL network in the same fashion as a SOL network based on a traditional ER ontology or a UML class ontology.

Semantic networks are graphs, and can be subjected to topological analysis in a straightforward manner (Staszak 2000b), as well as other types of network analysis. SOL makes it possible to characterize OOD models quantitatively, paving the way for innovative computational design analyses. For example, by examining the structure of the graph, it may be possible to algorithmically determine whether a given object-oriented design exhibits good cohesion. This offers a promising avenue for further research.

## **Conclusion**

In summary, by reformulating UML and ER in terms of a common semantic basis, SOL provides a vehicle for an enhanced, unified approach to modeling software systems that extends UML in an organic, backward-compatible fashion. It also provides a means to leverage the unique strengths and capabilities of semantic networks, both in terms of visualization and knowledge representation, to aid the software design process in terms of clarity, unity, and effectiveness.

## References

- Barker, Richard. (1990) CASE\*Method: Entity Relationship Modeling. Wokingham, England: Addison-Wesley.
- Codd, E. F. (1979). "Extending the Database Relational Model to Capture More Meaning." ACM Transactions on Database Systems, Vol. 4, No. 4: 397-434.
- Fisher, Kathleen M. & Hoffman, Robert. (2002). "Knowledge and Semantic Network Theory."
- Fowler, Martin. (2000). "UML Distilled Second Edition." Reading, MA: Addison-Wesley.
- Kroenke, David. (1999). "Database Processing: Fundamentals, Design and Implementation." (Seventh Edition.) Englewood, NJ: Prentice Hall.
- Semantica Software, Semantic Research Inc., 1055 Shafter Street, San Diego, CA 92106.
- Staszak, Chris. (2002a). "The Semantic Network Picture Book."
- Staszak, Chris. (2002b). "Application of Topological Analysis to Semantic Networks."