Lecture 4: Descriptions

Kenneth M. Anderson
Object-Oriented Analysis and Design
CSCI 6448 - Spring Semester, 2001

Goals for this Lecture

- Discuss four types of descriptions
- Have another class activity session

Descriptions

- Descriptions are the central activity of software engineering
- They are manifestations of thought
  - if you understand how descriptions work, and how they can differ from one another, you can improve how you think about problems
    - "thinking about descriptions is thinking about thinking"
  - this, then, is why we often have so many different notations for expressing problems and solutions
    - each represents a different way of thinking

(At least) 4 types of descriptions

- Designations
  - phenomena of interest
- Definitions
  - a formal statement of a term that can be used by other descriptions
- Refutable Descriptions
  - describes a domain, saying something that could, in principle, be refuted or disproved
- Rough Sketches
  - a tentative description of something that is still being explored or invented; uses undefined terms

Designations

- A designation singles out a kind of phenomenon as being of interest, and gives it a name
- They provide us with the tool we need to identify the important elements of our problem domains
- They consist of a recognition rule on the left. On the right, after a designation symbol "≈", is the designated term.
- Designations are, thus, informal; they rely on natural language to describe the recognition rule, which requires a human to interpret; however they can be used to build formal definitions, as we shall see

Example Designations

- x is a human being (Homo sapiens) ≈ Human(x)
- x is male ≈ Male(x)
- x is female ≈ Female(x)
- x is the biological mother of y ≈ Mother(x, y)
- x is the biological father of y ≈ Father(x, y)

More on Designations

- Designations must be…
  - phenomena that are clearly and unambiguously recognizable in the domain;
  - with good recognition rules
- …within reason, since most domains are informal, there will always be exceptions;
  - but do "well enough" for the purposes of the system that you are building
  - If you cannot write a good recognition rule you have probably chosen an unsuitable phenomenon
    - You need to choose your designations carefully because they form the foundation of all your descriptions!

Definitions

- Every term you use in a description should be defined
  - One way to define a term is to give a designation
- Once you have designations, you can build on them
  - $\forall$ x,y • ((Human(x) $\wedge$ (Mother(x,y)) $\rightarrow$ (Female(x) $\wedge$ Human(y)))

Definitions are…

- …relationships among designations
- Referring back to our previous example, how would you introduce the concept of brother?
  - as a designation?
  - x is the genetic brother of y ≈ Brother(x,y)
- This is not quite right…why introduce another designation, when we can formally define the term with a definition?
  - definition: designated term ✦ assertion

Defining Brother

- Define the term Brother using the existing designations and predicate logic, this formalizes the term
  - Brother(x,y) ✦ Male(x) ∧ ∃ f • (Father(f,x) ∧ Father(f,y)) ∧ ∃ m • (Mother(m,x) ∧ Mother(m,y))
- Careful use of definitions keeps the number of designations small; this, in turn, makes it easier to understand your descriptions
  - plus, in this particular instance, a brother is not really a separately observable phenomena from the designations you have so far, its simply a certain kind of relationship between them

Building on Definitions

- Once you have some definitions, you can use them to create more definitions
  - helping you to grow the number of formal descriptions you have to apply to your software development project
  - Uncle(x,y) ✦ ∃ p • ((Father(p,y) ∨ Mother(p,y)) ∧ Brother(x,p))

Definitions are not Assertions

- Definitions cannot be true or false
  - only well-formed or badly formed
  - only useful or not useful
- Think of it as a substitution
  - if I have an expression
    - ((Father(Ken,Kevin) ∨ Mother(Ken,Kevin)) ∧ Brother(Don,Ken))
  - I can substitute the phrase
    - Uncle(Don, Kevin)

Refutable Descriptions

- When we make assertions about a domain, we want them to be refutable
  - that is, we want it to be possible that someone can prove that the assertion is wrong
- Why would we want to do that?
  - because, for one, all of science is based on that notion! Respectable scientific theories are refutable
    - if it holds up under scrutiny, people gain confidence in the theory and build new theories on top of it

Refutable Descriptions, continued

- Why would we want to do that? (cont.)
  - Second, it forces us to be explicit and clear about our assertions, at a point when we are surrounded by uncertainty
    - creating a requirement that says "The software system must be responsive."
    - This type of statement is completely useless
      - and can't be refuted; if a system takes 100 hours to respond to a button click, the above requirement has been satisfied!

Refutable Descriptions, continued

- Instead say,
  - The system must provide feedback to a button click in .5 seconds, either by displaying the requested output or by presenting a "spinning" cursor
- This is a solid, specific requirement that can be refuted

Taking risks

- Refutable Descriptions Create Risks
  - Domain descriptions describe how things are in the system's environment or application domain
    - Refutable domain descriptions run the risk of someone saying "That's not true - here's a counterexample"
    - Why is this good?
  - Requirement descriptions describe how things ought to be when the system is installed
    - Runs the risk of someone saying "No, that's not the effect I require" or, later "Yes, that was the effect I required, but the system isn't achieving it…"

The importance of designations

- In order to create refutable descriptions, you need crisp and clear designations
  - This is the importance of a recognition rule
    - it eliminates ambiguity from a domain by allowing us to classify a particular phenomena
- You then create refutable descriptions by creating a set of assertions using these crisp and clear designations
  - It lets your users find counterexamples and helps to create a shared understanding of the problem

An Example

- A *plain segment* of track is a continuous stretch of single track with its sole entry point at the *entry* of the segment, and the sole exit point at the *exit* of the segment. A *fork switch* is a configuration with one entry and two exits; and a *join switch* is a configuration with two entries and one exit. Plain segments, fork switches, and join switches are all subtypes of the type *track unit*.
- A *rail network* consists of an assemblage of track units such that each exit of each unit is connected to an entry of a different unit, and vice versa. Two units with a connected entry and exit are said to be adjacent.
- Is this a refutable description?

Another Example

- Designations
  - By time t, the total volume of chemical that has flowed into the vat through the inlet pipe is x liters ≈ Inflow(x, t)
  - By time t, the total volume of chemical that has flowed out of the vat through the outlet value is x liters ≈ Outflow(x, t)
  - At time t, the total volume of chemical in the vat is x liters ≈ Contains(x, t)
- Definition
  - Netflow(c, t) ✦ ∃ i,o • (Inflow(i,t) ∧ Outflow(o,t) ∧ (c = i - o)
- Refutable Description
  - ∀ c,t • Contains(c, t) ↔ ∃ i,o•(Inflow(i,t)∧Outflow(o,t)∧(c = i-o)

Rough Sketches

- A description of something that is only partially understood or invented
  - They record vague, half-formed ideas when you do not have the time to be precise (say because, you are being precise about some other aspect of the application domain, that day)
- The defining characteristic of a rough sketch is it vagueness; vagueness is common if a development project starts by focusing on the machine and not the application domain
  - because in that case, the machine does not exist yet!

Rough Sketches have their place

- Sometimes its impossible to be precise about some aspect of the application domain
  - the application domain is informal, after all
  - and rough sketches appear typically at the begin of a development project when the application domain is still being understood
- but often the rough sketch is the "cuckoo" in the software development nest pushing out all other types of descriptions

Class Activity Section

- Develop instances of each type of description, discussed today for
  - problem: moving people from floor to floor in a building using elevators
  - problem context: people, floors, elevators, etc.
- Goal: Model the application domain **not** the Machine
- Start with either a rough sketch or precise designations and go from there