Specification

Some slides for Chapter 5
Of Ghezzi, Jazayeri, and Mandrioli
Outline

Types of specifications
  – operational
    • Data Flow Diagrams
    • UML diagrams - use-case diagrams and sequence diagrams
    • Finite State Machines
  – descriptive
    • Entity Relationship Diagrams
    • Logic-based notations
    • Algebraic notations
Specification

• A broad term that means *definition*
• Used at different stages of software development for different purposes
• Generally, a statement of agreement (*contract*) between
  – producer and consumer of a service
  – implementer and user
• Precisely describe as many desirable qualities as possible
Uses of specification

• Statement of user requirements
  – major failures occur because of misunderstandings between the producer and the user
  – "The hardest single part of building a software system is deciding precisely what to build" (Fred Brooks)
Uses of specification (cont.)

• Statement of the interface between the machine and the controlled environment
  – serious undesirable effects can result due to misunderstandings between software engineers and domain experts about the phenomena affecting the control function to be implemented by software
Uses of specification (cont.)

• Statement of requirements for implementation
  – design process is a chain of specification (i.e., definition)—implementation—verification steps
    • requirements specification refers to definition of external behavior
      – design specification must be verified against it
    • design specification refers to definition of the software architecture
      – code must be verified against it
Uses of specification (cont.)

- A reference point during maintenance
  - corrective maintenance only changes implementation
  - adaptive and perfective maintenance occur because of requirements changes
    - requirements specification must change accordingly
Specification qualities

• Precise, clear, unambiguous
• Consistent
• Complete
  – internal completeness
  – external completeness
• Incremental
Clear, unambiguous, understandable

- Example: specification fragment for a word-processor

Selecting is the process of designating areas of the document that you want to work on. Most editing and formatting actions require two steps: first you select what you want to work on, such as text or graphics; then you initiate the appropriate action.

*can an area be scattered?*
Precise, unambiguous, clear

• Another example (from a real safety-critical system)

The message must be triplicated. The three copies must be forwarded through three different physical channels. The receiver accepts the message on the basis of a two-out-of-three voting policy.

can a message be accepted as soon as we receive 2 out of 3 identical copies of message or do we need to wait for receipt of the 3rd?
Consistent

• Example: specification fragment for a word-processor

The whole text should be kept in lines of equal length. The length is specified by the user. Unless the user gives an explicit hyphenation command, a carriage return should occur only at the end of a word.

What if the length of a word exceeds the length of the line?
Complete

• Internal completeness
  – the specification must define any new concept or terminology that it uses
    • glossary helpful for this purpose
  – the specification must document all the needed requirements
    • difficulty: when should one stop?
Incremental

• Referring to the specification process
  – start from a sketchy document and progressively add details

• Referring to the specification document
  – document is structured and can be understood in increments
Classification of specification styles

- Informal, semi-formal, formal
- Operational
  - Behavior specification in terms of some abstract machine
- Descriptive
  - Behavior described in terms of properties
Example 1

• Specification of a geometric figure E:

E can be drawn as follows:
1. Select two points P1 and P2 on a plane
2. Get a string of a certain length and fix its ends to P1 and P2
3. Position a pencil as shown in next figure
4. Move the pen clockwise, keeping the string tightly stretched, until you reach the point where you started drawing

this is an operational specification
A descriptive specification

• Geometric figure E is described by the following equation
  
  \[ ax^2 + by^2 + c = 0 \]

  where \( a, b, \) and \( c \) are suitable constants
Another example

“Let a be an array of n elements. The result of its sorting is an array b of n elements such that the first element of b is the minimum of a (if several elements of a have the same value, any one of them is acceptable); the second element of b is the minimum of the array of n-1 elements obtained from a by removing its minimum element; and so on until all n elements of a have been removed.”

“The result of sorting array a is an array b which is a permutation of a and is sorted.”
How to verify a specification?

• “Observe” dynamic behavior of specified system (simulation, prototyping, “testing” specs)
• Analyze properties of the specified system
• Analogy with traditional engineering
  – physical model of a bridge
  – mathematical model of a bridge
Data Flow Diagrams (DFDs)

- A semi-formal operational specification
- System viewed as collection of data manipulated by “functions”
- Data can be persistent
  - they are stored in *data repositories*
- Data can flow
  - they are represented by *data flows*
- DFDs have a graphical notation
Patient monitoring systems

The purpose is to monitor the patients’ vital factors -- blood pressure, temperature, ...--reading them at specified frequencies from analog devices and storing readings in a database. If readings fall outside the range specified for patient or device fails an alarm must be sent to a nurse. The system also provides reports.
A refinement

Patient archive

Update archive

Recent Data

Data for Report

Generate Report

Report Request

Nurse

Formatted data

Central Monitoring

Limits

Alarm

Nurse

Patient data

Local Monitoring

Limits for patient

Clinical Data

Patient
UML use-case diagrams

- Define functions on basis of actors and actions
UML sequence diagrams

- Describe how objects interact by exchanging messages
- Provide a dynamic view
From Martin Fowler’s book UML Distilled showing centralized control
From Martin Fowler’s book UML Distilled showing distributed control
Finite state machines (FSMs)

- Can specify control flow aspects
- For example, a lamp

![Finite State Machine Diagram]

Push switch

On

Off

Push switch
Another example: a plant control system

![Diagram showing a plant control system with high-pressure and high-temperature alarms, and states On and Off.](image-url)
Declarative specifications

ER diagrams: semiformal specs
Logic specifications
Algebraic specifications
ER diagrams

• Often used as a complement to DFD to describe conceptual data models
• Based on entities, relationships, attributes
• They are the ancestors of class diagrams in UML
Relations

- Relations can be partial
- They can be annotated to define
  - one to one
  - one to many
  - many to one
  - many to many
Non binary relations

Diagram:
- Department
- Director
- Employee
- Project

Relationships:
- Data
- Duration
- HeadOf
- Assigned
- Participate
Logic specifications

Examples of first-order theory (FOT) formulas:
• \( x > y \) and \( y > z \) implies \( x > z \)
• \( x = y \equiv y = x \)
• for all \( x, y, z \) (\( x > y \) and \( y > z \) implies \( x > z \))
• \( x + 1 < x - 1 \)
• for all \( x \) (exists \( y \) (\( y = x + z \)))
• \( x > 3 \) or \( x < -6 \)
Example

• Program to compute greatest common divisor

\{i_1 > 0 \text{ and } i_2 > 0\}

P

\{(\exists z_1, z_2 \ (i_1 = o \cdot z_1 \text{ and } i_2 = o \cdot z_2) \text{ and not } (\exists h \ 
(\exists z_1, z_2 \ (i_1 = h \cdot z_1 \text{ and } i_2 = h \cdot z_2) \text{ and } h > o))\}
Specifying complete programs

A property, or requirement, for P is specified as a formula of the type

\{\text{Pre } (i_1, i_2, \ldots, \text{ in}) \} \\
P \\
\{\text{Post } (o_1, o_2, \ldots, o_m, i_1, i_2, \ldots, \text{ in})\}

Pre: precondition
Post: postcondition
Specifying procedures

\{n > 0\} -- n is a constant value
procedure search (table: in integer_array; n: in integer;
    element: in integer; found: out Boolean);
\{found \equiv (\text{exists } i \ (1 \leq i \leq n \text{ and } table (i) = element)\}\}

\{n > 0 \}
procedure reverse (a: in out integer_array; n: in integer);
\{\text{for all } i \ (1 \leq i \leq n) \text{ implies } (a (i) = old-a (n - i +1))\}\
Specifying classes

- Invariant predicates and pre/post conditions for each method
- Example of invariant specifying an array implementing ADT set

for all i, j (1 ≤ i ≤ length and 1 ≤ j ≤ length and i ≠ j) implies IMPL[i] ≠ IMPL[j]
(no duplicates are stored)
Descriptive specs

• The system and its properties are described in the same language
• Proving properties, however, cannot be fully mechanized for most languages
Algebraic specifications

• Define a *heterogeneous* algebra
• Heterogeneous = more than 1 set
• Especially useful to specify ADTs
Example

• A system for strings, with operations for
  – creating new, empty strings (operation new)
  – concatenating strings (operation append)
  – adding a new character at the end of a string (operation add)
  – checking the length of a given string (operation length)
  – checking whether a string is empty (operation isEmpty)
  – checking whether two strings are equal (operation equal)
Specification: syntax

algebra StringSpec;
introduces
  sorts String, Char, Nat, Bool;
operations
  new: () → String;
  append: String, String → String;
  add: String, Char → String;
  length: String → Nat;
  isEmpty: String → Bool;
  equal: String, String → Bool
Specification: properties

constrains new, append, add, length, isEmpty, equal so that for all [s, s1, s2: String; c: Char]
    isEmpty (new ()) = true;
    isEmpty (add (s, c)) = false;
    length (new ()) = 0;
    length (add (s, c)) = length (s) + 1;
    append (s, new ()) = s;
    append (s1, add (s2,c)) = add (append (s1,s2),c);
    equal (new (),new ()) = true;
    equal (new (), add (s, c)) = false;
    equal (add (s, c), new ()) = false;
    equal (add (s1, c), add (s2, c) = equal (s1,s2);
end StringSpec.
Requirements for a notation

• Ability to support separation of concerns
  – e.g., separate functional specs from
    • performance specs
    • user-interface specs
    • ...

• Support different views
Specifications for the end-user

• Specs should be used as common reference for producer and user
• They help removing ambiguity, incompleteness, …
• Can they be understood by end-user?
  – They can be the starting point for a prototype
  – They can support some form of animation
Conclusions

• Specifications describe
  – what the users need from a system (requirements specification)
  – the design of a software system (design and architecture specification)
  – the features offered by a system (functional specification)
  – the performance characteristics of a system (performance specification)
  – the external behavior of a module (module interface specification)
  – the internal structure of a module (internal structural specification)
Conclusions

• Descriptions are given via suitable notations
  – There is no “ideal” notation
• They must be modular
• They support communication and interaction between designers and users