#### 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

February 17, 200 'Kenn eth M. Anderson, 2000

#### **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

2

#### **Entity-Relationship Diagrams** Example ER Diagram ¥ A semi-formal notation for describing the structure and relationships of data Student Class -Akin to how Data Flow Diagrams are a semi-formal notation for describing the operations that access and N<u>ame</u> Subject Enrolled In manipulate data Course ID Age ¥ Problems Max\_Enrollment Sex -Syntax and Semantics are not precisely defined -Lack of Expressive power (taken from textbook page 200) ¥ requires the use of natural language annotations 'Kenn eth M. Anderson, 2000 5 February 17, 2000 February 17, 2000 'Kenn eth M. Anderson, 2000 6 ER Diagrams and UML Logic Specifications ¥ ER Diagrams can be seen as precursors to ¥ Vocabulary of Logical Expressions UML s Class Diagrams -Variables, constants, predicates, functions —Connectives: and ( $\land$ ), or ( $\lor$ ), not ( $\neg$ ), ¥ Differences implies ( $\Rightarrow$ ), equivalent ( $\equiv$ ) -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

February 17, 2000

7

February 17, 2000

'Kenn eth M. Anderson, 2000

-Example: ADT operators (Push, IsFull, )

¥ Combined with Vocabulary of Application

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

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

#### Logic Specifications

¥ Examples			¥ Helper Predicates and Functions		
-x > y and	y > z implies $x > z$		—Define the base properties of interest		
—for all x (	exists $y (y = x + z))$		¥ Used as a domain-specific vocabulary		
¥ Additional	Notes		—Modularize the specification		
<ul> <li>—Variables are either <i>free</i> or <i>bound</i></li> <li>¥ A formula with all variables bound is called <i>closed</i>; closed formulas are always either true or false</li> </ul>			$\forall e \sigma$ defined in one spectrused in another		
			¥ Examples		
—Expressions are <i>theories</i> in the logic			height(bob) = 72; tall(bob)		
—V&V am	ounts to <i>theorem proving</i>		-for p: person (height(p)>60 implies tall(p))		
February 17, 2000	'Kenn eth M. Anderson, 2000	9	February 17, 2000 'Kenn eth M. Anderson, 2000		

## Logic Specification Techniques

	¥ A property is defined
Postconditions	{Pre(i1, i2, i3, )}
—Textbook gives lots of	Р
examples on 204-205	{Post(01, 02, 03, , i1,
Assume <i1, i2,="" i3,=""></i1,>	i2, i3, >}
are input values	¥ Example
Assume <01, 02, 03, >	$\{ exists z (i1 = z * i2) \}$
are output values	P
	$\{o1 = i1/i2\}$

#### Logic Specification Techniques

Creating Logic Specifications

#### ¥ 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
  - $\{n \ge 0\}$

procedure reverse (a: in out int\_array; n: in int) {for all i (1<=i<=n) implies  $(a(i) = old_a(n-i+1))$ }

11

February 17, 2000

'Kenn eth M. Anderson, 2000

10

#### 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

'Kenn eth M. Anderson, 2000

#### 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

February 17, 2000

'Kenn eth M. Anderson, 2000

14

# **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

$F S = \{\}$		¥ Empty Set
$\mathbf{F} \mathbf{S} = \mathbf{S}^{\mathbf{G}}$	[1  ->2]	$F S = \{(1,2)\}$
		$\mathbf{F}$ Domain = {1}
		$\Upsilon$ Range = {2}
$\mathbf{F} \mathbf{S} = \mathbf{S}^{\mathbf{G}}$	[3  -> 4]	$\mathbf{Y} \mathbf{S} = \{(1,2), (3,4)\}$
		$# Domain = \{1, 3\}$
		$# Range = \{2, 4\}$
F S = S'	\[1]	$F S = \{(3, 4)\}$

February 17, 2000

15

13

February 17, 2000

'Kenn eth M. Anderson, 2000

RAISE Specification of POTS* * Plain Old Telephone Service	RAISE Specification of POTS scheme POTS =		
February 17, 2000 'Kenn eth M. Anderson, 2000 17	February 17, 2000 'Kenn eth M. Anderson, 2000 18		
RAISE Specification of POTS scheme POTS = class type	RAISE Specification of POTS scheme POTS = class type		
value			
variable			
February 17, 2000 'Kenn eth M. Anderson, 2000 19	February 17, 2000 'Kenn eth M. Anderson, 2000 20		

RAISE Specification of POTS = class type Line,	<pre>RAISE Specification of POTS scheme POTS =     class     type Line,     Status = Line m {On_Hook, Off_Hook},</pre>
February 17, 2000 'Kenn eth M. Anderson, 2000 21	February 17, 2000 'Kenn eth M. Anderson, 2000 22
RAISE Specification of POTS scheme POTS = class type Line, Status = Line m {On_Hook, Off_Hook}, Calls = Line m Line	RAISE Specification of POTS scheme POTS = class type Line, Status = Line $\vec{m}$ {On_Hook, Off_Hook}, Calls = Line $\vec{m}$ Line value

<pre>RAISE Specification of POTS scheme POTS =     class     type Line,         Status = Line m {On_Hook, Off_Hook},         Calls = Line m Line     value go_off_hook : Line → Unit,</pre>	$\begin{array}{ll} \textbf{RAISE Specification of POTS} \\ \textbf{scheme POTS} = \\ \textbf{class} \\ \textbf{type} & \textbf{Line,} \\ & \textbf{Status} = \textbf{Line} \ \overrightarrow{m} \{ \textbf{On_Hook, Off_Hook} \}, \\ & \textbf{Calls} = \textbf{Line} \ \overrightarrow{m} \textbf{Line} \\ \textbf{value} & \textbf{go_off_hook} : \textbf{Line} \rightarrow \textbf{Unit}, \\ & \textbf{go_on_hook} : \textbf{Line} \rightarrow \textbf{Unit}, \end{array}$
February 17, 2000 'Kenn eth M. Anderson, 2000 25	February 17, 2000 'Kenn eth M. Anderson, 2000 26
$\begin{array}{ll} \textbf{RAISE Specification of POTS} \\ \textbf{scheme POTS} = \\ \textbf{class} \\ \textbf{type} & \textbf{Line}, \\ \textbf{Status} = \textbf{Line} \ \overrightarrow{m} \{ \textbf{On_Hook}, \textbf{Off_Hook} \}, \\ \textbf{Calls} = \textbf{Line} \ \overrightarrow{m} \textbf{Line} \\ \textbf{value} & \textbf{go_off_hook} : \textbf{Line} \rightarrow \textbf{Unit}, \\ \textbf{go_on_hook} : \textbf{Line} \rightarrow \textbf{Unit}, \\ \textbf{place_call} : \textbf{Line} \times \textbf{Line} \rightarrow \textbf{Bool}, \end{array}$	$\begin{array}{llllllllllllllllllllllllllllllllllll$

27

February 17, 2000

'Kenn eth M. Anderson, 2000

28

February 17, 2000

'Kenn eth M. Anderson, 2000

RAIS scheme POTS class type	E Specification of PC S = Line, Status = Line $\vec{m}$ {On_Hook, Off_H	OTS	RAISE scheme POTS class type	E Specification of I = Line, Status = Line $\vec{m}$ {On_Hook, Off	POTS
value	Calls = Line $\overrightarrow{m}$ Line go_off_hook : Line $\rightarrow$ Unit, go_on_hook : Line $\rightarrow$ Unit, place_call : Line $\times$ Line $\rightarrow$ Bool, end_call : Line $\rightarrow$ Unit		value	Calls = Line $\overrightarrow{m}$ Line go_off_hook : Line $\rightarrow$ Unit, go_on_hook : Line $\rightarrow$ Unit, place_call : Line $\times$ Line $\rightarrow$ Bool end_call : Line $\rightarrow$ Unit	l,
variab	le		variable	e line_status : Status = $[L \rightarrow On]$	_Hook   L : Line ],
February 17, 2000	'Kenn eth M. Anderson, 2000	29	February 17, 2000	'Kenn eth M. Anderson, 2000	30
RAIS	E Specification of PC	DTS	RAISE	E Specification of I	POTS
scheme POTS class type value value	Line, Status = Line $\vec{m}$ {On_Hook, Off_H Calls = Line $\vec{m}$ Line go_off_hook : Line $\rightarrow$ Unit, go_on_hook : Line $\rightarrow$ Unit, place_call : Line $\times$ Line $\rightarrow$ Bool, end_call : Line $\rightarrow$ Unit le line_status : Status = [L -> On_Ho active_calls : Calls = []	ook},			20
February 17, 2000	'Kenn eth M. Anderson, 2000	31	February 17, 2000	'Kenn eth M. Anderson, 2000	32

<b>RAISE</b> axiom	Specification of P	OTS	<b>RAISE</b> axiom forall L, L <sub>1</sub>	Specification of F , L <sub>2</sub> : Line ¥	POTS
February 17, 2000	'Kenn eth M. Anderson, 2000	33	February 17, 2000	'Kenn eth M. Anderson, 2000	34
RAISE axiom forall L, L go_off_hook( go_on_hook(I place_call(L <sub>1</sub> , end_call(L)	<pre>Specification of P 1, L<sub>2</sub> : Line ¥ L) L) L)</pre>	OTS	RAISE axiom forall L, L <sub>1</sub> go_off_hook(l	Specification of F , L <sub>2</sub> : Line ¥ L)	POTS

<pre>RAISE Specification of POTS axiom forall L, L<sub>1</sub>, L<sub>2</sub> : Line ¥ go_off_hook(L) post line_status = line_status' [L -&gt; Off_Hook ],</pre>	<pre>RAISE Specification of POTS axiom forall L, L<sub>1</sub>, L<sub>2</sub> : Line ¥    go_off_hook(L) post line_status = line_status' [L -&gt;    Off_Hook ],    go_on_hook(L)</pre>
February 17, 2000 'Kenn eth M. Anderson, 2000 37	February 17, 2000 'Kenn eth M. Anderson, 2000 38
<pre>RAISE Specification of POTS axiom forall L, L<sub>1</sub>, L<sub>2</sub> : Line ¥ go_off_hook(L) post line_status = line_status' [L -&gt; Off_Hook], go_on_hook(L) post line_status = line_status' [L  -&gt; On_Hook],</pre>	RAISE Specification of POTS         axiom forall L, L <sub>1</sub> , L <sub>2</sub> : 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 <sub>1</sub> , L <sub>2</sub> ) as S

39

February 17, 2000

'Kenn eth M. Anderson, 2000

40

February 17, 2000

'Kenn eth M. Anderson, 2000

<b>RAISE Specification of POTS</b>	RAISE Specification of POTS
axiom forall L, L <sub>1</sub> , L <sub>2</sub> : Line ¥ go_off_hook(L) post line_status = line_status' [L -> Off_Hook],	axiom forall L, L <sub>1</sub> , L <sub>2</sub> : 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 ],	go_on_hook(L) post line_status = line_status' [L  -> On_Hook ],
place_call(L <sub>1</sub> , L <sub>2</sub> ) as S post S $\Rightarrow$ L <sub>1</sub> $\neq$ L <sub>2</sub>	place_call(L <sub>1</sub> , L <sub>2</sub> ) as S post S $\Rightarrow$ L <sub>1</sub> $\neq$ L <sub>2</sub> $\land$ active_calls = active_calls' [L <sub>1</sub>  -> L <sub>2</sub> ]
February 17, 2000 'Kenn eth M. Anderson, 2000 41	February 17, 2000 'Kenn eth M. Anderson, 2000 42
RAISE Specification of POTS	RAISE Specification of POTS
axiom forall L, L <sub>1</sub> , L <sub>2</sub> : Line ¥ go_off_hook(L) post line_status = line_status' [L  -> Off_Hook],	axiom forall L, L <sub>1</sub> , L <sub>2</sub> : 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 ],	go_on_hook(L) post line_status = line_status' [L  -> On_Hook ],
place_call(L <sub>1</sub> , L <sub>2</sub> ) as S post S $\Rightarrow$ L <sub>1</sub> $\neq$ L <sub>2</sub> $\land$ active_calls = active_calls' [L <sub>1</sub>  -> L <sub>2</sub> ] $\land$ L <sub>2</sub> $\notin$ dom active_calls'	place_call(L <sub>1</sub> , L <sub>2</sub> ) as S post S $\Rightarrow$ L <sub>1</sub> $\neq$ L <sub>2</sub> $\land$ active_calls = active_calls' [L <sub>1</sub>  -> L <sub>2</sub> ] $\land$ L <sub>2</sub> $\notin$ dom active_calls' $\land$ L <sub>2</sub> $\notin$ rng active_calls'
February 17, 2000 'Kenn eth M. Anderson, 2000 43	February 17, 2000 'Kenn eth M. Anderson, 2000 44

<pre>RAISE Specification of POTS axiom forall L, L<sub>1</sub>, L<sub>2</sub> : Line ¥ go_off_hook(L) post line_status = line_status' [L -&gt; Off_Hook ], go_on_hook(L) post line_status = line_status' [L  -&gt; On_Hook ], place_call(L<sub>1</sub>, L<sub>2</sub>) as S</pre>	RAISE Specification of POTSaxiom 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 $\Rightarrow$ Lu $\neq$ La $\diamond$ active calls = active calls' [L_1 $\rightarrow$ L_1			
$post S \Rightarrow L_1 \neq L_2 \land active\_calls = active\_calls' [L_1 \models L_2]$ $\land L_2 \notin dom active\_calls' \land L_2 \notin rng active\_calls'$ pre February 17, 2000 'Kenn eth M. Anderson, 2000 45	$post S \Rightarrow L_{1} \neq L_{2} \land active\_calls = active\_calls' [L_{1}   -> L_{2}]$ $\land L_{2} \notin dom active\_calls' \land L_{2} \notin rng active\_calls'$ $pre \ line\_status(L_{1}) = Off\_Hook$ February 17, 2000 'Kenn eth M. Anderson, 2000 46			
RAISE Specification of POTS	RAISE Specification of POTS			
axiom forall L, L <sub>1</sub> , L <sub>2</sub> : Line ¥ go_off_hook(L) post line_status = line_status' [L -> Off_Hook],	axiom forall L, L <sub>1</sub> , L <sub>2</sub> : 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 ],	go_on_hook(L) post line_status = line_status [ L  -> On_Hook ],			
$\begin{array}{l} place\_call(L_1, L_2) \ as \ S\\ post \ S \Rightarrow L_1 \neq L_2 \land active\_calls = active\_calls' \ [L_1 \mid \rightarrow L_2 ]\\ \land \ L_2 \not\in \ dom \ active\_calls' \ \land \ L_2 \not\in \ rng \ active\_calls'\\ pre \ line\_status(L_1) = Off\_Hook\\ \land \ L_1 \not\in \ dom \ active\_calls\\ February 17, 2000 \end{array} $	$\begin{array}{l} place\_call(L_1, L_2) \ as \ S\\ post \ S \Rightarrow L_1 \neq L_2 \land active\_calls = active\_calls' \ [L_1 \mid \rightarrow L_2 ]\\ \land \ L_2 \notin dom \ active\_calls' \land \ L_2 \notin rng \ active\_calls'\\ pre \ line\_status(L_1) = Off\_Hook\\ \land \ L_1 \notin dom \ active\_calls \land \ L_1 \notin rng \ active\_calls,\\ February 17, 2000 \ \land \ L_1 \notin rng \ active\_calls, \\ \end{array}$			

RAISE	Specification of P	OTS	RAISE end_call(L)	Specification of F	POTS
February 17, 2000	'Kenn eth M. Anderson, 2000	49	February 17, 2000	'Kenn eth M. Anderson, 2000	50
RAISE end_call(L) post	Specification of P	OTS	RAISE end_call(L) post if L e then else end	Specification of F ∈ dom active_calls'	POTS
February 17, 2000	'Kenn eth M. Anderson, 2000	51	February 17, 2000	'Kenn eth M. Anderson, 2000	52

<pre>end_call(L)   post if L ∈ dom active_calls'     then active_calls = active_calls' \{ L }     else     end</pre>	$\begin{array}{l} \text{RAISE Specification of POTS} \\ \text{end\_call(L)} \\ \text{post if } L \in \text{ dom active\_calls'} \\ \text{then active\_calls} = \text{active\_calls'} \setminus \{ L \} \\ \text{else } \exists \ L_3 : \text{Line } \ \ \ \\ \text{end} \end{array}$
February 17, 2000 'Kenn eth M. Anderson, 2000 53	February 17, 2000 'Kenn eth M. Anderson, 2000 54
$\begin{array}{l} \text{Product} \text{Product} \\ $	$\begin{array}{l} \textbf{RAISE Specification of POTS} \\ \textbf{end_call(L)} \\ \textbf{post if } L \in \textbf{dom active_calls'} \\ \textbf{then active_calls} = active_calls' \setminus \{ L \} \\ \textbf{else } \exists \ L_3 : Line \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \$

