Fitting the Pieces of the Web Engineering Puzzle

Oscar Pastor
Technical University of Valencia
Department of Information Systems and Computation
e-mail: opastor@dsic.upv.es

Abstract

As a modern version of Software Engineering, Web Engineering methods, notations and techniques want to provide a final software process with the required quality. To accomplish this goal, Web Applications must have a precise semantics. There are lots of approaches intended to provide a final solution. For instance, in current Web Engineering environments, we have Web Conceptual Schema specification techniques, Semantic Web-based languages where the Web Application meaning is properly captured, Web Services definition and specification standards... A too complex set of pieces for a puzzle that tries to solve an old problem, now within the Web world: how to define a Software Process for developing correct and complete Web Applications, from Requirements to the final implementation. In this talk issues related to Semantic Web, Conceptual Modeling, Web Services and so on will be analyzed in a unifying way, introducing the required set of relevant conceptual primitives that need proper representations, including those data and functional more conventional primitives, and those navigational and presentation-oriented more specific of Web Apps. The final intention is to fit all these pieces required to have a precise view of what modern Web Engineering has to be.

1. Introduction

The emerging discipline of Web Engineering is the last attempt to solve a well-known problem in the context of Software Engineering: how to elaborate a correct and complete software product, according to the customer requirements. Different methods and techniques have been proposed in the last decades to try provide the holy grail that would solve the problem, but all of them, one after the other, have not succeeded. A lot of work and illusion are being put in this new discipline, and there is really a very interesting set of proposals that allow us to dream –again!- that we have it. We have to try it. But to be successful, one of the most serious current problem is to put order in this set of different technologies that too many times seem to be “doing the war” independently. Web Services, Semantic Web, Ontologies, Conceptual Modeling, Business Process Languages for the Web... constitute a family of technologies that need to be put together, pushing in the same direction: to fulfil the goal of having a precise Software Production Process, now for the emerging Web Applications world. This is why it is strongly required to fit all these pieces in a unified framework.

It is clear that we don’t face at all a new problem. Philosophers are facing for centuries the problem of how to represent properly reality. The existence or non existence of universals -concepts whose representation we can perceive- has been an issue from the platonic realism –that defend the existence of universals- till the more modern nominalism and conceptualism philosophical approaches –arguing that universal or concepts are just a linguistic mechanism to model real world phenomena-. In some way, this philosophical discussion has been present in Software Engineering through Conceptual Modelling approaches. We could see Conceptual Modeling-based methods as a projection on Software Engineering of realism, in the sense that the only
software components that are present in a final product are those that have a conceptual counterpart in the corresponding source conceptual schema. Agile Methods, Extreme Programming-based would constitute the nominalism approach, not accepting the need of having previous, pre-existing modelling constructs as a required basis for any software representation at the solution space.

Within the emergent Web Engineering community, model-based approaches are providing sound methods to deal with a precise Web Application Production Process, where the features associated to system structure, dynamics, functionality, navigation and presentation are properly managed (OOHDM [7], WebML[1], OOWS[2],…). Furthermore, how to go from the conceptual schema (system specification) to the implementation (final software product) is precisely stated by defining a set of mappings between conceptual primitives and their corresponding software representations (OlivaNova [6]). Some concrete tools are even already present in industry, giving some kind of automated support to this web-oriented software production process.

According to these approaches, any Web Application is the result of systematically applying a set of transformation rules specified at a higher level of abstraction in a Web Conceptual Schema. If Web Applications would have been built from the beginning following these ideas, the semantic of any Web Application would be precisely characterized by its corresponding specification. Obviously this has not been the case. Generally speaking, Web Applications Development in practice has been during the last years and ad-hoc, informal process, where modelling support has not been considered at all an essential approach to deal with the complexity of Web development. In consequence, we face a situation where a huge number of Web Applications are running in the Web, with a mostly unknown semantic structure, and all of them independent from each other.

But as we commented before, humans are insistent in trying to structure the world. The World Wide Web has not escaped to this human goal. If we want to communicate Web Applications providing efficient web services to exploit the advantages of the global web, the semantic of a Web site needs to be precisely known. Semantic Web languages are introduced to represent Web site modes. They play the role of the Conceptual Schema in the Model-based Web Development approaches provided in the context of the most advanced Web Engineering methods. Semantically tagged data start to become available: the Semantic Web technology is just here.

But it is very interesting to remark that Conceptual Modelling for Web Applications and Semantic Web related technologies are facing a common, well-known problem: to understand the world, by providing a clear system specification. Conceptual Modelling selects a top-down strategy –from the model to the implementation- while current Semantic Web technology opts for a bottom-up approach –we have the software product: let’s provide any kind of structured specification in a clear-enough language.

According to this idea, a basic aspect is to characterize the set of conceptual primitives required to model a Web Application. Either if we chose a top-down or a bottom-up approach, the required conceptual primitives should be the same. For representation purposes, different languages can be selected to specify them, but the important point is to describe precisely those conceptual constructs needed to characterize the structure of any Web Application. In this paper, we basically introduce such a set of conceptual constructs, independently of any particular language or conceptual modelling approach. Our final intention is to characterize the expressiveness that has to be provided by any particular solution, either coming from the Conceptual Modelling or the Semantic Web domain.

In this context, Web Services Description Languages (WSDL) will play its role in a clear and precise way. However we face the modelling phase, we have to specify the precise
semantic of services, usually in the class scope. WSDL can be just the way of facing this service specification problem in a unified way. Summarizing this introduction, if concepts are precisely defined, every piece will fit in our puzzle. To accomplish our objective, after this introduction we present in the next sections the quoted set of conceptual primitives: in section 2 we introduce the conventional static and dynamic primitives, following an Object-Oriented Model to characterize the data and functional system architecture. In the section 3, the navigation and presentation conceptual primitives that complement the previous static and dynamic system views are presented. The work is ended with the conclusions and the corresponding references.

2. Conceptual Primitives related to Data and Functionality

Of course, a Web Application is still an application… What we mean by it is that beyond specific web-oriented aspects that we will face in the next section, a Web Application must be based on a precise Class Architecture –to characterize the static system’s view- and a precise Functional Model –to characterize the dynamic system’s view. We chose an Object Oriented Model for Conceptual Modelling purposes because it has been proved in many previous works that the OO Model is especially appropriate for conceptual modelling due to its proximity to human cognitive mechanisms. It seems to be a natural way of modelling to look at the world as a society of interacting objects, belonging to classes where data and functionality are formally specified. The question is now to fix what conceptual primitives need to be taken into account.

From the static point of view, the list of conceptual primitives is composed of:

- **classes**
- **relationships** between classes

The class specification includes the definition of class **attributes** and class **operations**. Every attribute has associated its type, its default value, if it is a constant, variable or derived and if it accepts null values. For every operation, its arguments must be specified, together with a special label to distinguish new and destroy operations, and shared operations with other classes when this is the case.

The valid relationships between classes are those of **association** / **aggregation** and **inheritance**. Association / aggregation are characterized by the binary cardinality (minimum and maximum) and by the constant or variable property of the established relationship. If the relationship is unidirectional, the induced part-of relation converts the association in aggregation.

Inheritance conceptual primitives include the specification of roles as specialization that gets activated only in periods of a given object live. The condition or operation that activates the role, and the condition or operation that deactivate it has to be specified.

Finally, **integrity constraints** allow specifying conditions that must hold in any valid state of an object. They are specified within the class scope as well-formed formulas built on class attributes.

From the dynamic point of view, the list of conceptual primitives is composed of:

- **preconditions** of operations, to state what conditions must hold for activating an operation;
- **valuations** of operations, to state what is the change of state generated by the occurrence of an operation, in terms of new attribute values or object creation / destruction;
- **transaction** definition, including local and global operations consisting of a set of operations belonging to the same class (if the transaction is local) or belonging to different classes (if global);
trigger specification, to fix when an operation will be activated in an automated way, because a given condition is fulfilled. These primitives have traditionally been present in a sound model-based software development process, and they need to be present in particular if we want to provide a method for developing Web Applications. It is in this context where Web Services Description Languages play a basic role. Often WSDL is used in a generic way, to describe services without fixing before a precise semantics for them.

If the dynamic, functional primitives are formally defined, WSDL can be used in such a precise way, because the basic primitives are defined with no ambiguity. Dynamic logic provided the required formal support to determine the meaning of service preconditions, valuations, triggers and the definition of transactions and processes. With such a basis, WSDL can be used as a standard language to specify services in a way understandable to any reader.

As we will see next, another important point is that functionality and navigation are clearly separated. This is important because too many times the specification of functionality and the specification of navigation are somewhat mixed in a way that lead to a fuzzy global specification.

But the data and functional specification is not all. A Web Application needs to specify particular navigation and presentation characteristics, specific of Web environments.

3. Conceptual Primitives related to Navigation and Presentation

It is not an easy task to define navigation, as there is no general definition accepted by everybody. According to interesting discussions hold within previous IWWOST editions (International Workshop of Web-Oriented Software Technology, [3], [4], [5]), our position is that navigation implies the change of a conceptual node through the activation of a navigational link. This implies that what do we mean by conceptual node –interaction unit that provides access to relevant data and functionality for a given agent- and navigational link –reachability relationship between conceptual nodes to satisfy a given agent’s goal- is basic for characterizing a navigational model.

For navigational purposes, we assume that any valid navigation must be accomplished by traversing a path that exists in the underlying class model. This means that we can navigate from one class to another if and only if there is a relationship specified between the involved classes.

The navigation specification must fit the features of particular agents. In consequence, the main navigation conceptual primitive is the navigational map that will be attached to any particular agent type. The navigational map represents the valid paths that any agent of the corresponding type can go through.

The other primitives are hierarchically structured. Any navigational map is made up of:

- **navigational nodes**, that includes a set of navigational contexts, that are the basic user interaction units, containing a set of navigational classes and navigational relationships;
- **navigational links**, that are binary relationships specifying a reachability relationship between two navigational nodes.

Any Web Conceptual Modelling or Semantic Web based approach has to provide the way to specify the required specific properties of both navigational classes and relationships. A navigational class includes the set of attributes and the set of operations that an agent can access. These accessible properties must exist in the structural class diagram introduced in section 2. This allows us to define a navigational class as a view of a class, where the subset of visible and accessible class properties is specified. As not all of the class
population has to be available, additionally filters on the class population can be defined associated to any navigational class.

Finally, a **navigational relationship** is defined as unidirectional, binary relationship that exists between two navigational classes of a given navigational context. They need to have a structural relationship counterpart in the associated Class Diagram. Depending on if the navigational relationship induces or does not induce navigation, we have navigation relationships of two different types.

If the navigational relationship does not imply navigation, we are just adding more information to the basic agent interaction unit that the navigational context is. If it implies navigation some more properties have to be specified:
- which is the target navigational context
- which is the attribute that will be used as “anchor” for activating the navigation

These are the most basic conceptual navigation primitives that must be provided. We also talked about presentation patterns. Now, we briefly introduce a set of conceptual presentation patterns, intended to complement the navigational view by specifying some presentation properties. These properties will also guide the user interaction provided by the Web Application.

The conceptual presentation patterns are basically:
- **information layout** (register, table, tree, master-detail, etc.)
- **ordering criteria** to indicate the chosen order to view the required information
- how to **group the visualization** of object (page cardinality, access mode)

With them, how the user will “see” and interact with the information provided at any navigational step, is precisely specified.

### 4. Conceptual Primitives Representation: Conceptual Modelling vs. Semantic Web

Once established the set of conceptual primitives that must be captured to properly specify the requirements of a Web Application, we must follow an approach to tackle with the software development process.

From a Conceptual Modelling point of view, the representation of those concepts must be defined before system implementation. At a higher level of abstraction, any conceptual modelling approach must provide the required graphical notations to represent the system requirements by using those conceptual primitives. Usually these primitives are organized into different diagrams. In an OO paradigm, we use a Class Diagram to represent the structural system properties (see Figure 1) and a Functional Model to represent the dynamics (see Figure 2).

The widely accepted, standard notation provided by UML is the needed tool to avoid any wheel reinventing syndrome from the notational point of view. The own standard provides some interesting modelling capabilities to extend its expressiveness when required. In particular, any primitive that is not present in the simplest form of the UML standard can be included through the definition of the corresponding stereotype. Once a stereotype is precisely defined syntactically and semantically, it becomes a new modelling element of the notation. The important process is to define always first the concept; next, its graphical representation. In the next figures, all the previous conceptual constructs are included in the selected, base UML diagram (Class Diagrams) as the corresponding, precise UML stereotype
In addition, the navigational primitives are represented by means of a Navigational Model (see Figure 3) and the presentation primitives within a Presentation Model (see Figure 4).
From a Semantic Web point of view, the representation of those primitives must be placed at implementation level. Development strategies oriented to apply the semantic web use implementation Semantic Web Languages (such as OWL [8]) to take into account those conceptual primitives. The specification of these properties is usually specified in schema files defining the valid implementation structures.

Semantic Web Languages provide a set of language constructs (classes, properties, datatypes, etc.) that let us to define the whole of primitives presented before (static, dynamic, navigational and presentation primitives) by means of an ontology. Then, we can define each of those primitives characterizing them by a set of attributes and relationships with other primitives. These descriptions will let to publish and share ontologies on the Word Wide Web.

Although there are different syntactic forms to write an OWL ontology, here, we are going to present the examples using the RDF/XML Syntax Specification. First of all, figure 5 show some of the Static Primitives defined previously in this section. These are described by class axioms that contain additional components that state their characteristics.

```
<!-- Static Primitives -->
...
<owl:Class rdf:ID="Class"/>

<owl:FunctionalProperty rdf:ID="class_name">
  <rdfs:domain rdf:resource="#Class"/>
  <rdf:type rdf:resource="&owl#DatatypeProperty"/>
  <rdfs:range rdf:resource="&xmlschema#string"/>
</owl:FunctionalProperty>

<owl:ObjectProperty rdf:ID="class_operations">
  <rdfs:range rdf:resource="#Operation"/>
  <owl:inverseOf>
    <owl:ObjectProperty rdf:about="#op_belongs_to_class"/>
  </owl:inverseOf>
  <rdfs:domain rdf:resource="#Class"/>
</owl:ObjectProperty>

<owl:Class rdf:ID="Attribute"/>

<owl:ObjectProperty rdf:ID="class_attributes">
  <owl:inverseOf rdf:resource="#att_belongs_to_class"/>
  <rdfs:domain rdf:resource="#Class"/>
  <rdfs:range rdf:resource="#Attribute"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="att_belongs_to_class">
  <rdfs:domain rdf:resource="#Attribute"/>
  <rdfs:range rdf:resource="#Class"/>
  <owl:inverseOf>
    <owl:ObjectProperty rdf:about="#class_attributes"/>
  </owl:inverseOf>
  <rdf:type rdf:resource="&owl#FunctionalProperty"/>
</owl:ObjectProperty>

<owl:Class rdf:ID="Operation"/>

<owl:FunctionalProperty rdf:ID="operation_name">
  <rdfs:range rdf:resource="&xmlschema#string"/>
  <rdfs:domain rdf:resource="#Operation"/>
```

Figure 5.- An excerpt of the Static Primitives

Next figure shows an example of use of the Static Primitives recently defined. This instantiation is based on the model drawn in figure 1. Note that the Static Primitives (terms) are presented in bold face.

<!-- Class3 definition -->

...
Again, as we did with the Static Primitives, figure 7 shows the description of some of the Navigational Primitives across class axioms. These primitives provide knowledge to Web data consumers to know the different possibilities to access some specific content (for a specific kind of user), mainly to assure they make a proper use of data.

```xml
<owl:Class rdf:ID="User"/>

<owl:ObjectProperty rdf:ID="cdm_User_Class">
<rdfs:label>cdm:User_Class</rdfs:label>
</owl:ObjectProperty>

<owl:Class rdf:ID="Nav_Subsystem">
<owl:Class rdf:ID="Nav_Node"/>
</owl:Class>

<owl:Class rdf:ID="Nav_Context"/>
</owl:Class>

<owl:Class rdf:ID="Nav_Class"/>

<owl:ObjectProperty rdf:ID="cdm_Class">
<rdfs:label>cdm:Class</rdfs:label>
</owl:ObjectProperty>

<owl:Class rdf:ID="Manager_Nav_Class">
<owl:Class rdf:ID="Complementary_Nav_Class"/>
</owl:Class>

<owl:FunctionalProperty rdf:ID="cdm_Operation">
<rdfs:label>cdm:Operation</rdfs:label>
</owl:FunctionalProperty>
```
Finally, based on the Navigational Primitives represented in the corresponding Ontology, next figure shows an instantiation of the Navigational Model taking figure 3 as example. Let’s remark that following this strategy, the semantics of any Navigational Model of a given Web Conceptual Schema can be easily interpreted, because the used properties are precisely fixed by the set of conceptual primitives required to specify any Web Applications Model. These required primitives are just those presented in this work.

A last aspect to be commented is how to introduce properly the Web Services “metaphor” in all this OWL, ontology-based framework. In principle, there is an elemental connection. As the www consortium Web Services Description Language proposal (WSDL) [9] is intended to specify in a precise way the semantics of any web service, WSDL could be use
to represent syntactically the concepts underlying the corresponding base ontology, supposed to be written in OWL.

In fact, OWL is giving some concrete answers to this questions, through the definition of the OWL-S extension ([10]), specially oriented to Web Services Specification. As a successor of the DAML (DARPA Agent Markup Language) initiative for web service automation, the language is built upon OWL and it has three properties to describe a service:

- The ServiceProfile and ServiceModel are the abstract representations of a service, and…
- The ServiceGrounding deals with the concrete level of specification.

The service profile describes the capabilities and parameters of the service. Basically it answers the questions “what does the service do” and “what does it require”. It is interesting to remark that the answer to these questions is fixed by specifying the basic, required conceptual primitives associates with a service in the context of a Web Applications, as we have done before. Complementary, the service model describes what happens when the service is carried out. More concretely, it describes how the service works by specifying the workflow and possible execution paths. The service grounding explains how the service can be accessed and used.

The interesting conclusion is that OWL and its OWL-S extension can be used together to specify the semantics of a Web Application, considering that the concepts required to specify any Web Application are precisely fixed and defined in advance. Furthermore, in this way the use of both proposals is determined, in the sense that the structure of any specification is based on the use of the predefined basic building blocks of any Web Application. As an extravalue, there is no ambiguity in deciding what kind of resources or components need to be included in a particular specification, that becomes understandable to any user.

Summarizing this section, we have a strategy accepting different representations but based on a common set of basic concepts. Having a fix set of conceptual primitives, it is feasible to define a set of mapping between conceptual primitives and their corresponding software representations, making possible the implementation of Web Conceptual Model Compilers.

In this environment, a fruitful strategy could be based on taking the best of the two approaches by:

- having complete conceptual models of web applications, at a higher conceptual (problem space) level, and
- implementing final applications by applying a set of systematic translation rules from those conceptual primitives into web semantic concepts representation, at the solution space level.

5. Conclusions

The main goal of the emerging Web Engineering discipline is to develop correct Web Applications, where structure, functionality, navigation and user interaction have to be properly represented. To do it, any Web Application has to provide a precise semantic associated to it. Only if such a precise meaning is given, it makes sense to provide web services whose structure and functionality is clearly specified, and that can be accessed and used by different agents.

This semantics can be provided in a top-down way, by defining a Web Conceptual Schema where all the relevant modelling components are specified. The resulting Web software
product is the corresponding representation of the Conceptual Model at the solution space level.

Alternatively, a bottom-up strategy can be used. In this case, a Semantic Web-based language (i.e. OWL) allows to specify those relevant conceptual constructs that characterize the meaning of the corresponding Web Application. This specification makes possible the connection of the application to any external potential agent. Web site models can be represented in this way by Semantic Web languages. The available Semantic Web infrastructure is immediately applicable for the Web Engineering field, thus making the processing of Web site models effective.

This way of working opens a very interesting door to implement full web model compiler, intended to transform a source Web Information System model into its corresponding final software product in the form of a Web Applications. The strategy is to associate a precise software representation to every web conceptual model primitive. The precise implementation of this set of mapping provides a formal approach to build model compiler that could create a Web Application from its corresponding Conceptual Schema in an automated way, using the most advanced Web Engineering technologies within the current widely accepted MDA framework.

In any case, the set of conceptual primitives required to fix the semantics of a Web Application must be clearly defined. In this keynote, this set is introduced. They are structured in data and functional conceptual primitives, and more specific, purely web-oriented navigational and presentation conceptual primitives. The final intention is to fix the required expressiveness for any Web Conceptual Modelling strategy, or any Semantic Web-based ontology language. According to that, we could conclude that Conceptual Modelling and Semantic Web are really the two sides of the same coin: the coin required to develop correct Web Applications. And we could also conclude that the main pieces of Web Engineering constituted by Web Services, Semantic Web and Conceptual Modelling of Web Applications can –should- be put into work in a unified way to make the pieces of this exciting Web Engineering puzzle fit.

References
