From Natural Language Requirements to Rigorous Property Specifications

Lori A. Clarke

Work done in collaboration with
Rachel L. Smith, George S. Avrunin

University of Massachusetts Amherst

The World We Live In

- Software developers write requirements in
  - Natural language
  - UML state diagrams and other mostly semantic-free notations
  - Or not at all
- Not precise enough to be used as the basis for
  - Consistency checking
  - Design and implementation
  - Test planning and verification
- Requirements and the system diverge!

Seat Control Properties

- A request for horizontal movement of the seat base gets priority over the front tilt motor if activated at the same time
- When the front tilt is requested to move upward and then the rear tilt motor is requested to move downward, only the front tilt motor will move
- When the horizontal base is requested to move forward and then requested to move backward, there is a minimum 50ms pause between changes in direction
- The rear tilt switch has priority over a recall message

Seat Control Property 1

- A request for horizontal movement of the seat base gets priority over the front tilt motor if activated at the same time
  - Need domain experts to clarify
    - "gets priority"
    - "if activated at the same time"
      start of activation at the same instant, or
      overlap of intervals where both are on?

Absence of (front_tilt_move) Between (horiz_req_on AND front_tilt_on) and horiz_req_off

Property Specifications Need to Be…

- Accessible
  - so that we understand what they're saying
- Precise
  - so that we can tell unambiguously whether a particular behavior satisfies or violates the property
- The problem is that…

  these goals usually conflict

Accessible

- Natural language is accessible (and most requirements are specified in it)
  - When the call button is pushed at a floor, the elevator cannot come to the floor more than once without opening its doors.
- But this is not precise or rigorous enough
  - What if the button is pushed repeatedly?
  - Does elevator have to come to the floor at all?
    - …

Precise Version of Example

- Have many formal notations for expressing properties precisely, e.g., Linear Temporal Logic

Precise Version of Example

- Have many formal notations for expressing properties precisely, e.g., Linear Temporal Logic

$$\Box((\text{call} \wedge \Diamond \text{open}) \rightarrow ((\neg\text{atfloor} \wedge \neg\text{open}) \,\mathcal{U}\, (\text{open} \vee ((\text{atfloor} \wedge \neg\text{open}) \,\mathcal{U}\, (\text{open} \vee ((\neg\text{atfloor} \wedge \neg\text{open}) \,\mathcal{U}\, (\text{open} \vee ((\text{atfloor} \wedge \neg\text{open}) \,\mathcal{U}\, (\text{open} \vee (\neg\text{atfloor} \,\mathcal{U}\, \text{open})))))))))$$

Precise Version of Example

- Have many formal notations for expressing properties precisely, e.g., Linear Temporal Logic

$$\Box((\text{call} \wedge \Diamond \text{open}) \rightarrow ((\neg\text{atfloor} \wedge \neg\text{open}) \,\mathcal{U}\, (\text{open} \vee ((\text{atfloor} \wedge \neg\text{open}) \,\mathcal{U}\, (\text{open} \vee ((\neg\text{atfloor} \wedge \neg\text{open}) \,\mathcal{U}\, (\text{open} \vee ((\text{atfloor} \wedge \neg\text{open}) \,\mathcal{U}\, (\text{open} \vee (\neg\text{atfloor} \,\mathcal{U}\, \text{open})))))))))$$

- But this is not really accessible (even for experts!)

Requirements for Requirements

- Provide a sound basis for design and implementation
  - Accessible and Precise
- Amenable to consistency and other analyzes
  - E.g. View all requirements that deal with the tilt operation
  - Check that these are consistent with each other
- Provide a sound basis for testing and verification

Must provide enough value to make the investment worthwhile!

Our Approach

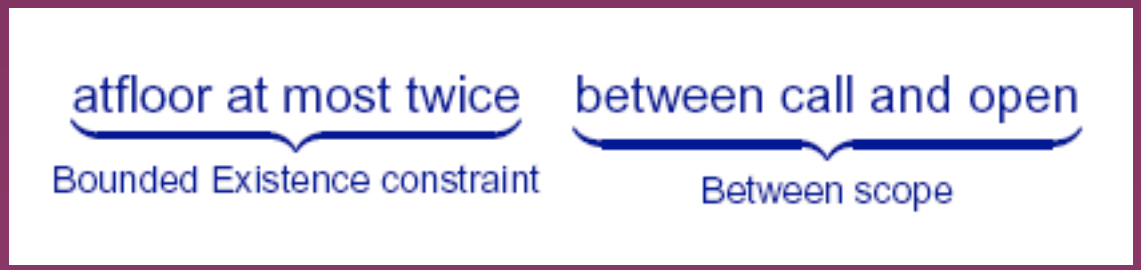
- Provide NL templates
  - Based on commonly occurring patterns
  - Expose the options that must be considered
  - Acceptable to developers
- Map to a precise formal notation
  - Basis for consistency analysis and verification
- Multiple views
  - Providing both should help and reassure developers

Build upon Specification Patterns

- Specification patterns [Dwyer, Avrunin, Corbett, 1999] intended as high-level abstractions
  - Generalized description of commonly-occurring requirements
  - Parameterizable
  - Formalism-independent
- Modeled on Design Patterns
  - Leverage experience of system developers by capturing description of good solutions to recurring design problem

Scopes and Constraints

- Each pattern has a constraint and a scope
  - Constraint gives requirement for behavior of system in that scope
  - Scope gives the extent of execution over which the constraint must hold



From Patterns to Formal Notations

Constraints

Absence	Never
Existence	Eventually
Universality	Always
Bounded Existence	Eventually n times
⋮	
Response	*action* results in *response*
Precedence	*action* enables *response*
Chain Response	*action* results in $resp_1$ $resp_2$ . . .
⋮	

Scopes

Global	Full execution
After *Q*	After first *Q*
Before *R*	Before first *R*
Between *Q* and *R*	Between any *Q* and next *R*
After *Q* until *R*	Between any *Q* and next *R*, if any

- Pattern system gives mappings from constraint-scope combinations to several formal notations (e.g., regular expressions, various temporal logics, etc.)

But Subtle Details are Critical

- Consider the following property:
  After the close-door button is pushed,
  the elevator doors are closed
- Response constraint with Global scope, but there are many questions about precise intent:
  - If button pushed repeatedly, should doors close repeatedly?
  - What, if anything, can occur between pushing the button and the doors closing?
  - Can the doors close without the button being pushed?
  - Does the button have to be pushed?

Property Specification Frameworks Need to Support

- Accessiblity
  - » Easily understandable by specifiers and users
- Precision
  - » Unambiguous, suitable for use with testing and verification tools
- Elucidation
  - » Specifier needs to have carefully addressed questions about details

PROPEL

- Extend the specification patterns:
  - » Represent pattern by template that explicitly shows options
    - – Help specifier consider relevant subtleties and alternatives
- Represent templates using two notations
  - » One is accessible (Disciplined Natural Language)
  - » One is mathematically precise (automata)
  - » Template representations are linked-- specifier can work with both simultaneously
- Decision tree for selecting the right pattern
  - » Constraint and scope

demo

What Next?

- Need to complete initial prototype of tool
- Several directions for further development:
  - Explore solution space
  - Investigate other ways of organizing properties and options
  - Support other precise formalisms
  - Explore integration with various testing/verification tools
  - Evaluation

Explore Solution Space

- Consider options for scopes
- Reexamine interaction between scopes and constraints
- Use decision tree for all options
  - Don’t bother with patterns at all
- Improve NL representation

Other Ways of Organizing Properties/Options

- Parameterization and Composition
  - Replace parameters in patterns by more complicated expressions
  - Certain replacements are fine but other are likely to be incorrect—not well understood in general
  - Composition of properties: chain patterns are simple versions

Support Other Precise Formalisms

- Other event-based formalisms
- State-based formalisms (e.g., the temporal logics)
  - Describe execution in terms of which propositions are true at a particular time
  - Some properties are more naturally expressed in state-based formalism
    - While mode is level_flight, landing gear switch is always disabled.
  - Options may be different for state-based formalisms
- Formalisms dealing with both states and events
  - Some properties naturally involve both states and events
    - While use_count is less than 10, registering a request always leads to resource_acquisition

Organizing Properties/Options

- Libraries of properties for a single system
  - Check for consistency, completeness
  - Refinement of property statements as systems passes through stages of development
  - Evolution of properties as system evolves

Integration with Testing/Verification Tools

- Integrate with other tools
  - » Make (correct) specification easier for users
  - » Interpret feedback from tool (e.g., execution violating property) directly in terms of property specification

Evaluation

- Want to evaluate if PROPEL approach is useful and discover ways to improve it
- Controlled experiments extremely expensive and difficult
- Looking for suitable case studies

Concluding Questions

- Can we help bridge the gap between informal intent and precise specification?
  - Elucidation: Encourage specifiers to think about and resolve the issues
- Can we provide a NL framework that is "comforting" to developers?
  - » NL improves high-level understanding
  - » Increases acceptance

Concluding Questions

- Can we provide a NL framework that is both “comforting” to developers, and rigorous enough to support analysis?
- How should we evaluate success
  - » Will specifiers choose to use this approach?
  - » Will specifiers update their requirements with this approach?
  - » Will this approach be used to support upstream activities?

Questions?