Towards a Uniform Framework to Support the Evolution of Software Models

Magdalena Widl

Business Informatics Group
Vienna University of Technology
Motivation and scope
Model-driven engineering
Motivation and scope

Model-driven engineering

Software verification, testing
Motivation and scope

Model-driven engineering

Model verification?

Software verification, testing
Motivation and scope
Model-driven engineering

Model verification?
Software verification, testing

Idea
Find errors on model level
Then work only on consistent models
Motivation and scope

Inconsistencies

<table>
<thead>
<tr>
<th>Static diagrams</th>
<th>Dynamic diagrams</th>
<th>Both</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

example from Calvanese et al., slides from ESSLI Summer School 2003, Vienna

Can this be instantiated?

<table>
<thead>
<tr>
<th>Off</th>
<th>Off</th>
<th>Idle</th>
<th>Idle</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

preparing coffee

coffee complete

prepare tea

define by

User

Coffee Machine

turnOn: void

prepareCoffee(strength: Int): Coffee

prepareTea(): Tea

turnOff: void
Inconsistencies

Static diagrams

Dynamic diagrams

Both

Can this be instantiated?

example from Calvanese et al., slides from ESSLI Summer School 2003, Vienna
Motivation and scope

Inconsistencies

<table>
<thead>
<tr>
<th>Static diagrams</th>
<th>Dynamic diagrams</th>
<th>Both</th>
</tr>
</thead>
</table>

Example from Calvanese et al., slides from ESSLI Summer School 2003, Vienna

Can this be instantiated?

User

CoffeeMachine

- turnOn() : void
- prepareCoffee(strength : Int) : Coffee
- prepareTea() : Tea
- turnOff() : void

Coffee

- coffeeComplete() : void
- prepareCoffee() : Coffee
- teaComplete() : Tea
- prepareTea() : Tea
Motivation and scope

Inconsistencies

Static diagrams  Dynamic diagrams  Both

u:U

off

on

prepareCoffee

prepareTea

coffeeComplete

teaComplete

implemented by

User

CoffeeMachine

turnOn() : void

prepareCoffee(strength : Int) : Coffee

prepareTea() : Tea

turnOff() : void

Prepared Coffee

preparing Coffee

preparing Tea

Prepared Tea

defined by

u:U

cm:CM

turnOn()

coffeeComplete()

prepareCoffee()

teaComplete()

prepareTea()
# Motivation and scope

## Inconsistencies

<table>
<thead>
<tr>
<th>Static diagrams</th>
<th>Dynamic diagrams</th>
<th>Both</th>
</tr>
</thead>
<tbody>
<tr>
<td>Person</td>
<td>Italian</td>
<td>English</td>
</tr>
<tr>
<td>Lazy</td>
<td>Latin Lover</td>
<td>Gentleman</td>
</tr>
<tr>
<td>Hooligan</td>
<td>{disjoint}</td>
<td>{disjoint,complete}</td>
</tr>
</tbody>
</table>

Example from Calvanese et al., slides from ESSLI Summer School 2003, Vienna

Can this be instantiated?

- Off
- Off
- Idle
- Idle

<table>
<thead>
<tr>
<th>Preparing Coffee</th>
<th>Preparing Tea</th>
</tr>
</thead>
<tbody>
<tr>
<td>turnOn()</td>
<td>prepareCoffee()</td>
</tr>
<tr>
<td>coffeeComplete()</td>
<td>prepareTea()</td>
</tr>
</tbody>
</table>

User implemented by CoffeeMachine:

- turnOn() : void
- prepareCoffee(strength : Int) : Coffee
- prepareTea() : Tea
- turnOff() : void

Defined by *u:U cm:CM*

- Off
- Off
- Idle
- Idle

<table>
<thead>
<tr>
<th>Preparing Coffee</th>
<th>Preparing Tea</th>
</tr>
</thead>
<tbody>
<tr>
<td>turnOn()</td>
<td>prepareCoffee()</td>
</tr>
<tr>
<td>coffeeComplete()</td>
<td>prepareTea()</td>
</tr>
</tbody>
</table>

U:U cm:CM
Motivation and scope

Inconsistencies

Static diagrams

Dynamic diagrams

Both

defined by

User

CoffeeMachine

| turnOn() : void |
| prepareCoffee(strength : Int) : Coffee |
| prepareTea() : Tea |
| turnOff() : void |

implemented by

User

CoffeeMachine

turnOn() : void
prepareCoffee(strength : Int) : Coffee
prepareTea() : Tea
turnOff() : void

example from Calvanese et al., slides from ESSLI Summer School 2003, Vienna
Motivation and scope
Model evolution

Evolution is multi-view, multidimensional

May introduce *inconsistencies*

Different evolution tasks may introduce different types of inconsistencies
## Motivation and scope

Model evolution

Evolution is multi-view, multidimensional

May introduce *inconsistencies*

Different evolution tasks may introduce different types of inconsistencies

### Idea

Establish a classification of changes in models and find which inconsistencies they may cause
Motivation and scope

We consider a *multi-view subset* of UML relevant for MDE:

- State machines
- Sequence diagrams
- Class diagrams
Motivation and scope

We consider a *multi-view subset* of UML relevant for MDE:

- State machines
- Sequence diagrams
- Class diagrams

Many complex constructs are omitted, but will be gradually added.
### Example

<table>
<thead>
<tr>
<th>Motivation</th>
<th>Related Work</th>
<th>Proposed Solution</th>
<th>Preliminary Work</th>
<th>Expected Contributions</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Many syntactically correct merges possible. But how to avoid inconsistency with state machine?
Example
Example

Motivation
Related Work
Proposed Solution
Preliminary Work
Expected Contributions

Example
Example

Motivation Related Work Proposed Solution Preliminary Work Expected Contributions

Example

Example

Example

Example

Example

Example
Example

Motivation
Related Work
Proposed Solution
Preliminary Work
Expected Contributions

Off
Off
Idle
Idle
Preparing Coffee
Preparing Coffee
Preparing Tea
Preparing Tea

turnOn

prepareCoffee

coffeeComplete

prepareTea
teaComplete

V
V
V

FAIL

FAIL

OK

OK

Problem
Many syntactically correct merges possible.
But how to avoid inconsistency with state machine?
Example

Motivation Related Work Proposed Solution Preliminary Work Expected Contributions

Example

Off
Idle
Preparing Coffee
Preparing Tea

turnOn

turnOff

prepareCoffee
coffeeComplete

prepareTea
teaComplete

turnOn()
coffeeComplete()
prepareCoffee()
u:U cm:CM

OK

turnOn()
coffeeComplete()
prepareCoffee()
teaComplete()
prepareTea()
u:U cm:CM

FAIL

turnOn()
coffeeComplete()
prepareCoffee()
turnOff()
teaComplete()
prepareTea()
turnOff()
u:U cm:CM

FAIL

Problem
Many syntactically correct merges possible.
But how to avoid inconsistency with state machine?
Example
Example

Motivation
Related Work
Proposed Solution
Preliminary Work
Expected Contributions

Off  Off  Idle  Idle
Preparing Coffee  Preparing Coffee  Preparing Tea  Preparing Tea

turnOn()

prepareCoffee()
don

prepareTea()
don

turnOff()

coffeeComplete()
don

teaComplete()
don

Problem
Many syntactically correct merges possible.
But how to avoid inconsistency with state machine?
**Example**

---

Motivation  
Related Work  
Proposed Solution  
Preliminary Work  
Expected Contributions

---

**Example**

```
V0  turnOn()  cm:CM  OK
  |      |                  |
  |      |  prepareCoffee()  |
  |      |                  |
  |      |  coffeeComplete() |

u:U  turnOn()  cm:CM  OK
  |                  |
  |  prepareCoffee() |
  |                  |
  |  coffeeComplete() |

```

**Problem**

Many syntactically correct merges possible. But how to avoid inconsistency with state machine?
Example

\[ V_0 \]

\[ V_0' \]

Related Work

Proposed Solution

Preliminary Work

Expected Contributions

Motivation

Many syntactically correct merges possible. But how to avoid inconsistency with state machine?
Example

Motivation
Related Work
Proposed Solution
Preliminary Work
Expected Contributions

Off
Off
Idle
Idle
Preparing Coffee
Preparing Coffee
Preparing Tea
Preparing Tea

turnOn
turnOff
prepareCoffee
coffeeComplete
prepareTea
teaComplete

Off
Off
Idle
Idle
Preparing Coffee
Preparing Coffee
Preparing Tea
Preparing Tea

turnOn
turnOff
prepareCoffee
coffeeComplete
prepareTea
teaComplete

u:U
cm:CM

turnOn() prepareCoffee()
coffeeComplete() turnOff()

FAIL

FAIL

OK

OK

V₀

V₀′
Example

Motivation
Related Work
Proposed Solution
Preliminary Work
Expected Contributions

6
Example

- Motivation
- Related Work
- Proposed Solution
- Preliminary Work
- Expected Contributions

### Example

```
V₀
```

- **V₀'**
  - u:U
  - cm:CM
  - turnOn()
  - prepareCoffee()
  - coffeeComplete()
  - turnOff()
  - prepareCoffee()
  - coffeeComplete()

- **V₀**
  - u:U
  - cm:CM
  - turnOn()
  - prepareCoffee()
  - coffeeComplete()

- **V₀''**
  - u:U
  - cm:CM
  - turnOn()
  - prepareCoffee()
  - coffeeComplete()
  - prepareTea()
  - teaComplete()

---

### State Machine

```
Off
```

- **Idle**
  - turnOn()
  - turnOff()
  - prepareCoffee
  - coffeeComplete
  - prepareTea
  - teaComplete

- **Preparing Coffee**
  - prepareCoffee
  - coffeeComplete

- **Preparing Tea**
  - prepareTea
  - teaComplete
Example

\[ V_0' \quad \text{OK} \quad V_0'' \quad \text{OK} \]

\[ V_1'? \]
Example

\[ V_0' \quad \text{OK} \quad V_0'' \quad \text{OK} \]

\[ V_1? \]

TurnOn()  
PrepareCoffee()  
TurnOff()  
PrepareTea()  
CoffeeComplete()  
TeaComplete()
Example

\[ V_0' \quad \text{OK} \quad V_0'' \quad \text{OK} \]

\[ V_1? \]

\[ \text{FAIL} \]

Many syntactically correct merges possible. But how to avoid inconsistency with state machine?
Example

Motivation
Related Work
Proposed Solution
Preliminary Work
Expected Contributions

Example

\[V_0'\]

\[V_0''\]

\[V_1?\]

\[\text{OK}\]

\[\text{FAIL}\]

\[\text{OK}\]

\[\text{FAIL}\]
Example
Example

Motivation Related Work Proposed Solution Preliminary Work Expected Contributions

Example
Example

Motivation | Related Work | Proposed Solution | Preliminary Work | Expected Contributions
--- | --- | --- | --- | ---

### Example

#### V₀’

- `turnOn()`
- `prepareCoffee()`
- `coffeeComplete()`
- `turnOff()`

#### V₀”

- `turnOn()`
- `prepareCoffee()`
- `coffeeComplete()`
- `prepareTea()`
- `teaComplete()`
- `turnOff()`

#### V₁?

- `turnOn()`
- `prepareCoffee()`
- `coffeeComplete()`
- `turnOff()`
- `prepareTea()`
- `teaComplete()`

#### Problem

Many syntactically correct merges possible. But how to avoid inconsistency with state machine?
Example

Problem
Many syntactically correct merges possible. But how to avoid inconsistency with state machine?
<table>
<thead>
<tr>
<th>Related work</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model Evolution</td>
</tr>
</tbody>
</table>

- Mens et al. 
- Challenges in Software Evolution 
- Challenges in Model-Driven Software Engineering 
- Brosch et al. 
- Model Versioning, Change Management 
- Maoz et al. 
- Semantic model differencing 
- Knapp et al., Schaefer et al., Eshuis et al. 
- Model checking dynamic UML diagrams 
- v.d. Straeten et al. 
- Description Logics 
- Formal Semantics of UML 
- Rumpe et al. 
- System Model 
- Eshuis et al. 
- Activity Diagrams 
- Luettgen and Mendler 
- Statechart Semantics via Intuitionistic Kripke Models
Related work

- Model Evolution
  - *Mens et al.* Challenges in Software Evolution
  - *Mens et al.* Challenges in Model-Driven Software Engineering
  - *Brosch et al.* Model Versioning, Change Management
Related work

- **Model Evolution**
  - *Mens et al.* Challenges in Software Evolution
  - *Mens et al.* Challenges in Model-Driven Software Engineering
  - *Brosch et al.* Model Versioning, Change Management

- **Model Verification**
Related work

- **Model Evolution**
  - *Mens et al.* Challenges in Software Evolution
  - *Mens et al.* Challenges in Model-Driven Software Engineering
  - *Brosch et al.* Model Versioning, Change Management

- **Model Verification**
  - *Maoz et al.* Semantic model differencing
  - *v.d. Straeten et al.* Description Logics
Related work

- Model Evolution
  - Mens et al. Challenges in Software Evolution
  - Mens et al. Challenges in Model-Driven Software Engineering
  - Brosch et al. Model Versioning, Change Management

- Model Verification
  - Maoz et al. Semantic model differencing
  - Knapp et al., Schaefer et al., Eshuis et al. Model checking dynamic UML diagrams
  - v.d. Straeten et al. Description Logics

- Formal Semantics of UML
Related work

- **Model Evolution**
  - Mens et al. Challenges in Software Evolution
  - Mens et al. Challenges in Model-Driven Software Engineering
  - Brosch et al. Model Versioning, Change Management

- **Model Verification**
  - Maoz et al. Semantic model differencing
  - Knapp et al., Schaefer et al., Eshuis et al. Model checking dynamic UML diagrams
  - v.d. Straeten et al. Description Logics

- **Formal Semantics of UML**
  - Rumpe et al. System Model
  - Eshuis et al. Activity Diagrams
  - Luettggen and Mendler Statechart Semantics via Intuitionistic Kripke Models
Approach and methodology

State of the art survey
Approach and methodology

State of the art survey
- Model evolution
- Model verification
- Semantics of UML
Approach and methodology

State of the art survey

Taxonomy of change
Approach and methodology

State of the art survey

Taxonomy of change

- Change in model evolution
- Definitions of inconsistencies
- Relations between change and inconsistency
Approach and methodology

State of the art survey

Taxonomy of change

UML subset
Approach and methodology

State of the art survey

Taxonomy of change

UML subset
Approach and methodology

State of the art survey

Taxonomy of change

UML subset

Verification methods
Approach and methodology

State of the art survey

Taxonomy of change

UML subset

Verification methods

- Model checking (focus on dynamic view)
- Analysis of state space
- Own model checker for software models?
Approach and methodology

State of the art survey

Taxonomy of change

UML subset

Verification methods

Handling complexity
Approach and methodology

State of the art survey

Taxonomy of change

UML subset

Verification methods

Handling complexity

- Identify complex tasks
- Incremental verification
Approach and methodology

State of the art survey

Taxonomy of change

UML subset

Verification methods

Handling complexity

Evaluation
# Approach and methodology

State of the art survey

Taxonomy of change

UML subset

Verification methods

Handling complexity

## Evaluation
- Eclipse-based implementation
- Benchmarks from previous project
- Students in “Model Engineering” lab
Approach and methodology

State of the art survey

Taxonomy of change

UML subset

Verification methods

Handling complexity

Evaluation
Semantics-aware model versioning
Using the SPIN model checker
Semantics-aware model versioning

Using the SPIN model checker
Semantics-aware model versioning
Using the SPIN model checker

Off:
printf("Off", CM[h]);
if
:: CM[h] == turnOn -> h++; goto Idle
:: CM[h] == acc -> goto end
fi;
Semantics-aware model versioning
Using the SPIN model checker

Idle:

printf("Idle", CM[h]);

if
    :: CM[h] == prepareCoffee -> h++; goto PrepareCoffee
    :: CM[h] == prepareTea -> h++; goto PrepareTea
    :: CM[h] == turnOff -> h++; goto Off
    :: CM[h] == acc -> goto end
fi;
Semantics-aware model versioning
Using the SPIN model checker

Preparing Coffee:
```plaintext
printf("PrepareCoffee", CM[h]);
if
:: CM[h] == coffeeComplete -> h++; goto Idle
:: CM[h] == acc -> goto end
fi;
```
Semantics-aware model versioning
Using the SPIN model checker
Semantics-aware model versioning
Using the SPIN model checker
Semantics-aware model versioning
Using the SPIN model checker
Semantics-aware model versioning
Using the SPIN model checker
Semantics-aware model versioning
Using the SPIN model checker
Semantics-aware model versioning
Using the SPIN model checker
Semantics-aware model versioning
Using the SPIN model checker
Semantics-aware model versioning

Using the SPIN model checker
Semantics-aware model versioning
Using the SPIN model checker
Semantics-aware model versioning
Using the SPIN model checker

mtype CM[7]; CM[0] = turnOn; CM[1] = prepareCoffee;
CM[6] = acc;
Semantics-aware model versioning
Using the SPIN model checker
Semantics-aware model versioning
Using the SPIN model checker
Semantics-aware model versioning
Using the SPIN model checker
Semantics-aware model versioning
Using the SPIN model checker
Semantics-aware model versioning
Using the SPIN model checker
Semantics-aware model versioning
Using the SPIN model checker
Semantics-aware model versioning
Using the SPIN model checker
Semantics-aware model versioning
Using the SPIN model checker
Semantics-aware model versioning
Using the SPIN model checker
Semantics-aware model versioning
Using the SPIN model checker
Semantics-aware model versioning
Using the SPIN model checker
Semantics-aware model versioning

Using the SPIN model checker
Semantics-aware model versioning
Using the SPIN model checker
Expected Contributions

- Survey on model evolution
- Taxonomy of change and inconsistencies
- Verification methods
- Integration into a formal framework to assist MDE