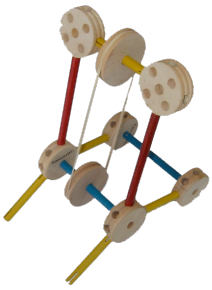

Checkbook Evaluation Notes

March 7, 2014

These notes explain UML model evaluation and usefulness characteristics consider toward grading UML models. *L. J. Waguespack, Ph.D.*

Pat's Checkbook Usefulness

Pat's checkbook problem is a relatively simple data intensive business scenario. The real challenge for this object-modeling exercise is to recognize all those elements that would be present if you were designing a relational database representation of Pat's needs. There are classes that represent "persistent" information needs like *transactions* and *tax related income/expense categories* that would have to be modeled almost identically in the relational paradigm as they need to be in the OO paradigm.

The thing that makes modeling in the OO paradigm different is that unlike entities in the relational paradigm, objects from classes in the OO paradigm are supposed to have individual behavior responsibilities to go along with the information that they possess. That requires that you focus in on the "business rules" (rather than "common practice") that explain the interrelationships between business objects that define what "can happen" in Pat's checkbook "world" Some of the challenge is identifying Pat's *perspective* on the information and activities that occur. For example if Pat doesn't control the Bank then the model can depict very little detail about the Bank except those services that it provides to satisfy Pat's needs expressed in the scenario.

So a big part of developing a USEFUL UML business model is getting into an OO mindset and trying to identify "which objects know what" and "which objects are responsible for doing what" to satisfy the business rules in the scenario provided. To be honest getting really good at thinking in the OO paradigm takes a lot of practice by both reading many good UML models as well as debating with other modelers about the best way to capture scenario content in classes and relationships. I don't really expect everyone at this novice stage of learning OO modeling to develop excellent UML models, but they should be useful!

So What Makes a Useful Model?

The first dimension of usefulness is syntax. That means using the UML syntax to express inheritance, association, message and parameter passing, sequencing, and synchronization clearly. It means in class diagrams using the correct diamond on associations where the cardinality is consistent with the placement of the diamond on the top or bottom of the connecting line. (Notice "top" and "bottom" of connecting line since all aggregations and compositions should be bottom of "whole" class to top of "part/member" class in the class diagram!) It means in sequence diagrams being sure the label on message lines is the service name invoked in the *receiving* object's class and representing conditions and iteration on messages correctly. It means naming model elements correctly: single nouns for class names, present tense imperative verbs for service names, and singular/simple nouns for attributes.

The second dimension of usefulness is semantics. Names need to relate seamlessly to the problem scenario and those names should *ALMOST* not need a detailed prose description of its purpose and relevance in the model. Names of classes should identify individual objects and each object would have only a single value of each of its attributes. Any structurally complex model element will probably be rendered as some form of collection: aggregation, composition, or simple association. Any class name that implies a group should probably define the owner of a collection that includes all the subordinate elements that compose or make up that group concept.

The third dimension of usefulness in UML models is consistency. The concepts depicted in the use case documentation should be easily perceivable in the class diagram. All the elements of a sequence diagram should map seamlessly back to the class diagram: objects of classes, services in receiver objects, messages between objects that have some navigation connection through associations, etc. The prose descriptions should focus only on what the serving object remembers and knows what to do so service descriptions should encapsulate each object's behavior responsibilities only referring to its own attributes, parameters in received messages, and objects to which it directs messages to support its service.

The fourth dimension of usefulness is completeness. Does the model retell a cogent "story" of all the responsibilities and behavior that is mentioned in the problem scenario. This should be pretty easy to check - scenario behaviors require sequence diagrams to describe the accomplishing actors, sender and receiver objects with relevant class diagram resources to utilize and manipulate.

So How Do I Evaluate Your Model?

I look for model elements that reflect the four dimensions of usefulness outlined above. I try to locate model elements that contribute to these dimensions building a cogent modeling representation of the scenario - usually reflected by few (if any) annotations on the model. I also try to identify model elements that seem to defeat achieving cogency in these dimensions by noting issues of syntax, semantics, consistency, or completeness by marking specific items (usually circling/numbering them on the model and expanding on the perceived deficiency either directly on the model or on an accompanying grading form). I make no attempt to identify EVERY questionable element, but I try to identify indicative elements that should lead you to find and correct similar deficiencies across the model. It takes as much as 30 minutes per model for this evaluation. Because modeling and evaluating models is significantly subjective, I require at least one preliminary submission to suggest improvements to help you to refine or improve the *usefulness* of your model before final grading.

Here are some general comments that recurred on the feedback of the Phase II of the checkbook for you to consider in your final revisions:

1) Your first task is to identify all the core concepts in the problem either by defining classes that represent the essence of the concept or by defining relationships between objects of classes demonstrating shared responsibility or cooperation.

2) The best way to test your model is to read the syntax of your symbols to yourself "OUT LOUD" and listen to what the symbols "say!" If the reading of the symbols syntax (class, attributes, services, relationships) matches up with the problem scenario and the actions needed to satisfy the requirements then you're on the right track. Usually reading them "out loud" to yourself will help you catch problems with cardinality ("How many of these belong to how many of these?) as well as "ownership" ("If one of these goes away do these related to them go away also?") and finally inheritance (gen/spec) ("Object of this child class IS A object of the parent class?")

3) Pat's categories may seem arbitrary (food, rent, entertainment, . . .) but when we introduce the probability that the checkbook assistant will support tax preparation, those categories take on a more focused purpose. If Pat's bright enough to align the categories with aspects of the tax form's input requirements, then "keeping track of expenses" can easily be expanded to track expenses that relate to taxes as well as income that is taxable.

4) The whole purpose of defining classes is to identify a framework, a template, of structure and behavior that allows many individual instances of a concept to be described only once in the class (e.g. many specific students in the student class, many specific courses in the course class, etc.). The key is to find a template that lets you plug in any specific attribute values to depict a specific instance of the class. So, the skill is "generalization" (how are all this individuals alike in structure [data attribute variables, and services]). *(Did you notice the reference to the OO Ontology! That's where all these model concepts come from! Maybe you should take 15 minutes and review that again!)*

5) You have several model examples from the class to use as frameworks for your model and documentation. The most compact is the last Zoo handout thats posted on the web. It eliminates redundant terms and highlights the basic elements that your submission must contain: Class diagram with descriptions for each element (class, attribute, service, relationship), Use Case's describing the interaction of Pat with the checkbook assistant, and Sequence Diagrams to depict the sequence of messages exchanged between objects in your class diagram demonstrating what ACTIONS occur in what order to complete a task that satisfies Pat's Needs. There are also the UML Guidelines and the Association and Class example handouts not to mention the Fowler text for inspiration!!! Model On!

Courtesy of one of your classmates!