Lecture 7: Requirements Modeling III

Last Week:
Modeling and Analysis (II)
Modeling Functionality
Structured Analysis
Object Oriented Analysis

This Week:
Modeling and Analysis (III)
Formal Modeling Techniques
Program Specification vs. Reqs Modeling
Egs: RSML, SCR, RML, Telos, Albert II
Tips on formal modeling

Next Week:
Communicating Reqs
Specification Languages
Documentation Standards
Traceability

Formal Methods in RE

- What to formalize in RE?
  - models of requirements knowledge (so we can reason about them)
  - specifications of requirements (so we can document them precisely)

Why formalize in RE?
- To remove ambiguity and improve precision
- Provides a basis for verification that the requirements have been met
- Allows us to reason about the requirements
  - Properties of formal requirements models can be checked automatically
  - Can test for consistency, explore the consequences, etc.
- Allows us to animate/execute the requirements
  - Helps with visualization and validation
- Will have to formalize eventually anyway
  - RE is all about bridging from the informal world to a formal machine domain

Why people don’t formalize in RE?
- Formal Methods tend to be lower level than other analysis techniques
  - They force you to include too much detail
- Formal Methods tend to concentrate on consistent, correct models
  - …but most of the time your models are inconsistent, incorrect, incomplete...
- People get confused about which tools are appropriate:
  - E.g. modeling program behaviour vs. modeling the requirements
  - Formal methods advocates get too attached to one tool
- Formal methods require more effort
  - …and the payoff is deferred
What are Formal Methods?

- Broad View (Leveson)
  - application of discrete mathematics to software engineering
  - ...involves modeling and analysis
  - ...with an underlying mathematically-precise notation

- Narrow View (Wing)
  - Use of a formal language
    - a set of strings over some well-defined alphabet, with rules for distinguishing which strings belong to the language
  - Formal reasoning about formulae in the language
    - E.g., formal proofs: use axioms and proof rules to demonstrate that some formula is in the language

- For requirements modeling...
  - A notation is formal if:
    - it comes with a formal set of rules which define its syntax and semantics.
    - the rules can be used to analyse expressions to determine if they are syntactically well-formed or to prove properties about them.

Varieties of formal analysis

- Consistency analysis and typechecking
  - "Is the formal model well-formed?"
    - [assuming that we only use modeling languages where "well-formedness" is a useful thing to check]

- Validation:
  - Animation of the model on small examples
  - Formal challenges:
    - "if the model is correct then the following property should hold..."
  - 'What if' questions:
    - reasoning about the consequences of particular requirements:
    - reasoning about the effect of possible changes
  - State exploration
    - E.g., use a model checking to find traces that satisfy some property
  - Checking application properties:
    - "will the system ever do the following..."

- Verifying design refinement
  - "does the design meet the requirements?"
Three different models??

R: a model of the requirements

is satisfied by

acts upon

D: a model of the environment

acts upon

S: a model of the software behaviour

FM in practice

From Shuttle Study [Crow & DiVito 1996]

- More errors found in the process of formalizing the requirements than were found in the formal analysis
  - Formalization forces you to be precise and explicit, hence reveals problems
  - Formal analysis then finds fewer, but more subtle problems
- Typical errors found include:
  - inconsistent interfaces
  - incorrect requirements (system does the wrong thing in response to an input)
  - clarity/maintainability problems

<table>
<thead>
<tr>
<th>Issue Severity</th>
<th>With FM</th>
<th>Existing</th>
</tr>
</thead>
<tbody>
<tr>
<td>High Major</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>Low Major</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>High Minor</td>
<td>17</td>
<td>3</td>
</tr>
<tr>
<td>Low Minor</td>
<td>6</td>
<td>0</td>
</tr>
<tr>
<td>Totals</td>
<td>30</td>
<td>4</td>
</tr>
</tbody>
</table>
How do FMs differ?

**Mathematical Foundation**
- **Logic**
  - first order predicate logic — e.g. RML
  - temporal logic — e.g. Albert II, SCR, KAOS
  - multi-valued logic — e.g. Xchek
- **Other**
  - algebraic languages — e.g. Larch
  - set theory — e.g. Z

**Ontology**
- **fixed**
  - states, events, actions — e.g. SCR
  - entities, activities, assertions — e.g. RML
- **extensible**
  - meta language for defining new concepts — e.g. Telos

**Treatment of Time**
- **State/event models**
  - time as a discrete sequence of events — e.g. SCR
  - time as quantified intervals — e.g. KAOS
- **Time as a first class object**
  - meta-level class to represent time — e.g. Telos

Three traditions ...

**Formal Specification Languages**
- Grew out of work on program verification
- Spawned many general purpose specification languages
  - Suitable for specifying the behaviour of program units
  - Key technologies: Type checking, Theorem proving

**Reactive System Modeling**
- Grew out of a need to capture dynamic models of system behaviour
- Focus is on reactive systems (e.g. real-time, embedded control systems)
  - support reasoning about safety, liveness, performance?
  - provide a precise requirements specification language
- Key technologies: Consistency checking, Model checking

**Formal Conceptual Modeling**
- Grew out of a concern for capturing real-world knowledge in RE
- Focus is on modeling domain entities, activities, agents, assertions
  - provide a formal ontology for domain modeling
- Key technologies: inference engines, default reasoning, KBS-shells

Applicability to RE is poor
  - No abstraction or structuring
  - closely tied to program semantics

Examples: Larch, Z, VDM, ...

Applicability to RE is good
  - modeling languages were developed specifically for RE
  - Examples: Statecharts, RSML, Parnas-tables, SCR, ...

Applicability to RE is excellent
  - modeling schemes capture key requirements concepts
  - Examples: Reqts Apprentice, RML, Telos, Albert II, ...

© 2000-2003, Steve Easterbrook
(1) Formal Specification Languages

- Three basic flavours:
  - **Operational** - specification is executable abstraction of the implementation
    - good for rapid prototyping
    - e.g., Lisp, Prolog, Smalltalk
  - **State-based** - views a program as a (large) data structures whose state can be altered by procedure calls...
    - ... using pre/post-conditions to specify the effect of procedures
    - e.g., VDM, Z
  - **Algebraic** - views a program as a set of abstract data structures with a set of operations...
    - ... operations are defined declaratively by giving a set of axioms
    - e.g., Larch, CLEAR, OBJ

- Developed for specifying **programs**
  - Programs are formal, man-made objects
  - ... and can be modeled precisely in terms of input-output behaviour
  - But in RE we're more concerned with:
    - real-world concepts, stakeholders, goals, loosely define problems, environments
  - So these languages are NOT appropriate for RE
    - but people fail to realise that requirements specification ≠ program specification

(2) Reactive System Modeling

- modeling how a system should behave
  - General approach:
    - Model the environment as a state machine
    - Model the system as a state machine
    - Model safety, liveness properties of the machine as temporal logic assertions
    - Check whether the properties hold of the system interacting with its environment

- Examples:
  - **Statecharts**
    - Harel’s notation for modeling large systems
    - Adds parallelism, decomposition and conditional transitions to STDs
  - **RSML**
    - Heimdahl & Leveson’s Requirements State Machine Language
    - Adds tabular specification of complex conditions to Statecharts
  - **A7e approach**
    - Major project led by Parnas to formalize A7e aircraft requirements spec
    - Uses tables to specify transition relations & outputs
  - **SCR**
    - Heitmeyer et. al. “Software Cost Reduction”
    - Extends the A7e approach to include dictionaries & support tables
(3) Formal Conceptual **Modeling**

**General approach**
- model the world beyond functional specifications
  - a specification is prescriptive, concentrating on desired properties of the machine
  - but we also need to capture an understanding of the application domain
  - hence build models of humans’ knowledge/beliefs about the world
- make use of abstraction & refinement as structuring primitives

**Examples:**
- **RML - Requirements Modeling Language**
  - Developed by Greenspan & Mylopoulos in mid-1980s
  - First major attempt to use knowledge representation techniques in RE
  - Essentially an object oriented language, with classes for activities, entities and assertions
  - Uses First Order Predicate Language as an underlyng reasoning engine
- **Telos**
  - Extends RML by creating a fully extensible ontology
  - meta-level classes define the ontology (the basic set is built in)
- **Albert II**
  - developed by Dubois & du Bois in the mid-1990s
  - Models a set of interacting agents that perform actions that change their state
  - uses an object-oriented real-time temporal logic for reasoning

---

**Example: State Transition Diagram**

```
idle
  ↓
don hook / quiet

busy
  ↓
Dial busy number / busy tone

on hook / quiet
  ↑
Dial number / dial tone

ringing
  ↓
Called party on hook / dial tone

on hook / quiet
  ↑
Called party off hook / connected

state transition stimulus response
```

---

© 2000-2003, Steve Easterbrook
Example: Statecharts

- State: idle, Dial tone, Busy tone, connected
- Transition: Lift receiver, Replace receiver, Dial (callee idle)
- Guard: Callee replaces receiver
- Stimulus: Callee lifts receiver
- Default entry state: Connected

Example: SCR

- Four Variable Model:
- System: Monitored Variables, input devices, input data items, output devices, output data items
- Software: Mode Transition Tables
- Environment: Monitored/Controlled Variables, Types, Constants, Event Tables, Condition Tables
- Dictionaries:
- Tables:
- also: Assertions, Scenarios, ...
SCR basics

- **Modes and Mode classes**
  - A mode class is a finite state machine, with states called system modes.
  - Transitions in each mode class are triggered by events.
  - Complex systems are described using a number of mode classes operating in parallel.

- **System State**
  - A (system) state is defined as:
    - the system is in exactly one mode from each mode class...
    - ...and each variable has a unique value.

- **Events**
  - An event occurs when any system entity changes value.
    - An input event occurs when an input variable changes value.
    - Single input assumption - only one input event can occur at once.
    - Notation: \( \ominus T(c) \) means “c changed from false to true.”
  - A conditioned event is an event with a predicate.
    - \( \ominus T(c) \text{ WHEN } d \) means: “c became true when c was false and d was true.”

SCR Tables

- **Mode Class Tables**
  - Define the set of modes (states) that the software can be in.
  - A complex system will have many different modes classes.
    - Each mode class has a mode table showing the conditions that cause transitions between modes.
    - A mode table defines a *partial function* from modes and events to modes.

- **Event Tables**
  - An event table defines how a term or controlled variable changes in response to input events.
  - Defines a *partial function* from modes and events to variable values.

- **Condition Tables**
  - A condition table defines the value of a term or controlled variable under every possible condition.
  - Defines a *total function* from modes and conditions to variable values.

Source: Adapted from Heitmeyer et al. 1996.
### Example: Temp Control System

#### Mode transition table:

<table>
<thead>
<tr>
<th>Current Mode</th>
<th>Powered on</th>
<th>Too Cold</th>
<th>Temp OK</th>
<th>Too Hot</th>
<th>New Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>Off</td>
<td>@T</td>
<td>-</td>
<td>t</td>
<td>-</td>
<td>Inactive</td>
</tr>
<tr>
<td></td>
<td>@T</td>
<td>t</td>
<td>-</td>
<td>-</td>
<td>Heat</td>
</tr>
<tr>
<td></td>
<td>@T</td>
<td>-</td>
<td>-</td>
<td>t</td>
<td>AC</td>
</tr>
<tr>
<td>Inactive</td>
<td>@F</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Off</td>
</tr>
<tr>
<td></td>
<td>-</td>
<td>@T</td>
<td>-</td>
<td>-</td>
<td>Heat</td>
</tr>
<tr>
<td></td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>@T</td>
<td>AC</td>
</tr>
<tr>
<td>Heat</td>
<td>@F</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Off</td>
</tr>
<tr>
<td></td>
<td>-</td>
<td>-</td>
<td>@T</td>
<td>-</td>
<td>Inactive</td>
</tr>
<tr>
<td>AC</td>
<td>@F</td>
<td>-</td>
<td>-</td>
<td>@T</td>
<td>Off</td>
</tr>
<tr>
<td></td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>@T</td>
<td>Inactive</td>
</tr>
</tbody>
</table>

*Source: Adapted from Heitmeyer et al. 1996.*

### Failure modes

#### Mode transition table:

<table>
<thead>
<tr>
<th>Current Mode</th>
<th>Powered on</th>
<th>Cold Heater</th>
<th>Too Cold</th>
<th>Warm AC</th>
<th>Too Hot</th>
<th>New Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>NoFailure</td>
<td>t</td>
<td>@T</td>
<td>t</td>
<td>-</td>
<td>-</td>
<td>HeatFailure</td>
</tr>
<tr>
<td></td>
<td>t</td>
<td>-</td>
<td>-</td>
<td>@T</td>
<td>t</td>
<td>ACFailure</td>
</tr>
<tr>
<td></td>
<td>t</td>
<td>-</td>
<td>@F</td>
<td>t</td>
<td>t</td>
<td>NoFailure</td>
</tr>
</tbody>
</table>

### Event table:

<table>
<thead>
<tr>
<th>Modes</th>
<th>@T(INMODE)</th>
<th>never</th>
</tr>
</thead>
<tbody>
<tr>
<td>NoFailure</td>
<td></td>
<td>never</td>
</tr>
<tr>
<td>ACFailure, HeatFailure</td>
<td>@F</td>
<td>t</td>
</tr>
</tbody>
</table>

*Warning light = Off, On*

*Source: Adapted from Heitmeyer et al. 1996.*
Consistency Checks in SCR

- Syntax
  - did we use the notation correctly?

- Type Checks
  - do we use each variable correctly?

- Disjointness
  - is there any overlap between rows of the mode tables?
    - ensures we have a deterministic state machine

- Coverage
  - does each condition table define a value for all possible conditions?

- Mode Reachability
  - is there any mode that cannot ever happen?

- Cycle Detection
  - have we defined any variable in terms of itself?

Using Formal Methods

- Selective use of Formal Methods
  - Amount of formality can vary
  - Need not build complete formal models
    - Apply to the most critical pieces
    - Apply where existing analysis techniques are weak
  - Need not formally analyze every system property
    - E.g. check safety properties only
  - Need not apply FM in every phase of development
    - E.g. use for modeling requirements, but don't formalize the system design
  - Can choose what level of abstraction (amount of detail) to model

- Lightweight Formal Methods
  - Have become popular as a means of getting the technology transferred
  - Two approaches
    - Lightweight use of FMs - selectively apply FMs for partial modeling
    - Lightweight FMs - new methods that allow unevaluated predicates