SE 1: Software Requirements Specification and Analysis

*Lecture 4: Basic Notations*

Nancy Day, Davor Svetinović


uw.cs.cs445
Announcements

- Send your group to
  cs445@student.cs.uwaterloo.ca
  by Friday, 13 Jan 2006 2:00pm.

- Grad students need to have their topics chosen and
  approved by Friday, 13 Jan 2006.

Assignments of TAs to groups will be posted to the
newsgroup. It is your responsibility to contact your group TA
to arrange meetings.
Review: Use Cases

- An actor is an external entity that interacts directly with the system.

- “A use case represents a series of interactions between an outside entity and the system, which ends by providing business value.” It is a collection of related success and failure scenarios that describe actors using a system to support a goal (an observable result of value to a particular actor).

- A scenario is specific sequence of actions and interactions between actors and system. A scenario is also called a use case instance. A scenario is one path through a use case.

A use case should capture an elementary business processes (EBPs): “a task with measurable business value performed by one actor in one place at one time ”.
Review: Use Case Descriptions

- Use case number
- Name
- Authors
- Event/Precondition
- System
- Actors
- Overview/Postcondition
- References
- Related Use Cases
- Typical Process Description
  - Basic Flow
  - Alternatives
  - Exceptions
  - Subsections
Review: Use Case Diagram

- Describes the set of all use cases graphically
- Shows relationships between actors and use cases
- Shows relationships between use cases
  - includes
- Shows the boundary of the system
Today’s Agenda

- Use cases (con’d from last class)
  - Finding use cases
  - Advanced use case modelling
  - Rules of thumb

- What aspects of the system do we need to describe?
- What are the features of a good specification notation?
Today’s Agenda

- Basic notations:
  - Event traces
  - Entity-relationship diagrams (ERDs)
  - Data flow diagrams (DFDs)
  - Finite state machines (FSM)
- Decomposition strategies
- RE modelling processes (relevant to this course)

We will concentrate on notations used to describe the functional (as opposed to non-functional) behaviour of a system.

Required Reading: None

In a specification, there are four aspects of a system that we want to describe:

**functions**: tasks that the system carries out

**communication**: interactions that the system has with its environment

**behaviour**: the permissible orderings of the functions

**conceptual (data)**: description of the data manipulated by the system

Different notations are better or worse at describing these different aspects of a system.
Later, we will discuss methods for system decomposition.
Specification Notations

What are the features of a good specification notation?

- Well-defined set of concepts
- Results in unambiguous, clear, precise, concise specification
- Ability to analyze (syntax, consistency checking)
- CASE tool support
- Understood by all stakeholders (graphical is good)
- Supports traceability

Other desirable, (but rarely obtainable):

- Can generate test cases
- Can determine conformance of implementation to specification
- Can check consistency of one specification with another
Basic Specification Notations

Most specification languages are a combination of notations used to describe different aspects of the system. Each notation emphasizes one of function, communication, or behavioural descriptions. The following are very common basic notations:

- Event Traces – communication, bit of behaviour
- Entity-relationship diagrams (ERDs) – conceptual
- Data flow diagrams (DFDs) – communication, bit of functional
- Extended finite state machines (FSMs) – behaviour
- Use case descriptions – functional

A trait of an engineering discipline is that there are “standard” notations.
Example: Turnstile

- GOAL: charge admission to a park or zoo
- Physical barrier that unlocks when a token is inserted into turnstile
- Buzzer sounds whenever the barrier is unlocked
- Want to keep track of the number of visitors to park
- Display the number of visitors upon request
Event Traces

- Graphical description of communication/events in one scenario
- Messages between system and its environment (or among system entities)
- Relative ordering in time (not absolute)
- Emphasize communication; show a bit of behaviour (ordering among messages), but not complete behaviour

Entity 1 | Entity 2
---|---
Time

vertical line = time line of entity/actor/object
horizontal line = event/message from one entity to another
Event Traces

Variations:

- Show iteration
- Associate conditions with an interaction
- Self-call (sending a message to one’s self)
- Asynchronous communication (non-blocking - i.e., continue behaviour without waiting for a reply)
Event Traces

Advantages:

- Simple (one scenario)
- Easy to understand
- Somewhat precise (although only relative timing)

Disadvantages:

- Not an efficient way to represent behaviour

Useful for elicitation and consideration of end-to-end system behaviour.
ERDs

ERD = Entity-relationship diagram

- Originally for database design (Chen, 1976)
- Emphasize concepts/data
- Graphical description of problem domain:
  - Entities (think “objects”)
  - Relationships (often named with verbs)
    - Relationships can represent communication, part-of, visibility, etc
  - At first, entities are concepts in the environment. After refinement, we separate concepts in the environment from concepts (i.e., components) of the system to show conceptual decomposition

In OO terms, the problem domain is called the conceptual level of description.

- Origin of class diagrams (UML)
ERDs - Symbols

Entity

Entity – (class) collection of real-world individuals with common properties

Rel

Relationship between entities

attr

Attributes – property
Variations

- **Cardinality** on relationships (1-many, many-many)
- Attributes on relationships
- Subtypes (inheritance)
- Directions on associations to indicate navigation
ERDs

Advantages:

- Simple (few symbols)

Disadvantages:

- What to model as entities vs attributes?
- May encourage too detailed modelling (i.e. getting into design details if used for too much decomposition at the requirements level)
Data Flow Diagrams (DFDs)

- Graphical description of flow of data among components
- Emphasize communication, bit of functional

Legend:

- Object – data source or sink
typically used at the start or end of data flow

- Function/Process – action, transformation of data

- Data Flow – labelled with data

- Data Store – internal (i.e., system) data source or sink
  NOT a req. concept
DFDs

Advantages:

- Simple
- Can be hierarchical and thereby also show communication among the components in a functional decomposition of the system

Disadvantages:

- Ambiguities:
  - When do functions get executed?
  - If multiple inputs, are they all needed?
  - Can’t distinguish data and control signals
  - If multiple outputs, are both always generated?
  - i.e., they are really only good for show communication!
A context diagram is a DFD where the system is represented as one function. It shows all the inputs and outputs of the system and which environmental entities they come from and go to. The context diagram shows the scope/boundary of the system.

A context diagram can also be done for major subsystems.
FSMs

- Describes control flow:
  - Behaviour depends on current state
  - State represents the history of input

- Graphical description of dialog between system and environment

- Emphasizes behaviour: shows order in which functions can be executed and order of communication

- Compact representation of a set of event traces

Legend:

- State – represents history of dialog; state of system/environment

- State Transition – triggered by input event; generates output
FSMs

Variations:

- **Hierarchy used to:**
  - Represent concurrent operations
  - Priority (interrupts, etc)

- Can be presented in a tabular format

- Communication mechanisms between multiple FSMs
  - Broadcast vs directed
  - Queues (delayed)
FSMs

Advantages:

- Most formal (least ambiguity)
- Compact way to represent many behaviours

Disadvantages:

- More complex
- Doesn’t show functional decomposition
Summary

These notations all represent different aspects of the system’s behaviour (views):

- Conceptual/structural/static (ERDs)
- Communication (DFDs, event traces)
- Behavioural/dynamic (event traces, FSMs)
- Functional (use case descriptions, pre/post conditions)

Most methods use a combination of variations on the above notations.

Note: FSMs specify behaviour; traces illustrate behaviour.
Decomposition Strategies

The purpose of decomposition during requirements analysis is to make the specification as simple and understandable as possible. It is not a decomposition for how you might implement the system.

Three common strategies for system decomposition are:

- **Functional**: break functions into subfunctions (e.g., Structured Analysis)

- **Process**: break functions into subfunctions, but at the lowest level functions can run concurrently (processes). (e.g., SDL)

- **Object Oriented**: break system into objects. (e.g., UML)
  - What are the responsibilities of each object?
  - How do objects interact?
Specification Languages

- Use Case Descriptions
- Use Case Diagrams – high-level DFDs, part of UML
- **UML** (Unified Modelling Language)
  - Object-oriented decomposition
  - Class and Object Diagrams – form of ERDs
  - Interaction Diagrams (Sequence and Communication Diagrams) – forms of event traces
  - State Diagrams – form of FSMs
  - UML has many other types of diagrams (12 diagram types right now?)
Specification Languages

- **SDL** (Specification and Description Language)
  - Focus is on precise specification of communication
  - Process-based decomposition
  - Essentially hierarchical data flow diagrams
  - At the lowest level, uses a form of FSMs representing processes
  - Message Sequence Charts (MSCs) – not officially part of SDL, but often used with it (form of event traces)
Specification Languages

- Algebraic specification (part of SDL)
  - Functions
  - Formal
  - Abstract

- Temporal Logic
  - Behavioural
  - Formal
  - Abstract
Standard Languages

Both UML and SDL are standard languages:

- UML – Object Management Group (OMG)
- SDL – ITU (International Telecommunications Union)
UML Process

Object-oriented decomposition:

1. Use cases and use case diagram
2. Concept (class) diagram (ERD)
3. State diagram for each class (FSM) – often considered part of design
4. Sequence diagrams and collaboration diagrams (event traces)
SDL Process

Process-based decomposition:

1. System diagram (DFD) – shows top-level decomposition
2. Block diagrams (hierarchical DFDs)
3. Process diagrams (FSM for lowest level processes)
4. Message sequence charts to illustrate traces (event traces)
Here is the recommended process for your project:

1. Context Diagram
2. Use case descriptions and use case diagram
3. System sequence diagrams for each use case
4. User interface
5. Partition into major subsystems – show top-level decomposition and communication between these subsystems
SE 1 Process

1. For each subsystem, show in UML
   (a) System state diagram (or major subsystem)
   (b) Concept (class) diagram
   (c) For each class,
       • Attributes, Operations
       • Pre/post conditions for each operation,
       • Class-level state diagram for all non-trivial classes
   (d) Class-level sequence diagram showing how each
       system sequence diagram is realized through the
       behaviour of objects
Along the way collect glossary entries, requirements for traceability matrix.

Consistency among diagrams and levels of description is important!
Project

For first partial SRS:

- Purpose, etc.
- Use case descriptions, use case diagram
- Context Diagram(s)
- Complete specification of call processing subsystem
- User interface specification
- Glossary, requirements tables (complete for call processing)

For final SRS:

- Fix any problems from past deliverables
- Complete specification of all parts of the system
Summary

- What aspects of the system do we need to describe?
- What are the features of a good specification notation?
- Basic notations:
  - Event traces
  - Entity-relationship diagrams (ERDs)
  - Data flow diagrams (DFDs)
  - Finite state machines (FSM)
- Decomposition strategies
- RE Processes

Next Lecture Topic: UML state diagrams (continued over several lectures)

Reading: Arlow and Neustadt, Ch.1, 2, 12, 13.2, 21, 22