Behavioural Model Fusion: An Overview of Challenges

Shiva Nejati and Marsha Chechik
University of Toronto
Model-Based Development

Requirements models

Structural models

Runtime models

Behavioural models

Figure 2: Reason/COBOL Control Flow Graph.
Motivation

→ Large software projects
  ➡ Developers work on a set of interrelated models

➢ Different features

➢ Different perspectives

➢ Different variants across a product family
Motivation

➔ Large software projects

➔ Developers work on a set of interrelated models

➢ Different features

➢ Different perspectives

➢ Different variants across a product family

We refer to the process of combining a set of interrelated models as model fusion
Model Fusion Activities

→ **Merge**
  ➤ *building a global view of a set of overlapping models*
    ➢ e.g., overlapping perspectives on a single feature

→ **Composition**
  ➤ *assembling a set of interacting features*
    ➢ e.g., run sequentially or in parallel

→ **Weaving**
  ➤ *incorporating cross-cutting concerns into a base system*
    ➢ e.g., aspect-oriented languages
Model Fusion Activities

→ **Merge**
   - building a global view of a set of overlapping models
     - e.g., overlapping perspectives on a single feature

→ **Composition**
   - assembling a set of interacting features
     - e.g., run sequentially or in parallel

→ **Weaving**
   - incorporating cross-cutting concerns into a base system
     - e.g., aspect-oriented languages
Topics in This Talk

Two fusion problems for behavioural models

1. Merging variant feature specifications
2. Composing Features and Analyzing Interactions
Overview

Models
Overview

Specifications of Relationships

Models
Overview

Models

Specification of Relationships

Fusing Models


c
(s3, t2)
(s4, t4)
(s5, t5)
(s6, t6)
(s7, t7)
t8

Fusing Models
Overview

Models

Specification of Relationships

Fusing Models

Results

Model Analysis
Overview

How to choose an appropriate formalism?

Specification of Relationships

Fusing Models

Results

Model Analysis

Models

Fusing Models

How to choose an appropriate formalism?
Overview

How to specify relationships between models?

How to choose an appropriate formalism?

Fusing Models

Results

Model Analysis

How to specify relationships between models?

Model Analysis

How to choose an appropriate formalism?

Fusing Models

Results

Model Analysis

How to specify relationships between models?

Model Analysis

How to choose an appropriate formalism?

Fusing Models

Results
Overview

How to specify relationships between models?

How to choose an appropriate formalism?

What is the semantics of fusion and what is its goal?

Results

Model Analysis
Overview

How to specify relationships between models?

How to choose an appropriate formalism?

What is the semantics of fusion and what is its goal?

What kind of analysis is needed to ensure the correctness of the process?
Topics in This Talk

→ Two fusion problems for behavioural models

- Merging variant feature specifications
- Composing Features and Analyzing Interactions
Topics in This Talk

Two fusion problems for behavioural models

- Merging variant feature specifications

Models: State Machines with overlapping behaviours
- Goal: Identifying and unifying states with similar behaviours while preserving their variabilities
Feature Variants

→ AT&T Call Logger feature: Logs call information and makes it available to subscribers through a web portal
→ AT&T Call Logger feature: Logs call information and makes it available to subscribers through a web portal

**Feature Variants**

**AT&T Call Logger feature**

- Logs call information
- Available to subscribers through a web portal

**Diagram**

- **Call logger - basic**
  - **Start**
  - **Link Call Cee**
  - **Wait**
  - **Timer Started**
  - **Log Success**
  - **Log Failure**
  - **Initialize Links**
    - setup [zone=source]/callee = participant
    - setup [zone=target]/callee = subscriber

- **Call logger - voicemail**
  - **Start**
  - **Link Subscriber**
  - **Pending**
  - **Log Success**
  - **Log Failure**
  - **Redirect To Voicemail**
    - subscriber?Ack

**Alternative Descriptions of an individual feature**
One Relationship:
\{(Wait, Pending),
(Success, Success),
(Failure, Failure)\}
One Relationship:
{(Wait, Pending),
(Success, Success),
(Failure, Failure)}
After a connection is set up, a successful call will be logged if the subscriber or participant sends Accept.
After a connection is set up, a successful call will be logged if the subscriber or participant sends Accept.
After a connection is set up, a successful call will be logged if the subscriber or participant sends Accept.
After a connection is set up, a voicemail will be logged if the call is redirected to a voicemail service.
After a connection is set up, a voicemail will be logged if the call is redirected to a voicemail service.
After a connection is set up, a voicemail will be logged if the call is redirected to a voicemail service.
Recap (Merging)

Specification of Relationships

Models

Fusing Model

Model Analysis
Recap (Merging)

Specification of Relationships

Variant models

Fusing Model

Model Analysis
Recap (Merging)

Heuristic matching techniques

Variant models

Fusing Model

Model Analysis
Recap (Merging)

Heuristic matching techniques

Variant models

Computing a common refinement

Model Analysis
Recap (Merging)

Heuristic matching techniques

Variant models

Computing a common refinement

Model checking parameterized state machines
Some Challenges in Merging

→ How models are related?
  ➤ identification, representation, and verification of relationships

→ Meta-model level merges for behavioural models
  ➤ need to augment meta-models with semantic aspects
    ➤ may also help merging heterogeneous behavioural notations

→ Industrial model merging tools
  ➤ often don’t make relationships explicit
    ➤ IBM Rational Software Architect and IBM Rational Rose
  ➤ model-level relationships vs. view-level relationships
Topics in This Talk

→ Two fusion problems for *behavioural* models

- Merging variant feature specifications
- Composing Features and Analyzing Interactions
Topics in This Talk

- Two fusion problems for behavioural models
  - Models: State Machines with interacting behaviours
  - Goal: Constructing a composition in which undesirable behaviours are absent

Composing Features and Analyzing Interactions
System Features

Call Blocking (CB)

- **States:**
  - $s_0$: initial
  - $s_1$: checking
  - $s_2$: idle
  - $s_3$: blocked

- **Transitions:**
  - $s_0 \xrightarrow{cb.setup?} s_1$
  - $s_1 \xrightarrow{cb.reject; / cb.unavail!} s_3$
  - $s_1 \xrightarrow{cb.accept; / cb.setup!} s_2$

Call Blocking blocks calling requests coming from addresses on a blocked list.

Record Voice Mail (RVM)

- **States:**
  - $t_0$: initial
  - $t_1$: idle
  - $t_2$: dialogue

- **Transitions:**
  - $t_0 \xrightarrow{rvm.setup?/ rvm.setup!} t_1$
  - $t_1 \xrightarrow{rvm.unavail? / rvm.unavail!} t_3$
  - $t_1 \xrightarrow{rvm.voicemail; rvm.unavail!} t_2$

Record Voice Mail records a voice mail message when the callee is not available.
System Features

Call Blocking (CB)

- **initial**
- **checking**
- **blocked**
- **idle**

- From **initial** to **checking**: cb.setup?
- From **checking** to **blocked**: cb.reject; / cb.unavail!
- From **checking** to **idle**: cb.accept; / cb.setup!
- From **blocked** to **idle**: s

Call Blocking blocks calling requests coming from addresses on a blocked list.

Record Voice Mail (RVM)

- **initial**
- **idle**
- **dialogue**

- From **initial** to **idle**: rvm.setup?/ rvm.setup!
- From **idle** to **dialogue**: rvm.unavail? / rvm.voicemail; rvm.unavail!

Record Voice Mail records a voice mail message when the callee is not available.

Descriptions of different features of a system.
Feature Interaction

Call Blocking (CB)

- **Initial State**: $s_0$
  - **States**: initial, checking, blocked, idle
  - **Transitions**:
    - $cb.setup? / cb.setup!$
    - $cb.reject; / cb.unavail!$

Record Voice Mail (RVM)

- **Initial State**: $t_0$
  - **States**: initial, idle, dialogue
  - **Transitions**:
    - $rvm.setup? / rvm.setup!$
    - $rvm.unavail? / rvm.voicemail; rvm.unavail!$
Feature Interaction

Call Blocking (CB)

- **S₀**: initial
  - cb.setup?
    - cb.reject; /
    - cb.unavail!
  - cb.accept; /
  - cb.setup!

- **S₁**: checking
- **S₂**: idle
- **S₃**: blocked

Record Voice Mail (RVM)

- **T₀**: initial
  - rvm.setup?/ rvm.setup!
- **T₁**: idle
  - rvm.unavail? /
    - rvm.voicemail; rvm.unavail!
- **T₂**: dialogue
Feature Interaction

Call Blocking (CB)

$S_0$:
- initial
- cb.setup?
- cb.reject; /
- cb.unavail!

$S_1$:
- checking
- cb.accept; /
- cb.setup!

$S_2$:
- idle

$S_3$:
- blocked

unavail

Record Voice Mail (RVM)

$t_0$:
- initial
- rvm.setup? /
- rvm.setup!

$t_1$:
- idle
- rvm.unavail? /
- rvm.voicemail;
- rvm.unavail!

$t_2$:
- dialogue
**Feature Interaction**

**Call Blocking (CB)**

- **$s_0$**: Initial state
  - Transition to $s_1$: cb.setup?
  - Transition to $s_2$: cb.reject; / cb.unavail!
  - Transition to $s_3$: cb.accept; / cb.setup!

- **$s_1$**: Checking state
- **$s_2$**: Idle state
- **$s_3$**: Blocked state

**Record Voice Mail (RVM)**

- **$t_0$**: Initial state
  - Transition to $t_1$: rvm.setup?/ rvm.setup!
  - Transition to $t_2$: rvm.unavail? /
    rvm.voicemail; rvm.unavail!
Feature Interaction

**Call Blocking (CB)**

- Initial state: $s_0$
- Checking state: $s_1$
- Blocked state: $s_3$
- Idle state: $s_2$

Transition:
- $s_0$ to $s_1$: `cb.setup?`
- $s_1$ to $s_3$: `cb.reject; cb.unavail!`
- $s_3$ to $s_2$: `cb.accept; cb.setup!`

**Record Voice Mail (RVM)**

- Initial state: $t_0$
- Idle state: $t_1$
- Dialogue state: $t_2$

Transition:
- $t_0$ to $t_1$: `rvm.setup?/ rvm.setup!`
- $t_1$ to $t_2$: `rvm.unavail? rvm.voicemail; rvm.unavail!`
Feature Interaction

Call Blocking (CB)

- **$s_0$**: initial
- **$s_1$**: checking
- **$s_2$**: idle
- **$s_3$**: blocked

- cb.setup?
- cb.reject; /
- cb.unavail!
- cb.accept; /
- cb.setup!

Record Voice Mail (RVM)

- **$t_0$**: initial
- **$t_1$**: idle
- **$t_2$**: dialogue

- rvm.setup? /
- rvm.setup!
- rvm.unavail?
- rvm.unavail!
- rvm.voicemail!

unavails
Feature Interaction

Call Blocking (CB)

- **$s_0$**: initial
- **$s_1$**: checking
- **$s_2$**: idle
- **$s_3$**: blocked

- **cb.setup?** → checking
- **cb.reject; cb.unavail!** → blocked
- **cb.accept; cb.setup!** → idle

Record Voice Mail (RVM)

- **$t_0$**: initial
- **$t_1$**: idle
- **$t_2$**: dialogue

- **rvm.setup?/rvm.setup!** → idle
- **rvm.unavail?/rvm.unavail!** → dialogue
- **rvm.voicemail** → dialogue

RVM records a message from a blocked caller!
Feature-Based Systems

→ Feature interactions are often resolved by
  ➤ finding a feature ordering in a linear architecture
Feature-Based Systems

Feature interactions are often resolved by finding a feature ordering in a linear architecture.

---

**Call Blocking (CB)**

\[
\begin{array}{c}
S_0 \\
\text{initial}\\
\rightarrow s_1 \\
\text{checking} \\
\rightarrow s_2 \\
\text{idle}\\
\rightarrow s_3 \\
\text{blocked}\\
\end{array}
\]

**Record Voice Mail (RVM)**

\[
\begin{array}{c}
t_0 \\
\text{initial}\\
\rightarrow t_1 \\
\text{idle}\\
\rightarrow t_2 \\
\text{dialogue}\\
\end{array}
\]
Feature-Based Systems

→ Feature interactions are often resolved by
  ↪ finding a feature ordering in a linear architecture

Call Blocking (CB)

Record Voice Mail (RVM)

Record Voice Mail (RVM)

Call Blocking (CB)
Feature-based systems

Feature interactions are often resolved by finding a feature ordering in a linear architecture.
Feature-Based Systems

Feature interactions are often resolved by finding a feature ordering in a linear architecture.

Call Blocking (CB)

Check initial
Blocked
Idle
cb.setup?
cb.reject; /
cb.unavail!
cb.accept; /
cb.setup!

Record Voice Mail (RVM)

Check initial
Idle
RVM.setup?/
RVM.setup!
RVM.unavail? /
RVM.voicemail;
RVM.unavail!

Record Voice Mail (RVM)

Check initial
Idle
RVM.setup?/
RVM.setup!
RVM.unavail? /
RVM.voicemail;
RVM.unavail!

Call Blocking (CB)

Check initial
Blocked
Idle
cb.setup?
cb.reject; /
cb.unavail!
cb.accept; /
cb.setup!

Record Voice Mail (RVM)

Check initial
Idle
RVM.setup?/
RVM.setup!
RVM.unavail? /
RVM.voicemail;
RVM.unavail!

but, the features may evolve, and hence,

⇒ feature orderings may have to be computed from scratch after each change
but, the features may evolve, and hence,

feature orderings may have to be computed from scratch after each change

This is very expensive!
but, the features may evolve, and hence,

feature orderings may have to be computed from scratch after each change

This is very expensive!

Exploiting domain-specific patterns used in feature design

these patterns introduce similarity to the behaviour of features

allowing us to lift local properties to global system-level properties
Recap (Interaction Analysis)

Specification of Relationships

Models

Model Fusion

Model Analysis
Recap (Interaction Analysis)

Specification of Relationships

interacting models describing different features

Model Fusion

Model Analysis
Recap (Interaction Analysis)

Synthesizing feature configurations

interacting models describing different features

Model Fusion

Model Analysis
Recap (Interaction Analysis)

Interacting models describing different features

Synthesizing feature configurations

Parallel composition

Model Analysis
Recap (Interaction Analysis)

Synthesizing feature configurations

interacting models describing different features

Model checking compositions for absence of undesirable interactions

Parallel composition
Some Challenges in Composition

- Techniques for verifying concurrent evolving systems
  - identifying design guidelines and patterns that make system components independent

- Formalisms for capturing interacting (open) systems
  - input/output automata, alternating (agent-based) transition systems, interface automata, etc

- Different notions of feature composition
  - sequential, parallel, quasi-sequential
Conclusion

- Studied two model fusion problems for behavioural models
  - merging models with behavioural overlap
  - composing models and analyzing their interactions

- Even though all fusion activities involve models that are somehow related
  - problems that arise in each case are different in details
Acknowledgements: Mehrdad Sabetzadeh, Steve Easterbrook, Sebastian Uchitel, Pamela Zave, and Rick Salay.

Thank You!

Questions?