Notations and Meta-Models

Notation:
• a graphical representation of a model, e.g. classes, associations, multiplicity
• appeals to intuition rather than formal rigor

Meta-model:
• a diagram which explains the notation
• UML diagrams explaining UML itself
Meta-model Example

- Feature
- Structural Feature
- Behavioral Feature

0..1 the meta-model describes the syntax of the UML diagrams and thus defines what a well-formed model is.

Parameter [ordered] *

Analysis and Design

- Final goal: “cutting code”
- UML diagrams etc are “just pretty pictures”
- HOWEVER:
  - Communication is of utmost importance, both inside the development team and outside with the customer/user
  - Build “the right” system – need for a “domain expert”
  - Need to concentrate and highlight the important details
  - Make extensive use of “use cases”
  - Build a road-map for the construction
  - Exploit o-o techniques
The UML Development Process

inception → elaboration → construction → transition

In the elaboration phase the requirements are refined and high-level analysis and design is carried out.

The construction phase consist of many iterations; it is the key phase.

The transition phase is mainly for Beta-testing, acceptance, performance optimization and user training.

Elaboration

Problem definition
1. What is it we are going to build?
2. How are we going to build it?

What are the risks
1. Requirement risks – did we understand what the customer wants?
2. Technological risks – can it be done with current technology?
3. Skill risks – can we do it with our expertise?
4. Political risks – always around, be realistic, always look at the $$$ side.

Plan the construction phase - iterations and milestones.
Views and Diagrams

- A model is an abstraction of a system or a context
- An architectural view is an abstraction of a model,
  - taken from a specific perspective / view
  - enabling the extraction of architecturally essential
    elements
- Starting point is always the “user view” as defined
  by the use cases
- The different model views:
  - structural view
  - behavioral view
  - environmental view
  - implementation view

Views and Diagrams - 2

- Structural view:
  - class diagrams (generic templates, “types”)
  - object diagrams – instances, “variables”

- Behavioral view:
  - sequence diagrams
  - collaboration diagrams
  - state chart diagrams
  - activity diagrams

- Implementation view:
  - component diagrams – modules

- Environment view:
  - deployment diagrams
Use Cases

- Scenarios, typical transactions and interactions have long been used for understanding requirements.
- Use cases do this in a more systematic manner.
- Introduced by Jacobson initially for telecommunications systems.
- Based on “scenarios”:
  - a scenario is a sequence of steps describing the interaction between a user and a system;
  - a user is a “role” i.e. an external entity using the system in some way, for example by sending signals, entering data, etc.; it could be a human user or a device, external system;
  - example: “the sensor sends a report consisting of 16 bytes every 40 msec”

Caveat: when writing the use cases do NOT concentrate on the functions but on the interactions!
Use Cases - 2

- A scenario is one transaction that can happen, but it could fail or lead to an alternate transaction.

- A use case is:
  - A set of scenarios tied together by a common user goal.
  - Frequently one finds a common “all-goes-well” case and many alternatives.
  - These alternatives might cover special cases (things go well) and error situations (things go bad).

Example use case: “Serving Dinner”

- make use of “structured writing”

Example: Serving Dinner

1. The use case begins when the actor Guest enters the restaurant.
2. The actor Guest has the possibility of leaving his/her (!) coat in the cloakroom after which he is shown to a table and given a menu.
3. When the actor Guest has had sufficient time to make up his mind, he is asked to state his order. Alternatively, Guest can attract the waiter's attention so that the order can be placed.
4. When the guest has ordered, the kitchen is informed what food and beverages contain.
5. In the kitchen certain basic ingredients such as sauces, rice, chilies, tacos, already have been prepared. Cooking therefore involves collecting together the basic ingredients, adding spices and so on and sorting out what needs to be done just before the dish is served. Also, the required beverages are fetched from its proper storage.
6. When the dish is ready, it is served to the actor Guest. When it has been eaten, the actor is expected to attract the waiter's attention.
7. When the bill is paid, Guest can fetch his coat from the cloakroom and leave the restaurant. The use case is then completed.
Use Cases - 3

A transaction is defined as an "atomic set of activities that are performed fully or not at all; it is invoked by a stimulus from an actor to the system or by a point in time being reached. A transaction consists of actions, decisions, and transmission of stimuli to the invoking actor or some other actor(s)."

An actor is a "role that someone or something in the environment can play in relation to a business"; alternatively, "an actor represents anything that needs to exchange information with the system. An "individual actor" sometimes referred to as user) is defined as an instance of class Actor. Further, the same person or other item can assume more than one role.

Use Cases – cont.

Actors / roles carry out business use cases
A business use case is typically broken down into a set of systems use cases
Start with a set of actors and then work out the use cases for each actor.

Many variations are possible, e.g.:
- actors – every external system or user or just the initiator
- show the one that “gets value”, the primary actor
- do NOT include everything – instead make up separate use cases
- add detail during the iterations

Fig 3-1,-2 Example Use Case and Diagram
Example – e-commerce

**Buy a Product**
1. Customer browses through catalog and selects items to buy
2. Customer goes to check out
3. Customer fills in shipping information (address; next-day or 3-day delivery)
4. System presents full pricing information, including shipping
5. Customer fills in credit card information
6. System authorizes purchase
7. System confirms sale immediately
8. System sends confirming email to customer

**Alternative: Authorization Failure**
At step 6, system fails to authorize credit purchase
Allow customer to re-enter credit card information and re-try

**Alternative: Regular Customer**
3a. System displays current shipping information, pricing information, and last four digits of credit card information
3b. Customer may accept or override these defaults
Return to primary scenario at step 6

---

**Actors**

- Actor is a "role" with respect to the system, e.g. "Trader", "Waiter", "Guest", "Subsystem", "NetworkInterface", . . .
- Possible actors: humans, external systems, system components, devices, . . .
- Actors carry out use cases, (transactions, not functions)
  - Fist step: define the list of actors,
  - Second step: define the use cases for each actor.
- The goal is a description of the use cases, not the actors.
- External events are an excellent source for identifying use cases. A given event:
  - may cause a system reaction,
  - may cause a reaction from the user.
- A business use case is a response to the user or an event.
- A system use case is an interaction with the software.
- *** remember: actors can also be receivers of output!***
UC - Relationships

- The **include** relationship factors common activities in a separate sub-use case which is referred to from the more general use cases.
- Similarities between use cases can be handled by a use case **generalization** which is another way to capture alternative scenarios.
- When more rules have to be observed in handling alternatives, a use case may be **extended**, which adds behavior to the base use case, but only at declared extension points.
UC – Relationships – cont.

Both generalization and extension allow splitting up a use case. This frequently happens during elaboration when a use case gets too complicated to be handled within one iteration. Handle the normal case first and the variations later:

- Use include when repetition occurs in two or more separate use cases [and you want to avoid repetition]
- Use generalization when describing a variation of normal behavior [and you wish to describe it casually]
- Use extend when describing a variation of normal behavior and the more controlled form declaring extension points in the base use case is preferred.

When to use Use Cases

- “I can’t imagine a situation in which I would not use use cases” – Fowler
  - They are an essential tool in requirements capture and in planning and controlling an iterative project
  - Capturing use cases is one of the primary tasks of the elaboration phase.
  - Try to do use cases and conceptual modeling at the same time
  - Use Cases represent an external view of the system
  - Do not expect, hence, to see correlations between use cases and the classes inside the system.
Fred Brooks – The Mythical Man Month

““The hardest single part of building a software system is deciding precisely what to build” [p.199]

No other part of the conceptual work is so difficult as:
- Establishing the detailed technical requirements
- Including all interfaces to
  - people
  - machines / systems
  - other software systems / components

No other part so cripples the resulting system if done wrong and is more difficult to rectify later. (security!)

Therefore the most important function that software builders do for their clients is the
- Iterative extraction
- Iterative refinement
of the product requirements

For the truth is the clients usually do not know what questions must be answered.

Building a Software System

Elaborate the user/customer requirements - build the base use cases

Construct the system in a number of iterations and refinements, building a series of models
- requirements model
- analysis model
- design model
- implementation model
- test model

Incremental development - the first approach is just a running skeleton executing a set of dummy methods; the first prototype performs mainline tasks, leaving out handling of exceptions, alternatives, incorrect inputs, clean-up and clean abort.

Each model captures some part of the system and are the output of the individual activities
Caveat: when writing the use cases do NOT concentrate on the functions but on the interactions!
Example: Serving Dinner

1. The use case begins when the actor Guest enters the restaurant.
2. The actor Guest has the possibility of leaving his/her (!) coat in the cloakroom after which he is shown to a table and given a menu.
3. When the actor Guest has had sufficient time to make up his mind, he is asked to state his order. Alternatively, Guest can attract the waiter's attention so that the order can be placed.
4. When the guest has ordered, the kitchen is informed what food and beverages contain.
5. In the kitchen certain basic ingredients such as sauces, rice, chilies, tacos, already have been prepared. Cooking therefore involves collecting together the basic ingredients, adding spices and so on and sorting out what needs to be done just before the dish is served. Also, the required beverages are fetched from its proper storage.
6. When the dish is ready, it is served to the actor Guest. When it has been eaten, the actor is expected to attract the waiter's attention.
7. When the bill is paid, Guest can fetch his coat from the cloakroom and leave the restaurant. The use case is then completed.