Observation Model

Introduction

This document is the design specification for the implementation of Martin's Fowler observation pattern. There are two main sources of information, which had been used to build this model; Martin Fowler's Analysis Pattern book [Fowler96] and "An Alternative Solution to the Observation Pattern Problem" [Nguyen98] presented in PLOP98.

Fowler goes into quite a bit of details describing models for capturing observations; specifically for medical purposes. Observations are broken down into two main categories. One is for making observations that contain a finite discrete set of possible values such as eye color or gender. These are called Traits. Another type of observation is for those observations that contain sets of ranged values that can have any numeric value and possibly an appropriate unit. These types of observations are called Measurements.

One possible way of building observations could be done by creating a hierarchy of classes describing each kind of observation. This approach properly works when the domain is well known and there is little or no change in the set of observations. Figure 1 shows what the resulting architecture might look like for some basic observations such as height, weight., eye color, hair color, and gender.

Notice here that we associate a set of observations for a Person. Measurements have quantities associated with them for the values. These quantities allow for the possibility of converting types (i.e. 1 inch to 2.45 centimeters). It is easy to describe these classes and it is also easy to extend behavior for these observations by simply programming each class with the needed behavior.

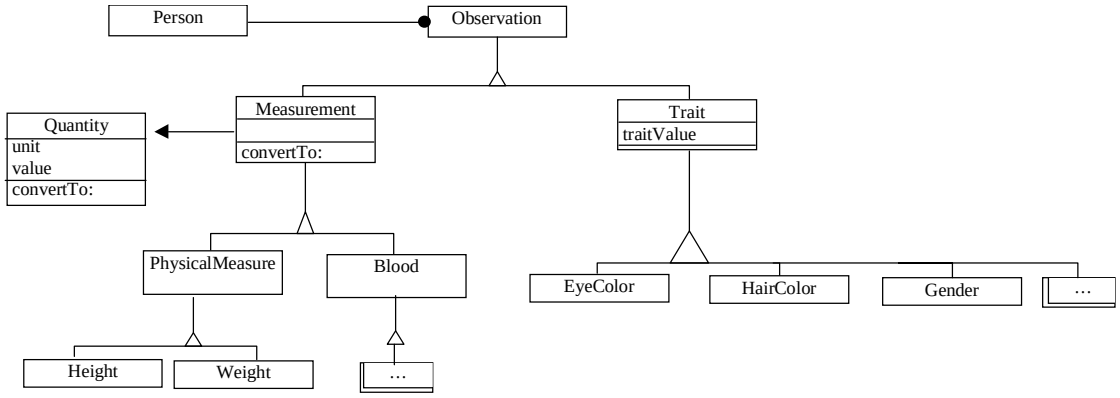

Figure 1 - Static Architecture based on Subclassing

On the other hand, if new specifications for observations are expected, this approach lacks the ability to add or change observations on-the-fly. This is because each time a new kind of observation is needed, or a current observation needs to change, either a new class has to be built, or the existing class has to be changed to reflect the needed changes. Afterwards a new release of the observation application has to be generated and distributed.

Martin Fowler presents some powerful analysis patterns that model the concepts of observations, including those that may change over time. Figure 2 is a class diagram for implementing one of the basic models presented in Fowler's book. Parties have Observations associated with them. There are two kinds of observations: Measurement and Trait. The first one represents those observations which are values in a continuos scale (then there is a unit associated to the value) e.g. 5 feet for height, 180 pounds for weight, etc. The last one (trait) represents discrete observations of a Party, such as blond for hair color, blue for eye color, AB for blood type, etc. The ObservationType describes the subject of the associated observation (e.g. height, weight, blood pressure, etc).

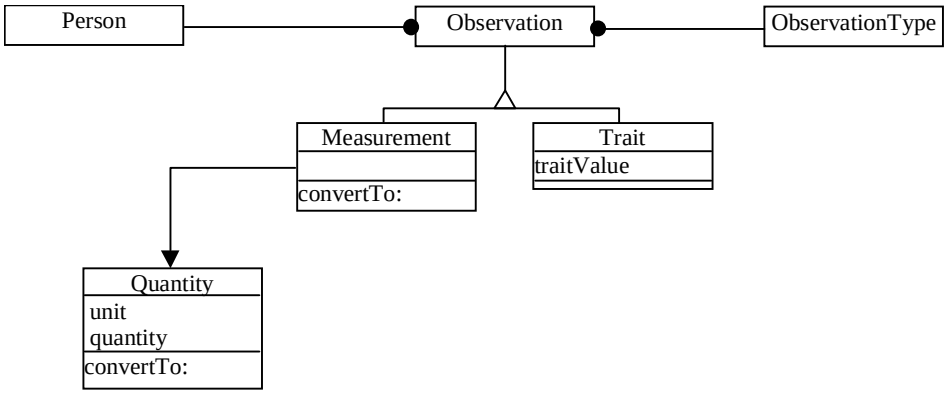

Figure 2 - Class Diagram of the Basic Observation Model

Figure 3 is a simple Instance Diagram with an example of Person called Smith with an observation of height being 5 ft, and an observation of eye color being blue.

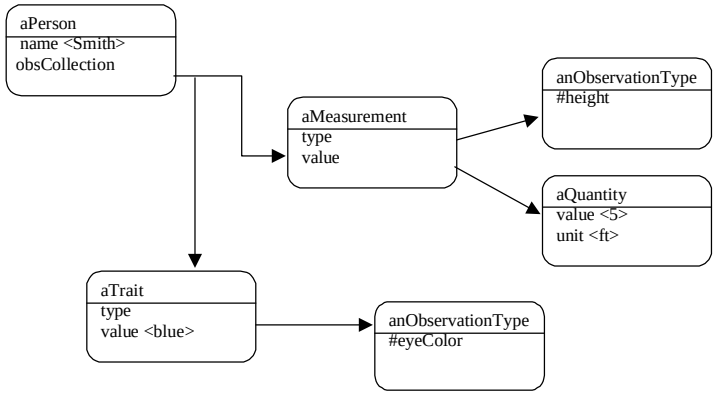

Figure 3 - Instance Diagram of the Basic Observation Model

As it is shown in the figure there is an instance of ObservationType for each different kind of observation. It means that if you want to add a new kind of observation, a new instance of ObservationType has to be created and added to the model (for more details see the *TypeObject* pattern [Johnson98]).

Composite Observations

The above observation model can be used to represent a large class of observations seen in many domains. However it is often the case that the observation might be more complications than the above model can realize. For example, an observation of the "cholesterol" of a patient is composed by two independent measures such as HDL and LDL. Martin Fowler referred to these as compound units for observations. We have extended this concept even further by allowing observations to be composed of other observations. So, a cholesterol observation contains two atomic observations of HDL and LDL. The resulting architecture can be seen in Figure 4.

One of the possible extensions to the original model, [Fowler96] is presented on page 50 as "Associated Observation". In particular this architecture deals with "ways to record the chain of evidence behind a diagnosis.

The architecture proposed in this document handle simple observations (Measurement and Trait) and complex observations (CompositeObservation), this architecture is based on the Composite design patterns [GOF]. Because the former composes the last one, in a very general way both are "associated";

but the semantic of the relationship is different from the idea expressed by the "associated observation" architecture. In fact, any class hierarchy and most of the design patterns have associative relationships. The problem is that general associations can be confusing until you give the associations meaning such as class hierarchy or composite. Martin discusses this in detail in his UML book [Fowler 97].

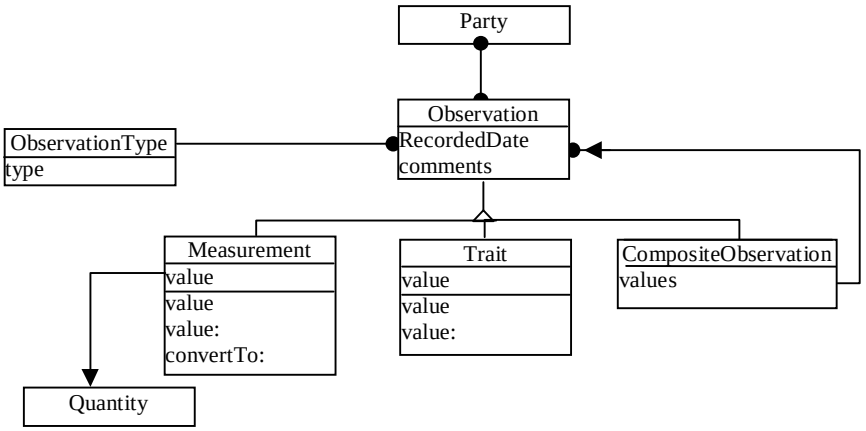

Figure 4 - Class Diagram of Composite Observations

The basic idea of Martin's associated observations is to allow observations to be linked to each other (the patient's thirst indicated the patient's diabetes) and observation concepts to be so linked (thirst indicates diabetes) very similar to diagnosis [Fowler96]. In our model a diagnosis is an observation but it does not have any diagnosis associations (in a general sense) with other observations. In other words we are not analyzing what other observations caused the diagnosis. However, our solution still makes it possible to associate what observations were present during the diagnosis observation so that analysis on the data can be done. This can be done by creating an observation type for the diagnosis that describes the structure of the related observations. It is also possible to associate observations about observations such as we observed that Doctor Jones validated the PKU observation.

An instance of CompositeObservation can be composed of any kind of observation (*Composite Pattern* [GOF 95]). In this way it is possible to define a complex Observation based on basic/atomic ones. For example, an observation about the blood pressure could be described in terms of a composite observation. The "diastolic pressure" and the "systolic pressure" are the components of the "blood pressure" observation (as a composite). Figure 5 shows an instance diagram for this simple example

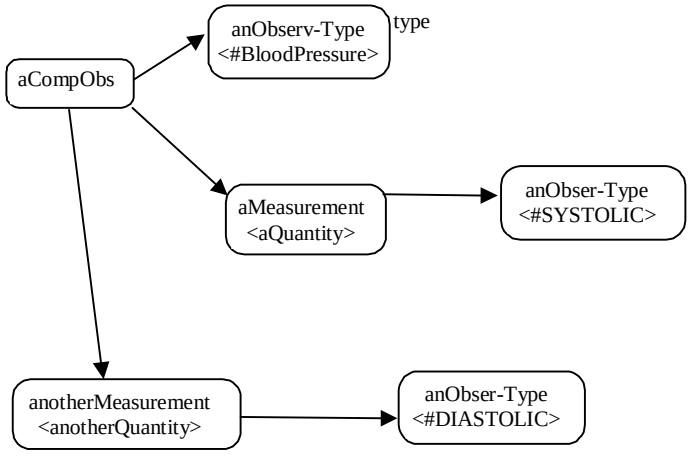

Figure 5 - Example of Blood Pressure

Note that the instance of CompositeObservation (with type #BloodPressure) and the instances of Measurement are associated with different instances of ObservationType (aMeasurement with #SYSTOLIC and anotherMeasurement with #DIASTOLIC). It makes clear the difference between these two different observations. Unfortunately, the relationship between the value of the observation and its type is not yet represented by the architecture. In fact, any value or quantity could be assigned to an instance of Observation whether that assignment is valid or not (in terms of the domains rules). It means that these rules have to be recorded in the head of the programmer (in order to validate inputs from the GUI) or in the head the final user. One solution for this problem is to add some part of the responsibility of validation to the ObservationType; afterward the model could describe by itself the validation rules (extracted from the domain). Nguyen and Dillon [Nguyen98] presented a similar architecture in "An Alternative Solution to the Observation Pattern Problem."

Validations

The proposed architecture handles different types of observations, "measurements," "traits," and "composed observations." The subject of each observation is defined by one particular instance of the class ObservationType. As previously mentioned, it is possible to extend each type for describing the set of possible valid values associated with them. This can be done by associating each instance of ObservationType with an appropriate *Strategy* [GOF] for validating the types.

Each ObservationType is associated with an object that is responsible for determining whether a value is valid or not. A class hierarchy of validators could have been developed and dynamically plugged in at run-time. However, analysis revealed that there are two basic kinds of values that could be the quantifier of an observation, first, a quantity (a single values of a continuous scale) e.g. height, weight, etc.; second, categories (single values, constants) e.g. eye color, hair color, etc. This allows for the ObservationTypes to in a sense be extended with types of validators. So in a sense, an observation uses a *TypeObject* to describe its type of observation, which in turn uses the *TypeObject* pattern to describe its validator. Descriptive data (metadata) can then be used to associate and instantiate the appropriate validators with the appropriate types of observations. This double use of the *TypeObject* pattern (which we call the *TypeSquare* pattern) is commonly seen in dynamic meta-architectures. Foote and Yoder [Foote98] describe this type of dynamic architecture in more detail.

The resulting architecture for Validators is shown in Figure 6.

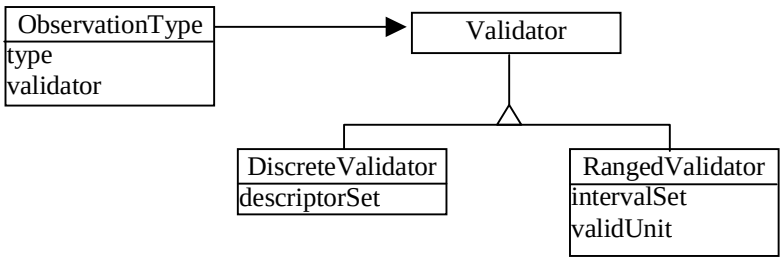

Figure 6 - Architecture for Observation Validation

Each subclass of Validator records a different class of valid elements. In the case of the DiscreteValidator, it knows a collection of valid constants e.g. blue, green, brown for valid eye color. In the case of the RangedValidator, it contains a collection of intervals where a value is valid and unit in which the quantity of the measure is "legally" expressed.

Notice that instances of Trait are always associated with an instance of DiscreteValidator and instances of Measurement are always associated with instance of RangedValidator. Thus, DiscreteValidator and RangedValidator are just describing the difference between the values they are

expecting to store in the #observationValue variable. Therefore, the Measurement and Trait classes can go away.

Figure 7 shows the resulting class diagram for the implementation of observations with validators. Parties are an abstract concept described in Fowler's book for dealing with dynamic organizations and people. Describe more here.

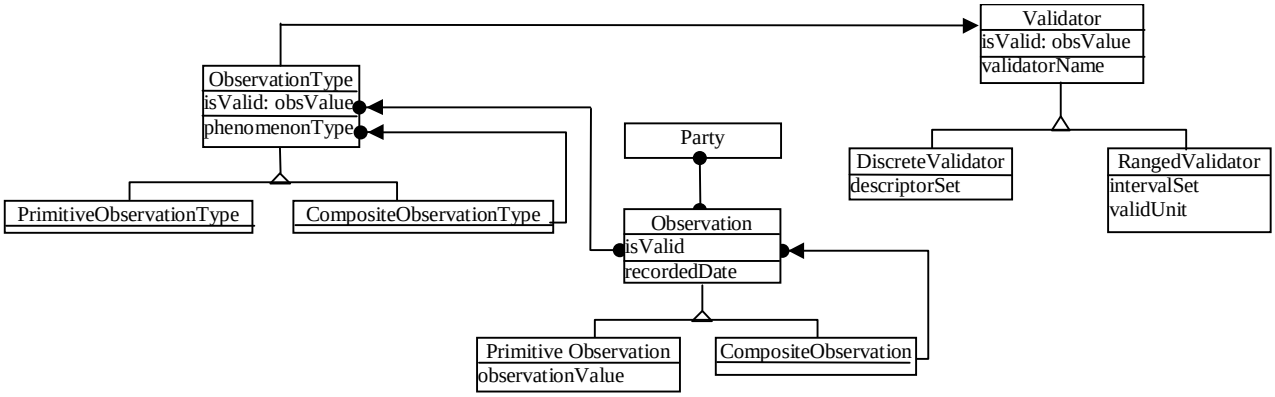

Figure 7 - Complete Class Diagram

Examples

A) *"A child gives a blood sample to determine the level of lead in his body. The result of that test would be an observation. An actual result value could be 10 micrograms/deciliter. The child could have 0 to many tests in their lifetime"*

WE HAVE TO REBUILD THE DIAGRAM

Figure 8 - Object Diagram for Example 1

The object diagram for the example is shown in Figure 8. In this case the Observation is a composed observation and the type is #TestResult. It knows a collection of one single observation, aMeasurement, which already has its own ObservationType.

B) *"I.- A person is given a skin test for Tuberculosis. They show a positive reaction.*
II.- They give a saliva sample for testing. The acid bath test show a positive reaction.
III.- The microbacteria growth test indicates a living organism.
IV.- The organism is identified as Microbacterium Tuberculosis Hominis.
V.- After several antibodies are applied to a sample, it is determined that the antibiotic Penicillin will effectively kill this strain of bacterium."

From the example above it is easy to recognize I, II as observations (probably they are Test-Results). III and IV could be results of the same test, so it could be observations within a new Test-Result. *Is V an observation? Or is it a new object a medical prescription?*.

The example is Medical Condition, which is an observation type associated with a CompositeObservation. Figure 9 presents some abstraction of the resulting architecture.

WE HAVE TO REBUILD THE DIAGRAM

Figure 9 - A more abstract representation of Composite Observation

Each instance representing composite (#TestResult) observation (Figure 9) is aCompositeObservation with #TestResult as its type and a collection of measurements or traits (as was shown in Figure 8)

References

[Fowler96] M. Fowler. *Analysis Patterns, Reusable Object Models*. Addisson Wesley. 1996

[GOF] Eric Gamma, Richard Helm, Ralph Johnson, John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software,* Addison-Wesley, Reading, MA, 1995.

[Johnson98] R. Johnson, B. Wolf. "Type Object". *Pattern Languages of Program Design 3*. Addisson Wesley, 1998

[Nguyen98] N. Nguyen, T. Dillon. "An Alternative Solution to the Observation Pattern Problem". Proceedings of Plop98. Technical Report #wucs-98-25, Dept. of Computer Science, Washington University Department of Computer Science, October 1998. URL: http://jerry.cs.uiuc.edu/~plop/plop98.

[Foote98] B. Foote, J. Yoder. "Metadata and Active Object Models". Proceedings of Plop98. Technical Report #wucs-98-25, Dept. of Computer Science, Washington University Department of Computer Science, October 1998. URL: http://jerry.cs.uiuc.edu/~plop/plop98.