The Object with States Ontology Design Pattern

Raúl García-Castro and Asunción Gómez-Pérez

Ontology Engineering Group, Center for Open Middleware
Universidad Politécnica de Madrid, Spain
{rgarcia,asun}@fi.upm.es

Abstract. This paper describes the Object with States content ontology design pattern that allows modeling the different states of an object and the restrictions on such object for its different states. It also presents an example of applying the pattern in a concrete use case in the ALM iStack ontology.

1 Introduction

An object can have different states over time for which different restrictions apply. Examples of objects with different states can be persons (which can be single, then married to another person and later become divorced or a widower) or research papers (which can be submitted for publication and after a round of reviews they can be accepted or rejected).

The goal of the Object with States content ontology design pattern described in this paper is to allow modeling the different states of an object and the restrictions on such object for its different states.

It is out of the scope of this pattern to model other information about object states such as: the time intervals in which an object is in a concrete state, transitions between states, or relationships or dependencies between states.

2 The Object with States Ontology Design Pattern

The Object with States ontology design pattern is a content pattern that aims to satisfy the following ontology requirements:

- Objects must have a unique state.
- Object states must belong to a single collection of non-duplicate elements (i.e., to a set).

Figure 1 depicts the Object with States pattern, which is available online as a reusable OWL ontology[1]. As can be seen, the pattern contains three classes, one for representing objects, another for representing object states, and a third one for representing sets of states. Besides, it contains object properties for relating objects and states (which are subproperties of those defined in the *Situation* pattern[2]) and for relating states and

[1] http://delicias.dia.fi.upm.es/ontologies/ObjectWithStates.owl
[2] http://ontologydesignpatterns.org/wiki/Submissions:Situation

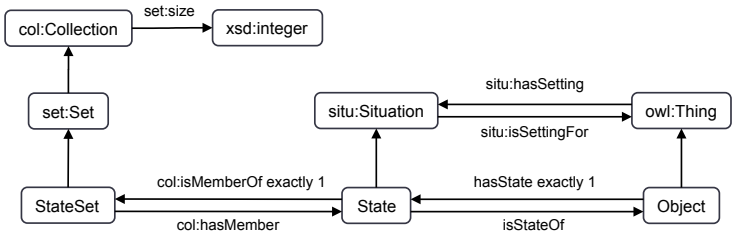

Fig. 1. The Object with states pattern.

sets of states (reused from the *CollectionEntity* pattern[3]) and a datatype property for defining the size of a set of states (reused from the *Set* pattern[4]).

Taking into account some fictitious classes and properties, which serve as an example of how to instantiate the pattern, the following ontology requirements can be satisfied with the pattern:

- The possible object states are: *StateA*, *StateB* and *StateC*.
- An object can have three different states.
- Objects in *StateA* must have at least one value for property *property1*.
- Objects in *StateB* must have at most one value for property *property2*.
- Objects in *StateC* must have exactly one value for property *property3*.

The following steps must be performed for instantiating the pattern (figure 2 presents a fictitious instantiation of it):

1. Represent all the possible states of the object as instances of the *State* class using the Value Partition pattern [1].
2. Define the set of states, which includes all the states, and its size.
3. Define classes to represent the object in each of the states.
4. Apply state-specific restrictions to these classes.
5. Define the *Object* class as a disjoint union of these classes.

Clearly, in the case when an ontology developer just needs to describe the state of an object, only the first of these steps is needed and an object description should just link to the state of the object. Alternatively, the different states could be modeled as literals but that is not recommended because it hinders extensibility; e.g., modeling them as individuals enables using these states in complex class descriptions as in our case.

One advantage of the pattern is that it allows to explicitly define the restrictions that an object must held in each of its states, instead of relying on documentation or software behavior to discover them. Furthermore, it allows simplifying the use of the ontology by hiding the disjoint union of state-specific object classes to end users, so they just have to deal with the *hasState* property and the *State* instances, while keeping the whole ontology for other purposes (e.g., data validation).

[3] `http://ontologydesignpatterns.org/wiki/Submissions:CollectionEntity`
[4] `http://ontologydesignpatterns.org/wiki/Submissions:Set`

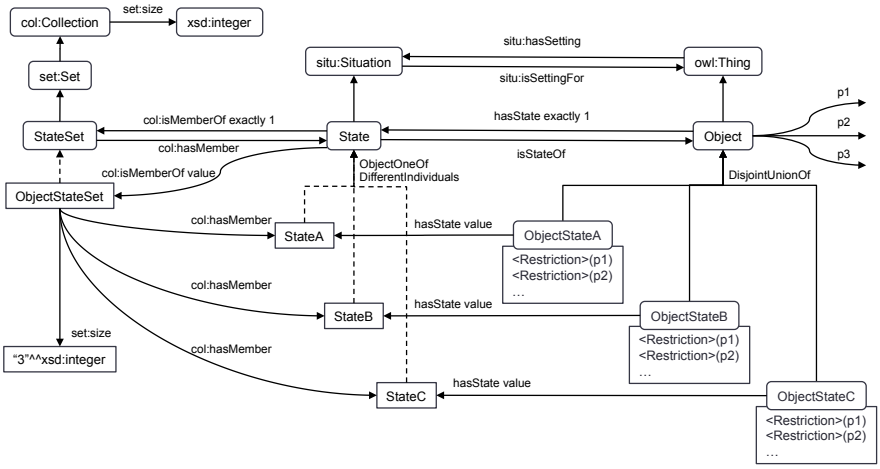

Fig. 2. A fictitious instantiation of the Object with states pattern.

Furthermore, regarding reasoning, the pattern allows checking the consistency of objects according to their states and classifying objects into their corresponding state by means of the *hasState* property (e.g., an object with the *hasState* property with a value of *StateB* can be automatically classified as an instance of *ObjectStateB*).

Even if it is not its primary intended use, the pattern does not disallow to define multiple states for a single object (e.g., a person can have a state of single or married and another state of child, adult, or senior). To instantiate the pattern to that end, the members of each state set should be defined in a subclass of *State* (which would define the value partition) and the classes for the object in each state should be defined in a separate disjoint union of classes.

3 Application of the Pattern

The Object with States ontology design pattern has been applied in the development of the ALM iStack ontology[5], which allows describing entities for software Application Lifecycle Management.

Software defects are one of the main concepts in that ontology. Defects have a concrete lifecycle, shown in figure 3, and in each of its states a defect has a set of required properties. Once a defect is registered into the ALM iStack platform it must have a certain status (usually the status will be *New* upon defect creation) and must satisfy a set of restrictions. In this case, a *New* defect must be assigned to some contributor and an *In-progress* defect must have a priority. These restrictions are propagated to the following states in the lifecycle.

As mentioned in the previous section, the class for registered defects and its subclasses do not need to be explicitly used when interchanging data between the differ-

[5] `http://delicias.dia.fi.upm.es/ontologies/alm-istack.owl`

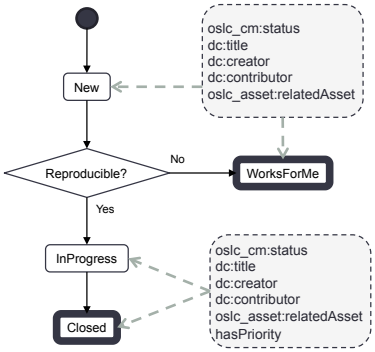

Fig. 3. The lifecycle of a defect.

ent components of the ALM iStack platform, i.e., components can simply talk about defects. These classes have been defined to explicitly specify the restrictions in each defect state and are mainly intended to be used for data validation.

Figure 4 depicts the subset of the ALM iStack ontology that is used to describe the different states of a defect. As can be seen, all those restrictions that are shared by a group of classes have been defined in the higher class in the hierarchy.

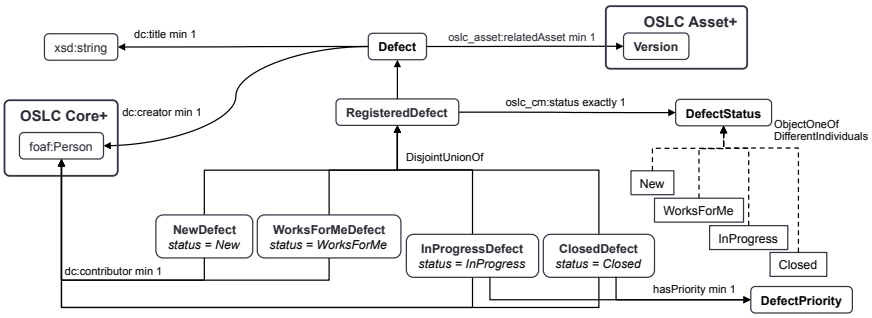

Fig. 4. The Object with states pattern applied in the ALM iStack ontology.

Acknowledgements

The authors are partially supported by the ALM iStack project of the Center for Open Middleware.

References

1. Rector, A.: Representing Specified Values in OWL: “value partitions” and “value sets”. W3C Working Group Note 17 May 2005. http://www.w3.org/TR/swbp-specified-values/