Lecture 10: Descriptive Specifications

Kenneth M. Anderson
Foundations of Software Engineering
CSCI 5828 - Spring Semester, 2000

Today s Lecture

- Introduce Descriptive Specifications
  - E-R Diagrams (Semi-Formal)
  - Axiomatic
  - Algebraic
  - Tour of the RAISE system
    - Developed in Denmark
    - Sold to European Manufacturing companies
    - Using RAISE to create these types of specifications
      - Has a full tool suite

Descriptive Specifications

- Focuses on Properties
  - Describes the desired properties of a system rather than its desired behavior
- Formalisms
  - Axiomatic (Logic)
  - Algebraic

Formalisms Provide Preciseness

- Use of Mathematical Formalisms
  - Properties are specified precisely by building on top of the precise mathematical syntax and semantics of the underlying formalisms
- Mathematical Foundations
  - Predicate logic, set theory, abstract algebra

Entity-Relationship Diagrams

¥ A semi-formal notation for describing the structure and relationships of data

—Akin to how Data Flow Diagrams are a semi-formal notation for describing the operations that access and manipulate data

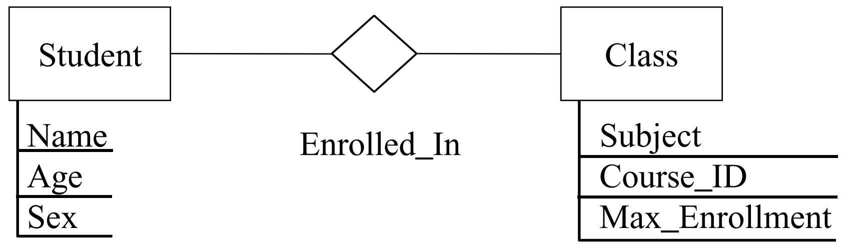
¥ Problems

—Syntax and Semantics are not precisely defined

—Lack of Expressive power

¥ requires the use of natural language annotations

Example ER Diagram



(taken from textbook page 200)

ER Diagrams and UML

¥ ER Diagrams can be seen as precursors to UML s Class Diagrams

¥ Differences

—operations and inheritance are added

¥ Advantages

—ER notation was never standardized, UML s class diagrams provide a standard notation

¥ however, remember that they are both semi-formal

Logic Specifications

¥ Vocabulary of Logical Expressions

—Variables, constants, predicates, functions

—Connectives: and ($\wedge$), or ($\vee$), not ($\neg$), implies ($\Rightarrow$), equivalent ($\equiv$)

—Quantifiers: exists ($\exists$), for all ($\forall$)

¥ Combined with Vocabulary of Application

—Example: set operators ($\in$, $\cup$, $\cap$,)

—Example: ADT operators (Push, IsFull,)

Logic Specifications

¥ Examples

—x > y and y > z implies x > z

—for all x (exists y (y = x + z))

¥ Additional Notes

—Variables are either *free* or *bound*

¥ A formula with all variables bound is called *closed*; closed formulas are always either true or false

—Expressions are *theories* in the logic

—V&V amounts to *theorem proving*

Creating Logic Specifications

¥ Helper Predicates and Functions

—Define the base properties of interest

¥ Used as a domain-specific vocabulary

—Modularize the specification

¥ e.g., defined in one spec; used in another

¥ Examples

—height(bob) = 72; tall(bob)

—for p: person (height(p)>60 implies tall(p))

Logic Specification Techniques

¥ Preconditions and Postconditions

—Textbook gives lots of examples on 204-205

Assume <i1, i2, i3, > are input values

Assume <o1, o2, o3, > are output values

¥ A property is defined

{Pre(i1, i2, i3,)}

P

{Post(o1, o2, o3, , i1, i2, i3, >}

¥ Example

{exists z (i1 = z * i2)}

P

{o1 = i1/i2}

Logic Specification Techniques

¥ Invariants and Assertions

—Logic specs are used to assert properties of portions of code as well

—For instance, to assert something that is always true of a routine or to record the assumptions about variables passed to a procedure

{n > 0}

procedure reverse (a: in out int_array; n: in int)

{for all i (1<=i<=n) implies (a(i) = old_a(n-i+1))}

Algebraic Specifications

- Make use of heterogeneous algebra
  - a collection of different sets on which several operations are defined
  - Traditional algebras are homogeneous, one set and a several operations; e.g. integers
  - Heterogeneous algebras contain multiple sets
    - e.g. length(ken) = 3
    - Here we have the set of strings and integers with one operation length defined

RAISE

Rigorous Approach to Industrial Software Engineering

- A Method and a Language
- Specification Language: RSL
- Specifications Refined in Levels
  - Associated consistency proof obligations
- Proofs of Properties Aided by Tools

Background Information

- In RAISE, they make use of a funny notion of the domain and range of a function
- Each function consists of a set of tuples. The domain is the set of elements that make up the first element of each tuple; the range is the set of elements that make up the second set of each tuple

Example

- S = {}
  - Empty Set
- S = S‘ [1 |-> 2]
  - S = {(1,2)}
  - Domain = {1}
  - Range = {2}
- S = S‘ [3 |-> 4]
  - S = {(1,2), (3,4)}
  - Domain = {1, 3}
  - Range = {2, 4}
- S = S‘ \[1]
  - S = {(3, 4)}

RAISE Specification of POTS*

* Plain Old Telephone Service

RAISE Specification of POTS

scheme POTS =

RAISE Specification of POTS

```
scheme POTS =
    class
        type


        value



        variable
```


RAISE Specification of POTS

```
scheme POTS =
    class
        type
```

RAISE Specification of POTS

scheme POTS =
  class
    type Line,

RAISE Specification of POTS

scheme POTS =
  class
    type Line,
      Status = Line $\overrightarrow{m}$ {On_Hook, Off_Hook},

RAISE Specification of POTS

scheme POTS =
  class
    type Line,
      Status = Line $\overrightarrow{m}$ {On_Hook, Off_Hook},
      Calls = Line $\overrightarrow{m}$ Line

RAISE Specification of POTS

scheme POTS =
  class
    type Line,
      Status = Line $\overrightarrow{m}$ {On_Hook, Off_Hook},
      Calls = Line $\overrightarrow{m}$ Line
    value

RAISE Specification of POTS

scheme POTS =
class
type Line,
Status = Line $\overrightarrow{m}$ {On_Hook, Off_Hook},
Calls = Line $\overrightarrow{m}$ Line
value go_off_hook : Line $\rightarrow$ Unit,

RAISE Specification of POTS

scheme POTS =
class
type Line,
Status = Line $\overrightarrow{m}$ {On_Hook, Off_Hook},
Calls = Line $\overrightarrow{m}$ Line
value go_off_hook : Line $\rightarrow$ Unit,
go_on_hook : Line $\rightarrow$ Unit,

RAISE Specification of POTS

scheme POTS =
class
type Line,
Status = Line $\overrightarrow{m}$ {On_Hook, Off_Hook},
Calls = Line $\overrightarrow{m}$ Line
value go_off_hook : Line $\rightarrow$ Unit,
go_on_hook : Line $\rightarrow$ Unit,
place_call : Line $\times$ Line $\rightarrow$ Bool,

RAISE Specification of POTS

scheme POTS =
class
type Line,
Status = Line $\overrightarrow{m}$ {On_Hook, Off_Hook},
Calls = Line $\overrightarrow{m}$ Line
value go_off_hook : Line $\rightarrow$ Unit,
go_on_hook : Line $\rightarrow$ Unit,
place_call : Line $\times$ Line $\rightarrow$ Bool,
end_call : Line $\rightarrow$ Unit

RAISE Specification of POTS

scheme POTS =
class
type Line,
Status = Line $\overrightarrow{m}$ {On_Hook, Off_Hook},
Calls = Line $\overrightarrow{m}$ Line
value go_off_hook : Line $\rightarrow$ Unit,
go_on_hook : Line $\rightarrow$ Unit,
place_call : Line $\times$ Line $\rightarrow$ Bool,
end_call : Line $\rightarrow$ Unit
variable

RAISE Specification of POTS

scheme POTS =
class
type Line,
Status = Line $\overrightarrow{m}$ {On_Hook, Off_Hook},
Calls = Line $\overrightarrow{m}$ Line
value go_off_hook : Line $\rightarrow$ Unit,
go_on_hook : Line $\rightarrow$ Unit,
place_call : Line $\times$ Line $\rightarrow$ Bool,
end_call : Line $\rightarrow$ Unit
variable line_status : Status = [L |-> On_Hook | L : Line],

RAISE Specification of POTS

scheme POTS =
class
type Line,
Status = Line $\overrightarrow{m}$ {On_Hook, Off_Hook},
Calls = Line $\overrightarrow{m}$ Line
value go_off_hook : Line $\rightarrow$ Unit,
go_on_hook : Line $\rightarrow$ Unit,
place_call : Line $\times$ Line $\rightarrow$ Bool,
end_call : Line $\rightarrow$ Unit
variable line_status : Status = [L |-> On_Hook | L : Line],
active_calls : Calls = [ ]

RAISE Specification of POTS

RAISE Specification of POTS

axiom

RAISE Specification of POTS

axiom forall L, L_1, L_2 : Line ¥

RAISE Specification of POTS

axiom forall L, L_1, L_2 : Line ¥

go_off_hook(L)

go_on_hook(L)

place_call(L_1, L_2)

end_call(L)

RAISE Specification of POTS

axiom forall L, L_1, L_2 : Line ¥

go_off_hook(L)

RAISE Specification of POTS

axiom forall L, L_1, L_2 : Line ¥
go_off_hook(L) post line_status = line_status‘ [L |-> Off_Hook],

RAISE Specification of POTS

axiom forall L, L_1, L_2 : Line ¥
go_off_hook(L) post line_status = line_status‘ [L |-> Off_Hook],

go_on_hook(L)

RAISE Specification of POTS

axiom forall L, L_1, L_2 : Line ¥
go_off_hook(L) post line_status = line_status‘ [L |-> Off_Hook],

go_on_hook(L) post line_status = line_status‘ [L |-> On_Hook],

RAISE Specification of POTS

axiom forall L, L_1, L_2 : Line ¥
go_off_hook(L) post line_status = line_status‘ [L |-> Off_Hook],

go_on_hook(L) post line_status = line_status‘ [L |-> On_Hook],

place_call(L_1, L_2) as S

RAISE Specification of POTS

axiom forall L, L_1, L_2 : Line ¥

go_off_hook(L) post line_status = line_status‘ [L |-> Off_Hook],

go_on_hook(L) post line_status = line_status‘ [L |-> On_Hook],

place_call(L_1, L_2) as S

post S $\Rightarrow L_1 \neq L_2$

RAISE Specification of POTS

axiom forall L, L_1, L_2 : Line ¥

go_off_hook(L) post line_status = line_status‘ [L |-> Off_Hook],

go_on_hook(L) post line_status = line_status‘ [L |-> On_Hook],

place_call(L_1, L_2) as S

post S $\Rightarrow L_1 \neq L_2 \wedge$ active_calls = active_calls‘ [L_1 |-> L_2]

RAISE Specification of POTS

axiom forall L, L_1, L_2 : Line ¥

go_off_hook(L) post line_status = line_status‘ [L |-> Off_Hook],

go_on_hook(L) post line_status = line_status‘ [L |-> On_Hook],

place_call(L_1, L_2) as S

post S $\Rightarrow L_1 \neq L_2 \wedge$ active_calls = active_calls‘ [L_1 |-> L_2]

$\wedge$ $L_2 \notin$ dom active_calls‘

RAISE Specification of POTS

axiom forall L, L_1, L_2 : Line ¥

go_off_hook(L) post line_status = line_status‘ [L |-> Off_Hook],

go_on_hook(L) post line_status = line_status‘ [L |-> On_Hook],

place_call(L_1, L_2) as S

post S $\Rightarrow L_1 \neq L_2 \wedge$ active_calls = active_calls‘ [L_1 |-> L_2]

$\wedge$ $L_2 \notin$ dom active_calls‘ $\wedge$ $L_2 \notin$ rng active_calls‘

RAISE Specification of POTS

axiom forall L, L_1, L_2 : Line ¥

go_off_hook(L) post line_status = line_status‘ [L |-> Off_Hook],

go_on_hook(L) post line_status = line_status‘ [L |-> On_Hook],

place_call(L_1, L_2) as S

post S $\Rightarrow L_1 \neq L_2 \wedge$ active_calls = active_calls‘ [L_1 |-> L_2]

$\wedge$ $L_2 \notin$ dom active_calls‘ $\wedge$ $L_2 \notin$ rng active_calls‘

pre

RAISE Specification of POTS

axiom forall L, L_1, L_2 : Line ¥

go_off_hook(L) post line_status = line_status‘ [L |-> Off_Hook],

go_on_hook(L) post line_status = line_status‘ [L |-> On_Hook],

place_call(L_1, L_2) as S

post S $\Rightarrow L_1 \neq L_2 \wedge$ active_calls = active_calls‘ [L_1 |-> L_2]

$\wedge$ $L_2 \notin$ dom active_calls‘ $\wedge$ $L_2 \notin$ rng active_calls‘

pre line_status(L_1) = Off_Hook

RAISE Specification of POTS

axiom forall L, L_1, L_2 : Line ¥

go_off_hook(L) post line_status = line_status‘ [L |-> Off_Hook],

go_on_hook(L) post line_status = line_status‘ [L |-> On_Hook],

place_call(L_1, L_2) as S

post S $\Rightarrow L_1 \neq L_2 \wedge$ active_calls = active_calls‘ [L_1 |-> L_2]

$\wedge$ $L_2 \notin$ dom active_calls‘ $\wedge$ $L_2 \notin$ rng active_calls‘

pre line_status(L_1) = Off_Hook

$\wedge$ $L_1 \notin$ dom active_calls

RAISE Specification of POTS

axiom forall L, L_1, L_2 : Line ¥

go_off_hook(L) post line_status = line_status‘ [L |-> Off_Hook],

go_on_hook(L) post line_status = line_status‘ [L |-> On_Hook],

place_call(L_1, L_2) as S

post S $\Rightarrow L_1 \neq L_2 \wedge$ active_calls = active_calls‘ [L_1 |-> L_2]

$\wedge$ $L_2 \notin$ dom active_calls‘ $\wedge$ $L_2 \notin$ rng active_calls‘

pre line_status(L_1) = Off_Hook

$\wedge$ $L_1 \notin$ dom active_calls $\wedge$ $L_1 \notin$ rng active_calls,

RAISE Specification of POTS

RAISE Specification of POTS

end_call(L)

RAISE Specification of POTS

end_call(L)
    post

RAISE Specification of POTS

```
end_call(L)
    post  if  L ∈ dom active_calls‘
          then
          else

          end
```

RAISE Specification of POTS

end_call(L)
post if $L \in$ dom active_calls‘
then active_calls = active_calls‘ \ { L }
else

end

RAISE Specification of POTS

end_call(L)
post if $L \in$ dom active_calls‘
then active_calls = active_calls‘ \ { L }
else $\exists$ L_3 : Line ¥

end

RAISE Specification of POTS

end_call(L)
post if $L \in$ dom active_calls‘
then active_calls = active_calls‘ \ { L }
else $\exists$ L_3 : Line ¥
active_calls‘ (L_3) = L
end

RAISE Specification of POTS

end_call(L)
post if $L \in$ dom active_calls‘
then active_calls = active_calls‘ \ { L }
else $\exists$ L_3 : Line ¥
active_calls‘ (L_3) = L $\wedge$
active_calls = active_calls‘ \ { L_3 }
end

RAISE Specification of POTS

end_call(L)
  post if $L \in$ dom active_calls‘
    then active_calls = active_calls‘ \{ L }
    else $\exists L_3$: Line ¥
      active_calls‘ (L_3) = L $\wedge$
      active_calls = active_calls‘ \{ L_3 }
    end
  pre

RAISE Specification of POTS

end_call(L)
  post if $L \in$ dom active_calls‘
    then active_calls = active_calls‘ \{ L }
    else $\exists L_3$: Line ¥
      active_calls‘ (L_3) = L $\wedge$
      active_calls = active_calls‘ \{ L_3 }
    end
  pre $L \in$ dom active_calls

RAISE Specification of POTS

end_call(L)
  post if $L \in$ dom active_calls‘
    then active_calls = active_calls‘ \{ L }
    else $\exists L_3$: Line ¥
      active_calls‘ (L_3) = L $\wedge$
      active_calls = active_calls‘ \{ L_3 }
    end
  pre $L \in$ dom active_calls $\vee$ $L \in$ rng active_calls

RAISE Specification of POTS

end_call(L)
  post if $L \in$ dom active_calls‘
    then active_calls = active_calls‘ \{ L }
    else $\exists L_3$: Line ¥
      active_calls‘ (L_3) = L $\wedge$
      active_calls = active_calls‘ \{ L_3 }
    end
  pre $L \in$ dom active_calls $\vee$ $L \in$ rng active_calls
end