Summary of UML Notation in Rational Rose 2002

By Marc-André Laverdière

With extracts from: http://www.sparxsystems.com.au/UML_Tutorial.htm and http://amstel.wins.uva.nl/~martinbe/umldiag/uml1.htm (Recommended Reference)

Covered Today

- Basics
  - Use Case Diagrams
  - Sequence Diagrams
  - System Sequence Diagrams
  - Class Diagrams
  - Activity Diagrams
  - Statechart Diagram
- Architecture
  - Packages
  - Component View
  - Deployment View

Use Case Diagrams

- Use Case Diagrams represent the functional requirements of the system, how they interconnection, and which actors uses them
- Actors here are roles and representatives, not specific persons
- Use Case Diagrams can be used to model pretty much anything.
- We will focus here on the most important elements of the notation
- There are many stereotypes, but most are of very little use, so we won’t detail them.

Use Case Diagrams

- The Use Case Diagram shows actors and related use cases, as well as relations between use cases

Relationships:

•extends: One use case extends the relationship of another. It is a branching off the original flow. See it as a conditional statement.

•uses/includes: The use case pointed to is used by the pointing use case every time (in this case, “Register with BookShop” includes “Login”)

Sequence Diagrams

- Sequence diagrams show the sequence of operation and relationships between objects.
- In the case of sequence diagrams, the operations are between instances of classes.
- Rational won’t let you explicitly show looping and conditional execution. Still, use the repetition notation and conditional notation with notes.
- Sequence diagrams should show the flow of operation. This will make your architectural decision very explicit.

Sequence Diagrams

This kind of arrow is a message

This kind of arrow is a return value

This is a self-call

This is an informal sequence diagram.

Sequence Diagrams

Example of ad hoc conditional and looping notation.

System Sequence Diagrams

- System Sequence Diagrams are used to illustrate scenarios, and represent the system as a black box.

Formal
Doesn't both with UI details

Informal

Class Diagrams

- A class diagram will show the classes that are part of a system or package.
- The internal data members and the methods are displayed.
- The Domain Model is a specific case of a class diagram. There are no methods, and data members are only of the most basic types (strings, numbers).
- Also the association arrows might not be directed, and aggregation is normally not shown.
- The classes shown in the domain model are those conceptual classes from the domain of study.

- The classes are shown here as an illustration
- A final design class diagram will normally have all methods and variables defined.
- Rational won't let you have undirected arrows. This diagram was drawn in Visio.
- You can open the arrow's property box in order to specify cardinality, aggregation, etc.
- The final note: the tool won't make you draw good diagrams, you need to know UML and force the tool to cooperate as much as possible.

An undirected association
Normal for domain models
However, it should have a cardinality

Inheritance

An aggregation
Normally not shown in domain models
It should have a name

Activity Diagrams

- Represents the sequence of action
- Used mostly for business modeling
- Of limited use in general
- Transitions are automatically triggered by completion of activity
- States are activities
- Special kind of statechart diagram

Statechart Diagrams

- Represents the state a system/subsystem is in, and events related to these states.
- Common use is to show sequence of external system events that are recognized and handled in the context of a use case
- Used in protocol design in the architecture phase.

Packages

- Groups classes together
- Some classes are public, which means that they can be accessed outside the package
- Private classes can only be accessed within the package.
- Rose Won’t let you draw the packages with the level of detail you need.
- In this case, the packages supported in Rose are mostly useful to structure layering and to show dependencies.
- In architecture, we bother about components first. Then these components become packages of classes.
- Structuring into components is necessary to ensure quality of software, especially regarding cohesion, coupling and reusability.

Packages

This illustration shows architectural structuring with packages and embedded packages. Remember that these packages have classes that define their interfaces.

Excerpt from http://www.cs.umd.edu/~kang/435/design/pd.html

Packages

This simple example illustrates well.
The Mammal Class is within the Mammals package.
As such, any referencing in class/sequence, etc. diagrams will therefore indicate the packages needed by the other classes.

It is strongly suggested to organize semantically similar classes in the same package.
This also holds true for related classes with specific performance constraints.

Component View Diagrams

- The actual architectural support is offered by Rational RealTime, but it doesn't support the latest UML notation, which is very restrictive.
- You will show your layering and packaging here.

Component View Diagrams

Each the package have the <<layer>> stereotype here. We could also have <<module>> and <<subsystem>>.
Keep in mind that layers can have subsystems and/ or modules, and subsystems have modules.

The {all} here shows a restraint

The dotted lines show dependency

Here the components are just images. For real component definition, use Rational Rose RealTime.

Deployment View Diagrams

- Rational makes it very hard to represent the deployment/execution view of a system.
- These diagrams show which processors run which processes, etc.

Visio lets you put the classes and packages within the processing units.

However, Rose won't let you. Instead, it'll show at the bottom of each CPU the threads and processes that are run.

Deployment View Diagram

- You use class and package stereotypes such as <<process>>, <<thread>>, <<dll>>, etc. in order to represent which kind of component.
- You can easily show distributed and client/server architectures this way.

This is what rational shows in its view, which doesn’t help that much.

Deployment View Diagram

- This is what the Deployment Diagram should look like (if Rational did draw real UML):