Formal Support for Merging and Negotiation

Shiva Nejati

Department of Computer Science
University of Toronto, Canada.

Email: shiva@cs.toronto.edu
Research Goal

Merging behavioural software models and resolving the inconsistencies
Motivation

What is merging?
- Putting models together while preserving certain properties of the given models

Why merge?
- To enable various kinds of analysis
- To obtain a global picture of the whole system
- To facilitate inconsistency/conflict detection and resolution
Strategies for Merging

Three-way Merge (see [Mens’ 02])

Making use of information in the common ancestor
Two-way Merge (see [Mens’ 02])

- Merging two versions of a model without relying on their common ancestor
Artifacts to Merge

- Textual merging
  - flexible but primitive, e.g., CVS

- Syntactic merging
  - Preserves structures but not semantics, e.g., [Mens ’00], [Sabetzadeh-Easterbrook ’03, ’05]
Artifacts to Merge

- Semantic merging

  - Preserves program outputs, e.g., [Yang-Horwitz-Reps ’92]

- Logical merging (composition)
  - Composition is a conjunction of logical formulas, e.g. [Zave-Jackson ’93]
Artifacts to Merge

**Behavioural merging**

- Preserves behaviours (and hence, temporal properties), e.g., [Huth '04], [Uchitel-Chechik '04]
A Practical Example

Telephony examples from Pamela Zave

- Different versions of telephony features
  - ... to accommodate different sets of end-user goals
- This can result in duplicates

Desiderata

- Merging different versions of telephony features
  - ... to produce a merged model that contains conditional behaviours capturing all possible scenarios
Call Logger Feature

Call Logger: logs call information so that it is available to the subscriber through a web portal.

vplus:

acsvoip:
Call Logger Feature

Call Logger: logs call information so that it is available to the subscriber through a web portal.

vplus:

acsvoip:
The Common Ancestor

Need to identify the commonalities
  ➜ . . . by unifying the vocabulary of the two models

vplus:

acsvoip:

Commonalities:
The Merge Model

**vplus:**

```
<table>
<thead>
<tr>
<th>Wait</th>
<th>Accept</th>
<th>Reject, TearDown</th>
</tr>
</thead>
<tbody>
<tr>
<td>Success</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Failure</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

**acsvoip:**

```
<table>
<thead>
<tr>
<th>Wait</th>
<th>Avail</th>
<th>Reject, TearDown, Unavail</th>
</tr>
</thead>
<tbody>
<tr>
<td>Success</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Failure</td>
<td></td>
<td></td>
</tr>
<tr>
<td>VoiceMail</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

**Merge:**

```
<table>
<thead>
<tr>
<th>Wait, Wait</th>
<th>?Userstatus [Id=vplus]</th>
<th>?Userstatus [Id=acsvoip]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Success, Success</td>
<td></td>
<td>?Avail [Id = acsvoip]</td>
</tr>
<tr>
<td></td>
<td>?Accept [Id = vplus]</td>
<td>?Unavail [Id = acsvoip]</td>
</tr>
<tr>
<td>Failure, Failure</td>
<td></td>
<td>?Reject, TearDown</td>
</tr>
<tr>
<td></td>
<td></td>
<td>VoiceMail</td>
</tr>
</tbody>
</table>
```
The Merge Model

**vplus:**
If participants **accept**, the call is successful

- **Wait**
  - Accept
    - **Success**
  - **Reject, TearDown**
    - **Failure**

**acsvoip:**
If participants are **available**, the call is successful

- **Wait**
  - **Avail**
    - **Success**
  - **Reject, TearDown Unavail**
    - **Failure**
    - **VoiceMail**

**Merge:**
If participants are **available** or **accept**, the call is successful

- **Wait, Wait**
  - **?Userstatus [Id=vplus]**
    - **Success, Success**
  - **?Avail [Id = acsvoip]**
    - **?Accept [Id = vplus]**
  - **?Reject, ?TearDown**
    - **?Unavail [Id = acsvoip]**
    - **Failure, Failure**
    - **VoiceMail**
Observations

-The merged model
  ✔ captures the behaviours of individual models
  ✔ highlights the similarities and differences between them

-But,
  ✗ it cannot capture interactions between behaviours of the original models
The Merge Model

**vplus:**
If participants *accept*, the call is successful

**acsvoip:**
If participants are *available*, the call is successful

**Merge:**
If participants are *available* or *accept*, the call is successful
Research Hypothesis

Behavioural model merging and inconsistency resolution are inter-related activities that can be effectively supported by tools and techniques based on property-preserving relations.
Research Hypothesis

Behavioural model merging and inconsistency resolution are inter-related activities that can be effectively supported by tools and techniques based on property-preserving relations.
Requirements for the Framework
Requirements for the Framework

Compute merge and check consistency

Original models

Original models

Original models
Requirements for the Framework

Compute merge and check consistency

Original models

consistent

Merge model

Original models
Requirements for the Framework

Original models

Compute merge and check consistency

inconsistent

Explore and prioritize inconsistencies

List of inconsistent items, or a structure that highlights inconsistencies

consistent

Merge model

Original models
Requirements for the Framework

- Compute merge and check consistency
- Explore and prioritize inconsistencies
- User properties
- List of inconsistent items, or a structure that highlights inconsistencies
- User properties
- Merge model
- Original models
- Consistent
- Inconsistent
- Original models
Requirements for the Framework

1. **Compute merge and check consistency**
2. **Explore and prioritize inconsistencies**
3. **Resolve inconsistencies**

- Original models
- User properties
- Priority list of items
- List of inconsistent items, Or a structure that highlights inconsistencies
- Original models
- Merge model
- User properties
Requirements for the Framework

Compute merge and check consistency

Explore and prioritize inconsistencies

Resolve inconsistencies

Consistent models

Inconsistent models

List of inconsistent items, or a structure that highlights inconsistencies

User properties

Priority list of items

Consistent models

Original models

User properties

Merge model

Original models
Requirements for the Framework

1. Compute merge and check consistency
2. Explore and prioritize inconsistencies
3. Resolve inconsistencies

- User properties
- Priority list of items
- Consistent models
- Original models
- List of inconsistent items, or a structure that highlights inconsistencies
- Consistent models
- Merge model
- User properties
- Original models

Consistent models
Desirable Properties of Merge

Soundness

- Merge model should exhibit the behaviours of the original models

\[ M_1 \models \varphi \lor M_2 \models \varphi \Rightarrow M_{merge} \models \varphi \]

- Merge model should report the inconsistent behaviours

\[ M_1 \models \varphi \land M_2 \models \neg \varphi \Rightarrow M_{merge} \vdash \bot \]

Precision

- Merge model should not add extra behaviours

\[ M_{merge} \models \varphi \Rightarrow M_1 \models \varphi \lor M_2 \models \varphi \]
Representing Knowledge

value of \( p \) is contradictory

\[ p = \mathcal{I} \]

value of \( p \) is unknown

\[ p = ? \]

increase in knowledge
Representing Models

Must (definite) and may (possible) transitions

- Complete and consistent: must = may, and all variables are boolean
- Incompleteness: must ⊂ may, or some variables are unknown
- Inconsistent: must ⊄ may, or some variables are contradictory
Merge Model: Definition

Merge is a common refinement

- Preserves common behaviours (sound)

\[ M_1 \]
\[ M_2 \]
\[ M_{merge} \]

- Every behaviour definite in either \( M_1 \) or \( M_2 \) must be definite in \( M_{merge} \) as well
- Every behaviour possible in \( M_{merge} \) must be possible in both \( M_1 \) and \( M_2 \)
Merge Model: Inconsistency

\[ M_1 \text{ and } M_2 \text{ are inconsistent if their common refinements are inconsistent models} \]
Several common refinements

- The least common refinement
- Identify commonalities with the help of users
- Compute the commonalities
Research Hypothesis

Behavioural model merging and inconsistency resolution are inter-related activities that can be effectively supported by tools and techniques based on property-preserving relations.
Behavioural model merging and inconsistency resolution are inter-related activities that can be effectively supported by tools and techniques based on property-preserving relations.
The Current Framework

Original models

Compute merge and check consistency

Explore and prioritize inconsistencies

Resolve inconsistencies

Consistent models

Consistent models

User properties

List of inconsistent items, or a structure that highlights inconsistencies

User properties

Priority list of items

Merge model

Original models
The Current Framework

1. Compute merge and check consistency
2. Explore and prioritize inconsistencies
3. Resolve inconsistencies
   - Consistent models
   - List of inconsistent items, or a structure that highlights inconsistencies
   - Kripke models, mappings (optional)
4. Merge model
   - Consistent models
   - User properties

Priority list of items

User properties
The Current Framework

Kripke models, mappings (optional)

Compute merge and check consistency

Explore and prioritize inconsistencies

Inconsistent merge model

Consistent models

Consistent models

Priority list of items

User properties

User properties

Merge model

Kripke models, mappings (optional)
The Current Framework

Compute merge and check consistency

Inconsistent merge model

User properties

Model-Checker (uses multi-valued logics for analysis)

Inconsistent models

Consistent models

Resolve inconsistencies

Priority list of items

User properties

Kripke models, mappings (optional)

Consistent models

Merge model

Kripke models, mappings (optional)
The Current Framework

- Compute merge and check consistency
- Resolve inconsistencies
- Consistent models
- Inconsistent merge model
- Kripke models, mappings (optional)
- User properties
- Consistent models
- Model-Checker (uses multi-valued logics for analysis)
- A list of variables & transitions extracted from CEXs
- Merge model

User properties
The Current Framework

Compute merge and check consistency

Inconsistent merge model

Consistent merge model

Kripke models, mappings (optional)

User properties

Model-Checker (uses multi-valued logics for analysis)

A list of variables & transitions extracted from CEXs

Choose a proposal for further exploration

Compute consistent proposals

Kripke models, mappings (optional)

User properties

Choose a proposal for further exploration
The Current Framework

Conflicts in mappings

Compute merge and check consistency

Kripke models, mappings (optional)

User properties

Choose a proposal for further exploration

Compute consistent proposals

Model-Checker (uses multi-valued logics for analysis)

Inconsistent merge model

A list of variables & transitions extracted from CEXs

Choose a proposal for further exploration

User properties

Kripke models, mappings (optional)

Merge model

Inconsistent

consistent

consistent

consistent

Choose a proposal for further exploration

A list of variables & transitions extracted from CEXs

User properties

Choose a proposal for further exploration

Kripke models, mappings (optional)

Compute merge and check consistency

User properties
Current Results

- Created an algorithm for computing a sound merge for Kripke structures
- Created a methodology for negotiation and conflict resolution
  - Detect causes of inconsistency at the level of variables and transitions
  - Produce alternatives for resolving the most significant inconsistencies
  - Measure degree of inconsistency between models
  - Accommodate user preferences
  - Provide (limited) feedback on user choices
Future Work: Theory and Approach

Investigate techniques for computing mappings that capture similar states

- help users identify the differences between alternative mapping choices
- obtain optimum mappings through exploring local similarities

Develop methods for managing negotiation

- partitioning the list of inconsistent items
- help users to browse through different proposals

Extend the framework to work with other behavioural models
Future Work: Evaluation

- Realistic case-studies
- Better tool support
- Our notion of success, given realistic models
  - can find appropriate merges
  - the number of inconsistent items is reasonable
  - inconsistencies can be effectively resolved
    - number of proposals is not too large
    - heuristics for choosing between them are well-understood
    - method is compositional
Questions

Need help on

- additional related work
- comments on this approach
- ideas on how to evaluate the work
- suggestions for future work
Thank You!

Questions?

Acknowledgments: Marsha Chechik, Pamela Zave, Mehrdad Sabetzadeh, Steve Easterbrook, Sebastian Uchitel, Greg Brunet, Nan Niu.
Where to Merge?

➡ Viewpoints modelling
   ➡ Models originating from different sources
   ➡ Need to combine incomplete and inconsistent views

➡ Distributed Software Engineering
   ➡ Separate models are being produced in parallel
   ➡ Need to periodically merge models

➡ Feature Composition/Interaction
   ➡ Requirements as units of functionality or features
   ➡ Need to add/remove features and detect interactions
Where to Merge?

➡️ Software Refactoring or Restructuring
  ➡️ Duplication makes software maintenance difficult
  ➡️ Need to merge duplicate models

➡️ Software Evolution
  ➡️ Software programs evolve continuously
     ➢ patches used to work with an old version, no longer work with the new one
  ➡️ Need to merge patches with the new version
Other Important Factors

- History of changes
- Minimum number of changes
- Accommodating user knowledge and preferences
- Readability and precision
- Measuring degree of inconsistency between models
**Merge Model: Definition**

- Merge is a common refinement
  - Preserves common behaviours (sound)

- Every behaviour definite in either $M_1$ or $M_2$ must be definite in $M_{\text{merge}}$ as well
- Every behaviour possible in $M_{\text{merge}}$ must be possible in both $M_1$ and $M_2$
Merge Model: Definition

Merge is a common refinement

- Preserves common behaviours (sound)

Every behaviour definite in either $M_1$ or $M_2$ must be definite in $M_{merge}$ as well

Every behaviour possible in $M_{merge}$ must be possible in both $M_1$ and $M_2$
**Merge Model: Inconsistency**

\[ M_1 \text{ and } M_2 \text{ are inconsistent if their common refinements are inconsistent models} \]
Merge Model: Inconsistency

\( M_1 \) and \( M_2 \) are inconsistent if their common refinements are inconsistent models.

\[ M_{\text{merge}} = \left\{ \langle s_0, t_0 \rangle \right\} \]

\( p = \frac{1}{2}, q = \frac{1}{2} \)

\( s_2, t_1 \) \( s_1, t_2 \)