Semantic modeling

An overview of the modeling techniques of the public Praxeme methodology

Objective

Semantic modeling is promised a bright future, although maybe under other names. Indeed, whatever we undertake, common sense and judgment lead us to think at this particular level of abstraction:

• To build content-rich services (like in SOA), you must first model the business.

• To merge systems from different organizations or corporations, the only solution is to identify the essential core and isolate the business fundamentals.

• To radically innovate and change processes and organizations, you must escape from current practices, get rid of any bias and identify a new starting point.

• To encourage and sustain ergonomic design, it is better to map GUIs to business objects rather than operations, etc.

Therefore, whatever our goal is and in order to do the right thing, we must first look at the reality and then step away from our usual habits or our cultural conditioning. This is what semantic modeling has to offer.

Content

- What is semantic modeling used for?
- What is semantic modeling?
- Illustrations
- Additional considerations

Author Dominique VAUQUIER

Translator Jean POMMIER, ILOG

Reviewers Carolin CHAI, Atos Origin

Anthony JERVIS, Accenture

Version 1.3, le 27 May 2008

dominique.vauquier@praxeme.org ☎+33 (0)6 77 62 31 75

What is semantic modeling used for?

An anecdote

When we opened an account for the Praxeme Institute, our Treasurer and I were astonished by the following: In order to store our names and address in the system, the bank teller had no other option than opening dummy accounts for each of us, putting one euro on each. At least we had won something! Yet, this is a trivial proof that our IT systems do not always match reality.

Generalization

It would be easy to find other stories to emphasize our point. We all know too well the status of our IT systems in this area: complication rather than complexity, redundancy rather than rationalization. But take a look at the reasons. By default, functional design has driven IT development for the past decades. Due to our culture and education, at least in the West, we think of the world in terms of actions. IT systems and applications have therefore been built as a set of procedures and functions. In the previous story, the procedure is the opening of a bank account. The opening has been decomposed in sub-functions, in a very Cartesian way. Evidently the designer has never looked at the particular needs of an Association and its relations with people and the roles they play in this organization, etc.

Another factor has reinforced, amplified and dramatized the shortcomings of such a functionalist approach: the typical organization selected for IT development activities; that is to say that the Project mode has been widely adopted for software development. Of course this mode has tactical, psychological and economical benefits. But it also has a major drawback: a project addresses a short term goal, a goal which is, itself, expressed in terms of actions for business users and stakeholders. Naturally, a functional approach fits well with this mode and actually reinforces its limitations.

What can we do?

Today we must simplify our systems, address increasing ambitions and master technologies in order to bend them so they can serve the enterprise. Not only do we have to redesign systems but also clusters and systems of systems. In addition to its shortcomings for building legacy systems, the functionalist approach is powerless to address the complexity of tomorrow's mega systems.

How can we elaborate components both content-rich and reusable? How can we merge systems from several companies, entities, parties? How can we continuously adjust the organization and processes of the enterprise without being limited by the complexity of IT systems? How can we enable and facilitate organizational design? How can we properly leverage technology, i.e. not at the expense of the business reality and context?

We are facing huge challenges to preserve the agility and innovation capability of our enterprises, putting their survival at risk.

To address these challenges, we cannot keep using the same approach and thinking or pretending that they will work, eventually. Why would we be more successful than previous generations when they, as opposed to us, could rely on proven processes and formal thinking?

A new way

The public methodology, Praxeme, brings a new frame of reference. The framework is called the Enterprise System Topology. It includes semantic modeling as the initial and most upstream formal representation activity. To support this semantic aspect of the enterprise, Praxeme proposes a process based on object orientation. This is the topic of this article. Although we envision addressing this semantic aspect with other techniques in the future, this is currently out of scope.

What is semantic modeling?

What it is not

What a semantic model is not? A semantic model is not a process or activity model: it lies upstream, in a world that humans do not live in. In which we don't see actors, organizational structures, habits and common practices. When speaking about a semantic model, we use terms like business objects. Yet, many so-called business models or business object models end up by just being limited and poor conceptual models of data. They only capture a small and truncated part of the business. Even their structure is potentially erroneous as we cannot be sure of their stability until the modeling activity has included operations and rules.

Let's immediately put a confusing myth to rest: when you look at a UML class diagram, with classes and attributes, this is not automatically a semantic model. In order to satisfy the encapsulation requirement, operations should be part of the diagram. When you see links—associations—which do not have a name, it means that the semantic has not been expressed. Bottom line, the use of UML or object orientation does not guarantee the semantic aspect of a model.

Furthermore, semantic (or conceptual) is not at all synonym of general. A good and finalized semantic model must be precise, formal, detailed and… annotated with comments. Otherwise it cannot be used efficiently.

What it should be

What is a semantic model? The semantic model represents the objects and concepts used by the enterprise while conducting its business, independent of the means.

The process of semantic modeling elaborated in Praxeme is based on the object-oriented approach and fully leverages the object paradigm. For instance, objects and concepts are represented as classes. A class does not correspond to the concept used in software and programming languages, but the general concept commonly used by philosophers since Aristotle. This is our way to describe our perception of reality.

The model expresses all semantics related to these objects and concepts: information, actions and transformations. The class connects these three modeling dimensions. Simply looking at the information side will lead to a data model, which is not enough. Information needs to satisfy some constraints and is used in calculations. The model encapsulates these constraints and the operations triggering the computations. Classes are grouped into classifications through inheritance and into networks via associations. The process leverages the UML expressivity: associative classes, ternary association, qualified association, derivation, etc.

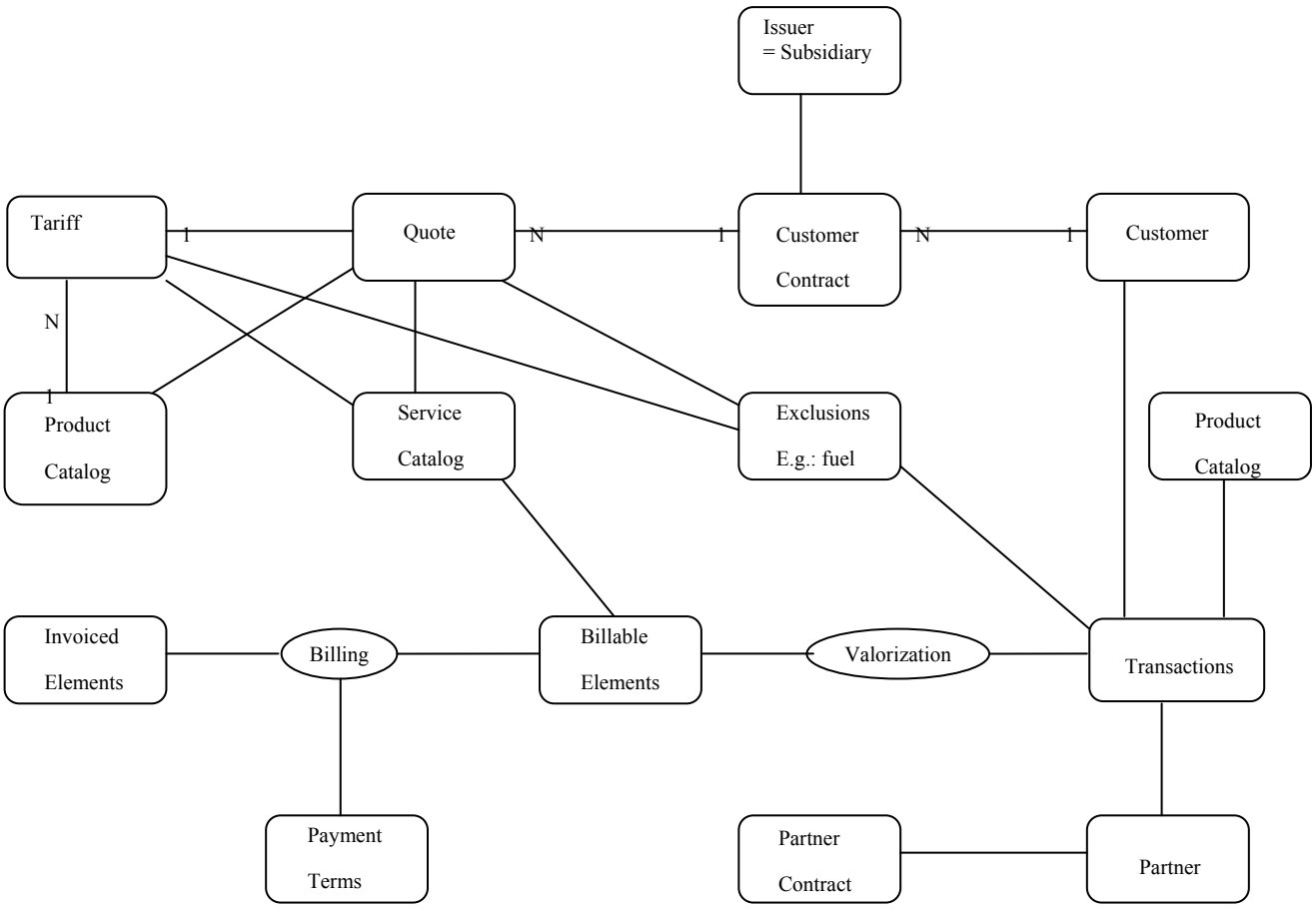
The semantic modeler must master all UML features. Otherwise he/she may produce a limited representation of the business. However, semantic modeling does not equal UML: the UML standard, which has a software connotation, is used as an instrument. What the semantic modeler is after is not Java classes, attributes and operations to implement in an IT system. He looks for information, actions and transformations as semantic characteristics of real life objects and concepts. To achieve this goal, he/she leverages the UML meta-model, reusing some meta-classes to map his/her own representation categories. The UML attribute, possibly a derived one, captures a piece of information. The UML operation captures an action, on or of an object. State diagrams will capture transformations as sequence of operations and transitions.

Illustrations

Simplified model of legacy application

Let's start with a draft of a model using a common approach.

Figure 1. Quick draft (Merise)



This model will be used as a starting point for the discussion below. Boxes are extremely simplified views of the model. The goal of this article is to illustrate the benefits of semantic modeling based on an object-oriented approach. It highlights its positive impact on the quality of the system structure and design.

Genericity

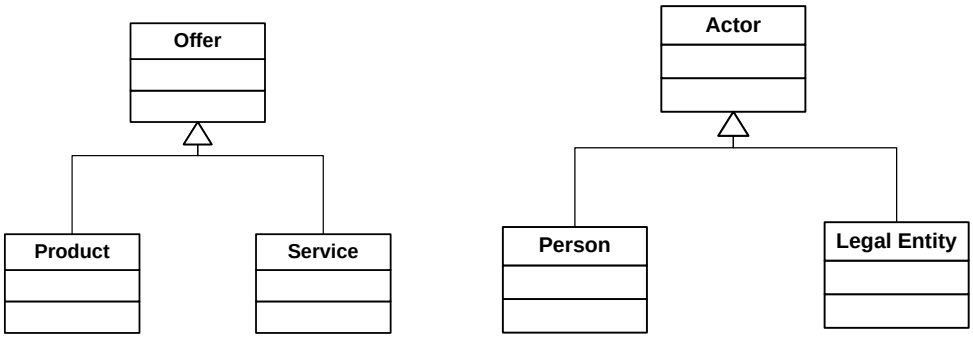
The Object-oriented approach fosters model genericity. For instance, it leads to a simplification of the model through the creation of generic classes such as Offer and Actor, hiding irrelevant additional details about these two key concepts.

Here are some examples of the benefits of this approach:

- Removal of redundancy: the properties (information and behavior) which are shared by Services and Products are only represented once in the model, on the upper class, Offer.
- Structure simplification: the number of associations decreases since the reservation, processing and pricing of an offer apply to both Products and Services but are represented only once, on the upper class.

- Power of the model: the model is more compact and concise; the process of selecting an offer covers both product and service selection. That is a better approach than developing separately a solution for product selection and another one for service selection. Savings are substantial.

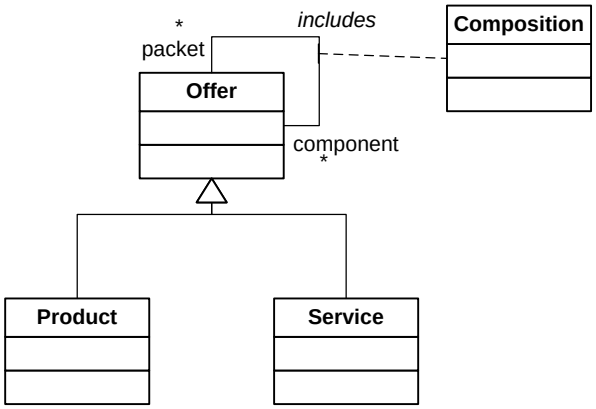
Figure 2. Examples of generic models



Nevertheless, the modeler should not push genericity too far. We too often see models full of inheritance graphs, with dozens of classes which only contain their name. This is a misuse of the notation: in this case the model is used as a semantic network in order to capture all the business vocabulary, inappropriately.

Offer design

Figure 3. Solution to model the combination of offers



In Figure 3 the class diagram captures some of the semantics of the "Offer" through the mechanism of composition (reflexive association categorized by the associative class "Composition" to attach information to the relation). Thanks to this model and the "includes" association in particular, an offer can be either a product, a service or a combination of both (several products, a set of services or an offer combining several products and services). To ease the reading of the model, the modeler named the roles of the association which reads as follow: "a packet includes components."

Multiplicities capture the scalability of the model and show how concepts can be combined to represent real situations. In order to remain as universal as possible, semantic models usually use the most generic multiplicity, i.e. "*" for many . In the event the model needs to be restricted to take into account specific constraints in an enterprise, the best is to implement a MDM solution (master data management) rather than altering the semantic model.

The constraint which prevents an offer from being a component of itself or the one preventing the creation of cycles is encapsulated into the model. No need to express everything graphically: the model is a whole and does not end with the diagram. Structural constraints are actually sometimes better expressed as pre-conditions rather than on the class diagram. All the constraints and business rules should be captured, or at least identified; otherwise, the model is incomplete.

The reflexive association "includes" is represented as a class, here called "Composition". This class is called an Association Class. A semantic model includes many association classes. This formalism allows the modeler to establish a formal definition of pertinent concepts. In the above example, a composition describes a pair of offers, one being the container or packet, the other a component. The structure of the model represents the formal definition of this relation, better than any speech. The association class can, as any other class, bear properties (attributes and operations). It can even be, in turn, associated to other classes.

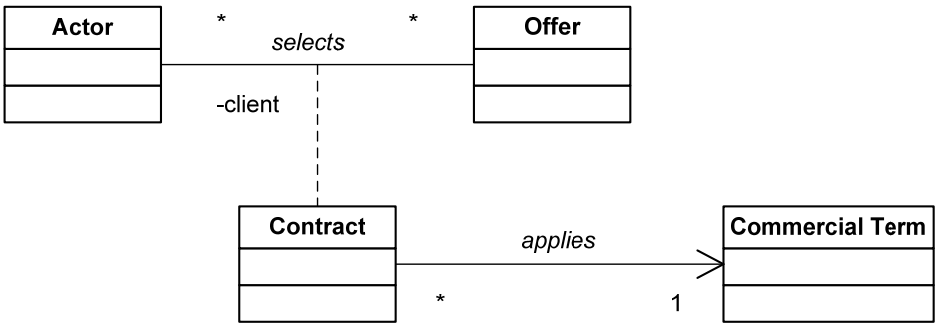
This technique leads to the creation of compact models, able to deal with several levels of decomposition, where, in other common approaches, this would have produced several classes and tables.

Contract setting

Semantic models must be readable. More precisely, the model must provide a way to capture then restitute the context of the verbal description of the reality. UML offers the required expressivity, which the modeler needs to leverage appropriately.

The figure below corresponds to the following statement: an actor selects one or potentially several offers. Within this relation, the actor plays the customer role. Like in the previous example about composition, the select association is reified. Indeed, the act of selecting an offer produces some information which needs to be stored and checked. This way, the model introduces the concept of "Contract". Thanks to this visual representation, the reader knows that a specific contract applies to one and only one customer, and one and only one offer. However, thanks to the composition discussed earlier, a contract can apply to a combination of products and services.

Figure 4. A customer selects an offer



An association class can be linked to other classes, a capability not offered by earlier methods (Merise, SA/SD…). In this example, a contract is subject to certain commercial terms.

In case the modeling exercise identifies structural or content differences between contracts across the various partners and parties, it will be easy to extend the model accordingly with sub-classes of the Contract class. Such extension will not impact what has already been captured around the concepts of actors and offers.

The model « speaks » to us, it tells a story through the proper naming of associations and the roles on the associations. Moreover, association classes are leveraged to capture information on how concepts relate to each other.

Pricing

In legacy systems you often find pricing information spread across numerous tables (recently, although across three countries, I identified no less than 20 fields for this purpose).

Semantic modeling is reluctant to include such redundancy. Based on common sense, a term should only appear once and only once in a model. If you see the same term used in two different settings, you are facing one of the two following options:

- Either the term means two different things, in which case the modeler must find another term for one of them;

- Or the term describes one single concept indeed, and the model needs to be restructured to only use and display one instance of the term.

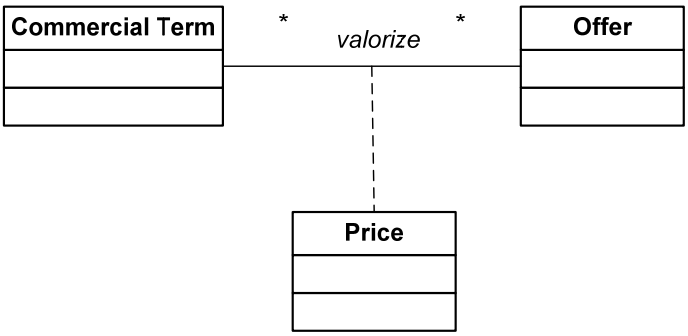
In our model, "Price" will appear only once, like any other concept or information for that matter. Pricing occurs for the couple: Offer and Commercial Term. This is captured by the valorize association.

Multiplicities indicate that one offer can get different pricings, depending on the commercial terms. Conversely, one set of commercial terms (e.g. standard catalog) can apply to several offers.

The number of n-ary associations and association classes is proportional to the model expressivity, one quality of a semantic model.

UML allows setting a direction on an association. However, this feature is rarely used in semantic modeling because it restrains the model navigation. In our example, it would be easy to orient the association following the way we read the diagram; allowing navigation from Commercial Terms to Offer, but not the other way. Such a restriction has, more often than not, no semantic associated to it and would only limit the possibilities of a system derived from such a model. Such limitations actually fall into the scope of logical modeling, a subsequent step of the modeling activity chain.

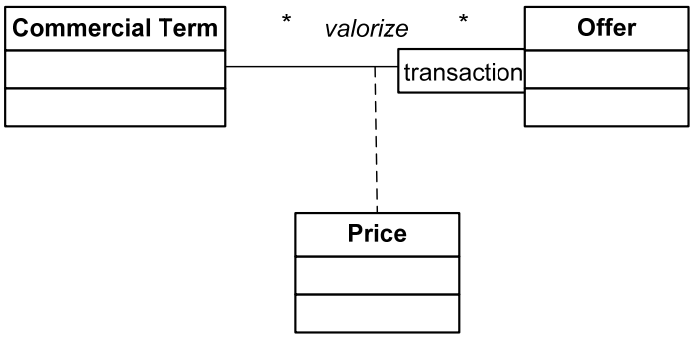
Figure 5. Only one way to price an offer



Semantic models are characterized by two things: the fact that terms are used only once in a diagram and the conciseness of the model. With this approach, and as opposed to common ones, a term must *only be used once.*

Model generalization

Figure 6. A qualified association



The only difference between Figure 5 and Figure 6 is the box on one branch of the association. In UML terms we call this box a Qualifier. Qualifiers extend the expressiveness of the modeling language.

For instance, in our example, the transaction qualifier can be either purchase or sale. With that, for a given offer and based on the value of the qualifier, a set of commercial terms will lead to a certain price. This way, the model allows the storage of both the price of an offer and the one at which it got purchased, both for products and services, and splitting these prices in two distinct subsets. Suppliers can be characterized by the Actor class and its subclasses. Incidentally, this avoids from duplicating information when an actor is both a supplier and a customer:

it is recorded only once, along with its specific role, as a natural person or a legal entity. The role of supplier or customer is then determined through its structural connections with the other objects.

Such a straightforward outcome of semantic modeling is completely the opposite of a functional approach. The latter would have led to two steps and two forms: "build the catalog (for sales)" and "manage procurement/supply". Each of these two needs would have likely be addressed by two distinct projects and, unless of extreme precautions, lead to two different databases.

When executed upfront, semantic modeling prevents such divergence to happen and, subsequently, simplifies the derived systems. In addition, it standardizes terminology, glossaries and representations.

UML expressiveness is usually not used to its full potential. UML is actually a valuable instrument for semantic modeling and the formal representation of knowledge. Qualified associations in particular are an elegant way to express constraints and increase the applicability of models. Without much overhead and attention, semantic modeling prevents duplication and redundancy of functions; Two common flaws of a functionalist design.

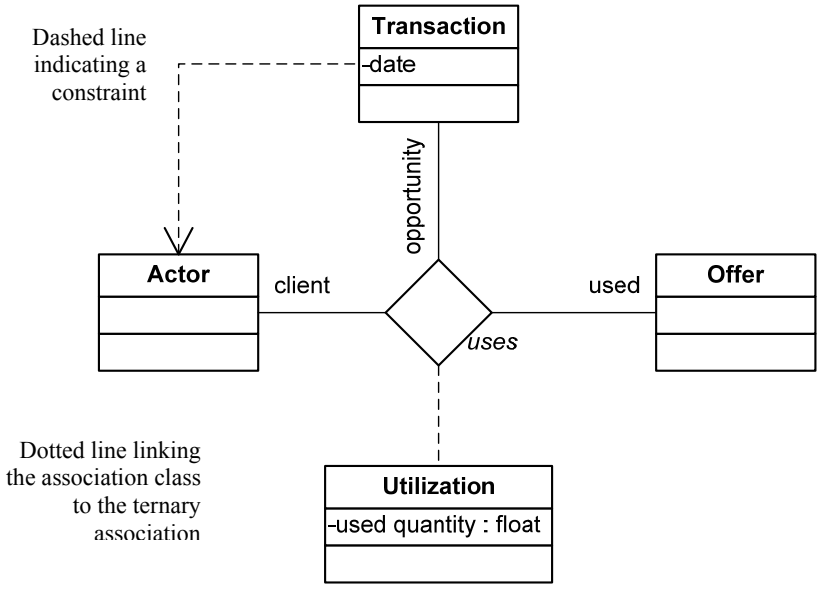
Utilization

A customer uses products and/or services. The use of customer in this context seems to violate the uniqueness rule for the terms. However, and albeit informally, what we express here is the constraint preventing an actor to use a product or a service of which he/she is not already a customer, with the corresponding contract in place.

If the uses association were binary (i.e. only between Actor and Offer), it would allow only one use, by one (actor, offer) pair. This would quickly lead the company to bankruptcy! To address this, the modeler needs to introduce a third term: Transaction, which corresponds to a date or a period… This relation, a ternary association, is represented by a diamond. It means that we do not create only pairs but triplets: an actor uses an offer through a transaction.

The use of multiplicity on ternary associations requires some explanations. From a graphical standpoint, it seems that a transaction could involve several offers and only one actor, but is it really the case? The question can be answered with a constraint (dotted arrow) to which the modeler will attach a note or a formal expression (using OCL, for instance).

Figure 7. A ternary association



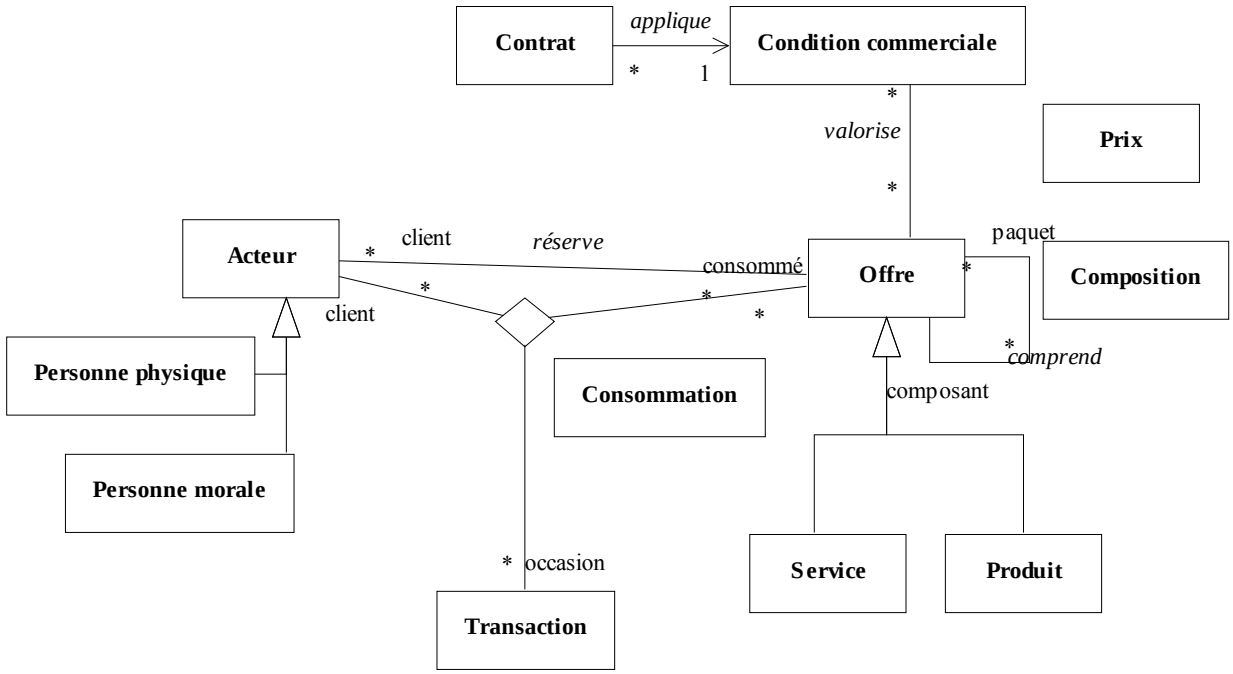
Speech and natural representation involve many concepts, themselves defined by other concepts. Such concepts are formalized with categorized associations, with as many branches as there are parameters. Semantic modeling highlights these parameters: they are essential for a correct understanding of the

scope of the model. A less rigorous model would dilute these parameters into groups of binary associations.

Summary

We briefly introduced the key concepts pertaining to modeling. This model is simple but provides the structure for the overall model. We can now compare it to the original figure 1, which was the outcome of a common data-driven approach. To make our point, we only looked at the structural dimension of semantic modeling. Beyond additional or secondary classes, the complete model will include the definition and documentation of all the class attributes (information) and operations, as well as all the constraints.

Figure 8. The overall class diagram, result of our modeling exercise



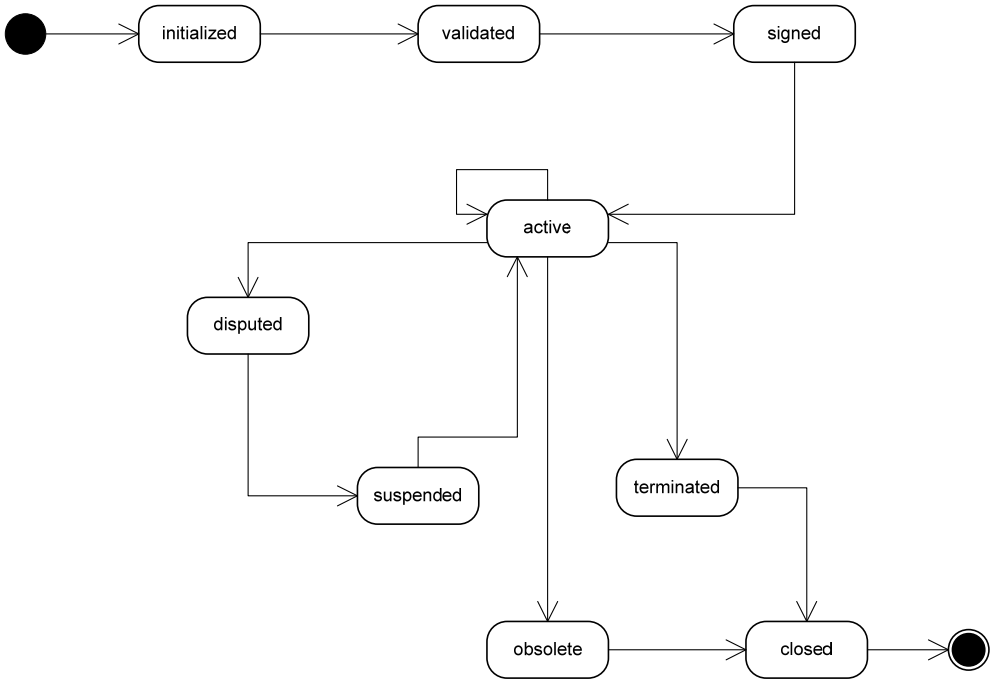
Additional considerations

The previous section triggers two follow-up discussions:

1. Semantic modeling does not end with a static description of the universe. It encompasses all the core business knowledge. Such knowledge includes information (data) and behavior (actions). It covers business rules which have a semantic nature. It excludes rules which depend on the organization or which are technical. The next section illustrates the dynamic dimension of the model, leveraging the concept of state machines.
2. Around this core or along this spine, the model will grow to capture more business specifics and details. It can grow from a dozen of classes, like in our example, to one hundred or more. To address this scalability need, UML provides the concept of package. We will use this feature to structure objects into domains in a subsequent architecture diagram.

State machines

Figure 9. An example of (informal) state diagram for the Contract class



The above diagram provides an example of an informal use of the concept of state machine. The Object Oriented approach aims at designing self-contained and highly coherent components. Concretely, objects are responsible for their state. Whatever action triggered on an object, we know that the object will behave accordingly to its constraints. Of course such a benefit is only realized thanks to a rigorous modeling exercise.

We are dealing here with the contractual dimension of semantic modeling (the other two are: the structural modeling which we discussed in the first part of this article, and the functional modeling, which describes the content of operations and processes). To capture this contractual dimension, UML provides a very powerful paradigm: the state machines represented as state diagrams. This paradigm is largely underutilized especially in non technical IT applications (e.g. accounting).

State machines are required during modeling in two cases: firstly, when adjectives are used to qualify a concept or, secondly, when we mention milestones or states in an object's life cycle. The modeler first identifies the states which are usually named using adjectives applying to the name of the class. He/she then connects these states using transitions, respecting transition constraints. Leveraging the 2 dimensions of the diagram, instead of only a one-dimension sequence, the modeler will be more keen to take into account potential disruptions or loops in the object life cycle. Such ability will lead to a more fluid and richer model. Being better prepared to any expected event, the model will also be more robust.

Finally, to complete the model, the modeler analyzes the triggering and execution of the transitions. In doing so, he/she highlights events, conditions and operations. This is one reason why non technical IT applications have trouble properly identifying the operations at a conceptual level. The state machine is the right way to identify such operations.

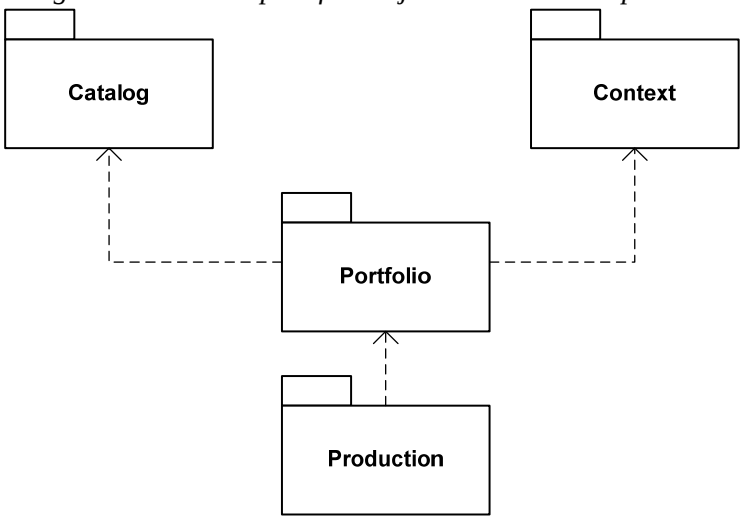
To describe transformations and identify operations, the semantic modeler leverages states machines. Such a new technique has a huge impact because it is a perfect way to take transformations into account. In common approaches, transformations are either ignored or pushed into procedures which then lead to a key source of complication.

Object domain decomposition

Decisions about the model structure have long term consequences. Thus, the modeler must think twice and thoroughly before selecting a decomposition criterion. Because both theory and pragmatism have proven that functional domains lead to redundancy and inadequate coupling, we will not use them as a decomposition criterion, at least not to structure the semantic model or the business frame of reference. Praxeme recommends a structure based on object domains, at least for the system's core, for the business layer.

The following package diagram is a draft of the logical architecture. It is an unnecessary constraint in a semantic model because the relations restrict and limit the navigation. For instance, such formalism imposes to set directions on certain associations. However, as the model grows, modelers need this formalism to facilitate team work. The good news though is that such diagrams lay the foundations for the transformation of the semantic model into a logical model, for instance, in preparation for a service-oriented architecture.

Figure 10. An example of an object domain decomposition



At the semantic level, the modeling scope gets organized into « object domains ». This leads the way to the Logical Architecture. By introducing object domains in addition to the traditional functional domains, Praxeme brings a radical change to the structure of IT systems.

Conclusion

The modeling options and techniques highlighted in this article are customary for semantic modeling experts. We limited the scope of the article to the structural quality of the model. One should also consider:

- The definition of the modeling scope through a pre-modeling exercise,
- Encapsulation of business rules,
- The specification of operations at the semantic level,
- Design patterns,
- Etc.

Semantic modeling is a demanding discipline and exercise. It captures the business core concepts and the essence of the business in a way which can be leveraged by the downstream analysis and implementation activities: it translates target market definitions and strategic goals, it instills an innovative way to design processes, and it lays the ground to the design of rich and highly-reusable services.

Our ability to create real modeling skills will directly impact the success of tomorrow's projects and the realization of our ambitions (e.g. IT convergence, agility, reuse, innovation). The inherent cultural change requires the support of the top management. All successful semantic modeling initiatives share one factor: an assertive leadership to mitigate potential resistance, to overcome the obstacles—cultural or psychological—, and to valorize the real impact and benefit for the enterprise.

dominique.vauquier@praxeme.org

Bibliography – References

On the Praxeme Institute website (http://www.praxeme.org):

- *Le Guide de l'aspect sémantique*, référence PxM-10, par Dominique VAUQUIER, exposé du procédé de modélisation ➔ translation in progress.
- *Modélisation sémantique*, support du module de formation en deux jours, contribution d'Unilog Management avec des illustrations tirées de plusieurs expériences de modélisation ➔ partly available on http://www.sustainableitarchitecture.com/.

Books:

- *Resilient Information System, Progressive Recasting with SOA*, Pierre Bonnet, Orchestra Networks, France, Jean-Michel Detavernier, SMABTP and Dominique Vauquier, Praxeme Institute, 2008, Wiley.
- *Object Engineering – The Fourth Dimension*, Philippe DESFRAY, 1994, Addison & Wesley.
- *Développement orienté objets, Principes, processus, procédés*, Dominique VAUQUIER, 1993, Eyrolles.

▲