Lecture 6: Descriptions

Kenneth M. Anderson

Object-Oriented Analysis and Design

CSCI 6448 - Spring Semester, 2003

Goals for this Lecture

- Discuss two types of descriptions
  - designations and definitions
- Discuss two states for a description
  - refutable and rough sketch
- Discuss Events and Intervals
  - adding the notion of time to descriptions
- Discuss two generic characteristics of descriptions
  - Scope and Span
- 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

Descriptions: Types and States

- Description Types
  - Designations
    - phenomena of interest
  - Definitions
    - a formal statement of a term (can be used by other descriptions)
- Description States
  - 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 (I)

- 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 (II)

- 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
    - Take, for example, 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 (III)

- 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?
  - How about: the entries of a join switch are always attached to the exit of a plain segment of track

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 situation, 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 beginning 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

Dynamic Domains

- Dynamic Domains have state
  - This state changes over time
  - To model this change, we introduce
    - events
    - intervals
  - To model states, without worrying about state changes, we can use
    - points in time

Points in Time

- p is a point in time ≈ TimePoint(p)
- The point p precedes the point q in time ≈ Precedes(p,q)
- This way of treating time lets you talk about states, about what is true in the world at a particular point in time
  - but its not particularly helpful in talking about state changes

A TimePoint Example

- A Traffic Intersection
  - The North-South light is red at time p ≈ NSRed(p)
  - The North-South light is green at time p ≈ NSGreen(p)
  - The East-West light is red at time p ≈ EWRed(p)
  - The East-West light is green at time p ≈ EWGreen(p)
- An associated description
  - $\forall p \bullet (TimePoint(p) \rightarrow ((NSRed(p) \vee EWRed(p)) \wedge (NSRed(p) \rightarrow \exists q \bullet (Precedes(p,q) \wedge NSGreen(q))) \wedge (EWRed(p) \rightarrow \exists q \bullet (Precedes(p,q) \wedge EWGreen(q)))))$

Events and Intervals

- If we want to model changes of state, we need to think about intervals of time that are split by events
  - events are atomic (they cannot be interrupted) and instantaneous (time does not progress during an event)
  - intervals occur between events
    - state does not change during an interval

Events and Intervals, Continued

- Designations
  - e is an atomic instantaneous event ≈ Event(e)
  - The event e occurs before the event f ≈ Earlier(e,f)
  - v is an interval of time in which no event occurs ≈ Intvl(v)
  - The event e begins the interval v ≈ Begins(e,v)
  - The event e ends the interval v ≈ Ends(e,v)
- Definitions
  - InitIntvl(v) ♦ Intvl(v) ∧¬ ∃ e • Begins(e,v)
  - FinalIntvl(v) ♦ Intvl(v) ∧¬ ∃ e • Ends(e,v)
- Note: the definitions make InitIntvl() and FinalIntvl() available without implying that they are true of any individual interval

Discussion

- In this model, events cannot occur at the same time
  - Earlier(a, b) means that a occurs before b
  - The restriction is that an event represents a transition of the world from one state to another without passing through any intermediate state
  - This means that any action that has an internal time structure must be regarded as two or more events
  - In addition, abstraction plays a key role
    - what is the appropriate “event” for a chess game?

Bank Example

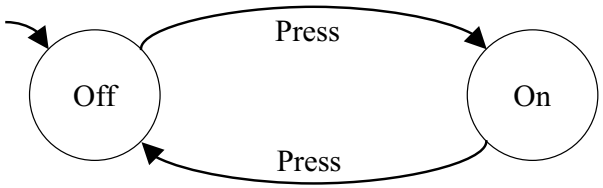
- In event e, account a is debited cash amount m ≈ Debit(e,a,m)
- In event e, account a is credited cash amount m ≈ Credit(e,a,m)
- How do you treat a transfer?
  - As a single transaction?
  - As two events?
  - What about a transfer in which the debit account and the credit account are the same?
    - Does the balance of the account dip?

Modeling State Changes

- In an event-interval view of the world, nothing changes without an event
- If a fact does not reference an interval, then it is a phenomena that does not change
  - Lecturer t teaches course c ≈ Teaches(t,c)
- In an event-interval system, this would imply that teacher t has taught course c eternally
- To give our teachers a break, try
  - Lecturer t teaches course c in interval v ≈ Teaches(t,c,v)

An Event-Interval Example

- A light bulb
  - In interval v, the light is on ≈ On(v)
  - In interval v, the light is off ≈ Off(v)
  - In event e, the button is pressed ≈ Press(e)



Associated Descriptions

- Switch must be on or off in any particular interval, its initial state is off
  - $\forall$ v • ((On(v) $\rightarrow$ $\neg$Off(v)) $\wedge$ (Intvl(v) $\rightarrow$ (On(v) $\vee$ Off(v))) $\wedge$ (InitIntvl(v) $\rightarrow$ Off(v)))
- The light changes state via Press events
  - $\forall$ v,w,e • ((Ends(e,v) $\wedge$ Begins(e,w)) $\rightarrow$ (((On(v) $\wedge$ Off(w)) $\vee$ (Off(v) $\wedge$ On(w)))) $\leftrightarrow$ Press(e))

Why do we need descriptions?

- Why do we need more than one type?
  - You will not be able to express everything about a software problem using one type of description
  - The complexity of software problems do not allow you to think about the whole problem at once
    - So, identify distinct aspects and describe them separately; this is known as **separation of concerns**

Scope

- A useful description has a carefully defined scope
  - What domain are you describing?
  - What phenomena are you describing?
  - Example: maps
    - Map of the earth or map of the moon?
    - Rainfall map or Population map?
      - For the former, you might create a designation like "The average annual rainfall in area a is r ≈ rainavg(a,r)"

More on Scope

Descriptions are connected to domains by designation sets; the designation set provides the vocabulary the description can use in describing the domain

The designation set, in fact, determines the scope of the description, since you cannot talk about non-designated phenomena…

…well you can, but then you have a rough sketch

An example

- x is a company employee in time interval y ≈ Comp(x,y)
- x is a time interval in the life of the company ≈ Intl(x)
- x is the manager of y in interval z ≈ Superior(x, y, z)

- A description can talk about the relationships among these phenomena, *but no others!*
- Comp(x,y) is not true unless Intl(y) is true; and
- Superior(x,y,z) is not true unless both Comp(x,z) and Comp(y,z) are true; and
- Superior(x1,y,z) and Superior(x2,y,z) are not both true unless x1 and x2 are the same individual; and
- In any interval z, there is exactly one individual y for which Super(x,y,z) is false for every x

Span

- A useful description also has a span
  - which is the particular subset of the designated phenomena that it includes
  - return to the Map example
    - for a population map, we need to know what part of the world's surface is depicted; and
    - we need to know whether it is the population in 1990 or in 1800 that is shown

Span Example

- "Thank you for your call. We value it highly. It will be answered in the order in which it was received."
- What's wrong with the proceeding sentence?
  - Hint: "All calls are answered in the order in which they are received."
- The scope selects a domain's individuals; the span selects a subset of those individuals
  - limiting a description's span can help to focus your attention on particular sub-problems

Setting Span

- Scope and Span are closely related;
  - in fact you can express a reduced span by introducing new terms that help to narrow your scope
  - For example
    - x is a citizen of country y ≈ CountryCitizen(x,y)
    - dCitizen ♦ CountryCitizen(x,Denmark)
  - Using dCitizen in a description reduces its span to the citizen's of Denmark (rather than the citizens of the entire world)

Setting Span: Rule of Thumb

- Choose a span of description that allows you to say exactly what you want to say
  - A smaller span would prevent you from saying everything you want
  - A larger span would force you to say too much
    - Saying too much reduces the context independence of a description

Scope and Span Benefits

- The concepts of scope and span can be very useful to a designer
  - When in the process of design, you can ask questions such as
    - Am I making the right kind of separation?
    - Am I using the wrong scope?
    - Am I using the wrong span?
  - Sometimes a problem seems hard, when in fact it is an easy problem surrounded by irrelevant and confusing material

An example

- In a Knock-Out Tennis Tournament, if the number of competitors is a power of 2 —say, 2 to the *n*th power—then there will be n rounds. In each round, half of the players in the round are eliminated, and the other half go forward to the next round, in which there will be only half as many matches. Eventually, there will be a round with only one match, and the winner of that match is the winner of the tournament. If the number of competitors is not a power of 2, then some of them miss the first round, and proceed directly to the second round; from the second round onwards, everything continues as normal.
- The question is: If there are 111 competitors, how many matches are there in the whole tournament?

Questions

- What's the smallest possible scope relating competitors and matches?
- What's the smallest span?
- What's the problem with the proceeding example? Hint: it has to do with its scope and span