Business Model Application of UML Stereotypes

Daniel Brandon, Jr.
Christian Brothers University, USA

OVERVIEW

The UML (Unified Modeling Language) has become a standard in design of object oriented computer systems (Schach, 2004). UML provides for the use of stereotypes to extend the utility of its base capabilities. In the design and construction of business systems, the use of stereotypes is particularly useful stereotypes, and this article defines and illustrates these.

UML STEREOTYPES

"Stereotypes are the core extension mechanism of UML (Lee & Tepfenhart, 2002). If you find that you need a modeling construct that isn't in the UML but it is similar to something that is, you treat your construct as a stereotype" (Fowler & Kendall, 2000). The stereotype is a semantic added to an existing model element, and diagrammatically it consists of the stereotype name inside of guillemots (a.k.a. chevrons) within the selected model element (Schach, 2005). The guillemot looks like a double angle bracket (<< … >>), but it is a single character in extended font libraries (Brown, 2002). Each UML model can have zero or many stereotypes; and each stereotype can define a set of tagged values, constraints, and possibly a special icon or color (Arlow & Neustadt, 2005). The UML defines about 40 of these stereotypes, such as "<<becomes>>", "<<include>>", and "<<signal>" (Scott, 2001). However, these 40 standard stereotypes are not particularly useful in business models and do not add the meaning necessary for automatic code generation in a UML CASE tool.

One common general use of the stereotype is for a metaclass. A metaclass is a class whose instances are classes, and these are typically used in systems in which one needs to declare classes at run time (Eriksson & Penker, 1998). A similar general use is for powertypes. A powertype is an object type (class) whose instances are subtypes of another object type. Figure 1 shows an example of the use of stereotypes for powertypes (Martin & Odell, 1998). Another type of usage is in the "binding" of parameterized (template) classes historically in C++ and now in the latest version of Java (Oestereich, 1999).

USER-DEFINED STEREOTYPES FOR BUSINESS SYSTEMS

In the design of business systems we have found some stereotypes that were occasionally useful, and two stereotypes that are extremely useful. When defining stereotypes it is useful to describe (Eriksson & Penker, 1998):

1. On which (UML) element the user-defined stereotype should be based
2. The new semantics the stereotype adds or refines
3. One or more examples of how to implement the user-defined stereotype

Figure 1.

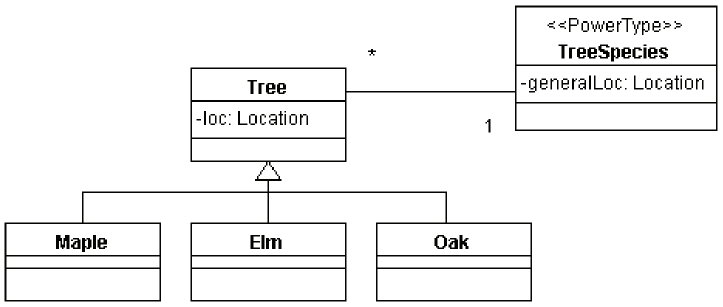


Figure 2.

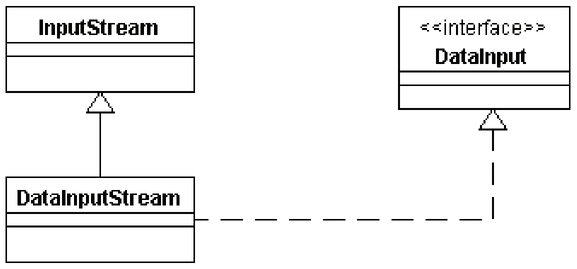
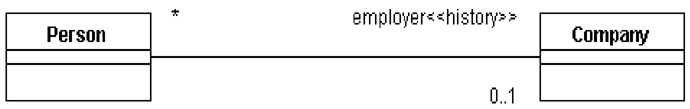


Figure 3.



An issue is where such definition takes place both within UML and in the UML software tool. If a modeling tool does not provide such built-in support (and most do not), many modelers put a note in the model or a reference to external documentation (Arlow & Neustadt, 2005).

One common use of stereotypes in business systems is for interfaces as found in Java or CORBA; this is shown in Figure 2. An interface typically has public functionality but not data (unless holding data for global constants). The class model element has been modified with the "<<interface>>" notation (Oestereich, 1999). This is commonly used for UML CASE products that do not have separate interface symbols or where these symbols do not allow data (i.e., global constants).

Another common stereotype usage in business systems is to clarify or extend a relationship. Figure 3 shows a stereotype called "history," which implies a "many" cardinality for history purposes, that is, each Person has zero or one current employers but may have many employers in terms of the employee's history. It may imply some common functionality upon code generation such as (Fowler & Kendall, 2000):

Company Employee::getCompany(Date);

Still another common stereotype usage in business systems is to define ancillary classes that are not part of the core business model, but are needed for a particular implementation. Boundary, controller, and entity stereotypes were predefined in UML (Oestereich, 1999) with special symbols in some products; however, the definition and use of these varies with author. Schach (2005) divides implementation classes into Entity classes, Boundary classes, and Control classes. Entity classes are part of the core business model and represent entities like people, organizations, transactions, places, and things. Boundary classes are used for communication between the software product and the actors such as I/O operations; and Control classes are used for algorithms and business rules. Each of these types of classes is designated with the stereotype notation.

CODE WRITING AND GENERATION

Most modern UML CASE (computer-aided software engineering) products can generate "skeleton" classes from the UML class diagrams and possibly other diagrams. For business systems design, we need to write the code for our classes (usually implemented in Java or C++) based on both the structural model (UML class diagram) and the dynamic model (UML activity diagram). This process is shown in Figure 4. It is very important that consistency between the two diagrams is achieved.

Many such CASE products allow the user to write his or her own "class generation scripts" in some proprietary scripting language or in a general scripting language (i.e., Python). With user-defined stereotypes, the user can modify the class generation script code to use his or her stereotypes as needed.