From System Specifications to Component Behavioral Models

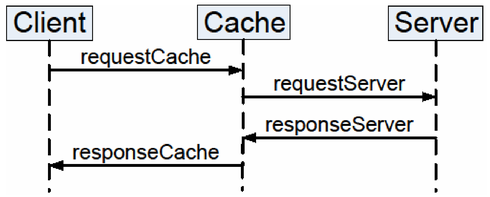
Ivo Krka, George Edwards, Yuriy Brun, and Nenad Medvidovic
University of Southern California

Problem

Early System Specifications

- Early development activities (e.g., requirements elicitation and refinement) occur in the context of incomplete information and uncertainty
- Resulting specifications are partial, and capture high-level behavior and characteristics of the system (e.g., UML sequence diagrams and OCL constraints)

Web Cache Specification



requestCache
pre: ¬requestPending
post: requestPending

responseCache
pre: cached and requestPending
post: ¬requestPending

requestServer
pre: ¬cached and requestPending

responseServer
pre: ¬cached
post: cached

dataUpdate
post: ¬cached

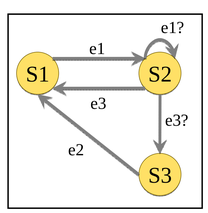
dataChanged
post: ¬cached

The Problem Statement

- Synthesizing more comprehensive behavioral models from early system specifications is beneficial
- Existing component-level synthesis approaches ignore the inherent partiality of the early specifications
- Existing approaches which account for partiality of specifications generate only system-level models

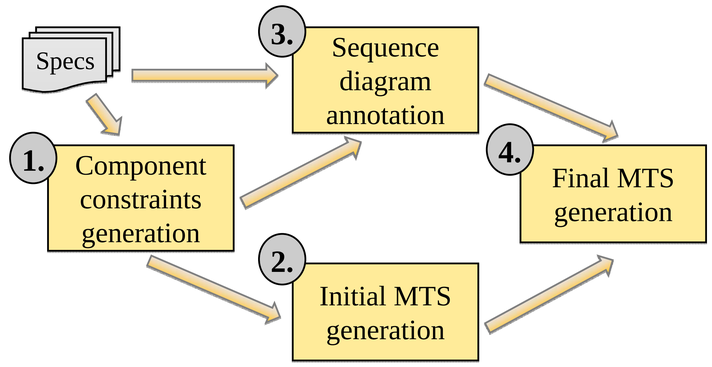
Modal Transition Systems

- A formalism for capturing both required and potential behavior in a single model
- Refinement of an MTS corresponds to notion of a “more specified” model



Approach

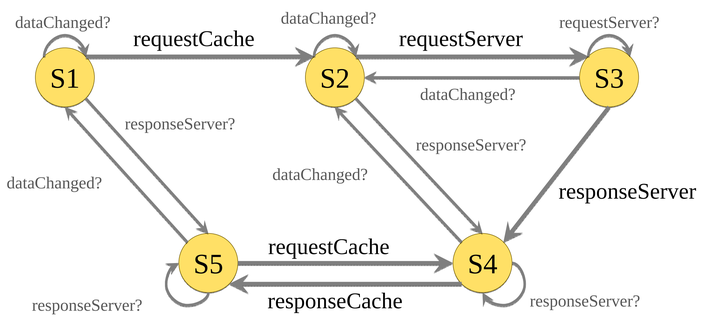
Component-Level MTS Synthesis



Algorithm Phases in a Nutshell

1. We translate the provided system-level constraints to component-level constraints which restrict the allowed behavior of each component
2. For each component, we generate an MTS which does not violate the component constraints and contains only potential transitions
3. We annotate the sequence diagrams with conditions that represent state variable values that must hold at particular points in scenario execution
4. We refine the initial component MTSs by incorporating the scenarios as required behavior

Cache Component’s Final MTS



Outcome

Utilizing Component MTSs

- Discovering specification discrepancies
  - *Conflicts* between scenarios and properties (e.g., a scenario cannot execute as specified)
  - *Inconsistent* component states (e.g., a component does not reach a desirable state)
- Requirements elicitation
  - Proposing *new scenarios* (e.g., requestServer→responseServer→ dataChanged → responseServer)
  - Proposing *new properties* (e.g., $\Box((\text{requestPending} \wedge \neg\text{cached}) \Rightarrow \bigcirc(\text{requestPending}))$)
- Off-the-shelf component selection
  - Final MTS explicitly captures behavior that is required by the specification and forbids behavior proscribed by the specification
  - By comparing specifications of OTS components with the final MTS, we can select the most appropriate components
- Comparison of as-intended and as-implemented
  - After implementation, the behavior of a component needs to be checked against the *key requirements* (i.e., how the system was intended to behave)
  - Comparison between the final MTS with the behavior extracted from the implementation can serve as a measure of *architectural drift*

Contributions and Conclusions

- Modal Transition Systems provide means to effectively capture the partial nature of early system specifications
- We designed an algorithm which compliments the existing behavioral model synthesis algorithms
- We enumerated ways in which component MTSs can support and enrich software development activities
- An in-depth description of this work as well as several new directions can be found in our ESEC/FSE 2009 paper: http://csse.usc.edu/~ybrun/pubs/Krka09fse.pdf