<table>
<thead>
<tr>
<th>title</th>
<th>Formal Verification of UML-based Specifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>author(s)</td>
<td>Soeken, Mathias</td>
</tr>
<tr>
<td>citation</td>
<td>2010年度科学技術振興機構瑞々卿離散構造処理系プロジェクト講究録 pp.205-221.</td>
</tr>
<tr>
<td>issue date</td>
<td>2011-06</td>
</tr>
<tr>
<td>doc url</td>
<td><a href="http://hdl.handle.net/2115/48438">http://hdl.handle.net/2115/48438</a></td>
</tr>
<tr>
<td>type</td>
<td>conference presentation</td>
</tr>
<tr>
<td>note</td>
<td>ERATOセミナ No.34. 2011年1月31日</td>
</tr>
<tr>
<td>file info</td>
<td>34_all.pdf</td>
</tr>
</tbody>
</table>

The document contains information about the formal verification of UML-based specifications, including the author, citation details, and related notes.
In the recent years, researchers started to enrich the classical textbook specification of embedded systems by more formal descriptions such as the Unified Modeling Language (UML). This enables to lift the starting point for verification techniques to the specification level and leads to an improved design flow, where crucial flaws can be detected, even if no executable implementation is available. However, research on efficient solving techniques for the respective verification tasks is just at the beginning. In the talk, the improved design flow based on specifications enriched with formal descriptions is introduced. We illustrate how early design flaws become evident within this flow and briefly review algorithms aimed at detecting them. Furthermore, first experimental results are presented. Overall, the aim of the talk is to provide an introduction into this emerging area and to present first results.
Formal Verification of UML-based Specifications

Mathias Soeken, Robert Wille, and Rolf Drechsler
Institute of Computer Science, University of Bremen
Bremen, Germany
msoeken@informatik.uni-bremen.de
January 31, 2011

About
Dipl.-Inf. Mathias Soeken (msoeken@informatik.uni-bremen.de)

Curriculum Vitae
• 2004 - 2008: Received Diploma Degree in CS at University of Bremen
• 2008 - 2009: Internship at Mentor Graphics, Hamburg
• since 2009: PhD Student at the Computer Architecture Group of Prof. Rolf Drechsler at University of Bremen

Research Interests
• Formal Verification
  • PhD Thesis about Specification Checking
  • Reversible Computation and Quantum Computation
  • Supervisor of a Graduate Students Project
  • Development of RevKit, a Toolkit for Reversible Computation

Specification Verification

Specification
(as Textbook)

System model
(e.g. in SystemC)

... manually

System model
(e.g. in SystemC)

...
Specification Verification

Property Checking

Specification (as Textbook)

manually

System model (e.g. in SystemC)

...
**UML Class Diagram**

- **Classes**
- **Attributes**
- **Operations**
- **Associations**
- **Constraints**

**Host**
- `ack: String`
  - `process()`

**Client**
- `req: String`
  - `0..8 clients`
  - `1..2 hosts`

**Command**
- `context Host::process()`
  - `pre: clients->size() > 0`
  - `post: clients@pre->at(0).req = "exit" implies ack = "good"`
  - `inv: ack.isDefined()`

---

**UML Object Diagram**

- **Client**
  - `req: String`
  - `<client1>`
  - `<client2>`

- **Host**
  - `ack: String`
  - `<host>`

- **Command**
  - `inv: a ok. isDefined()`
  - `implies: ack = "good"`

---

**UML Object Diagram**

- **Client**
  - `req: String`
  - `<client1>`
  - `<client2>`

- **Host**
  - `ack: String`
  - `<host>`

- **Command**
  - `inv: a ok. isDefined()`
  - `implies: ack = "good"`

---

**UML Object Diagram**

- **Client**
  - `req: String`
  - `<client1>`
  - `<client2>`

- **Host**
  - `ack: String`
  - `<host>`

- **Command**
  - `inv: a ok. isDefined()`
  - `implies: ack = "good"`
What is Inconsistency?

A
v: Integer
w: Boolean

C
u: Integer

B
x: Integer
y: Integer

D
z: Boolean

2 1

as ... UML-inconsistent Model is OCL-inconsistent

Mathias Soeken et.al. Formal Verification of UML-based Specifications 10/29
What is Inconsistency?

A

v: Integer
w: Boolean
C

u: Integer
B

x: Integer
y: Integer
D

z: Boolean

2 1

as ... UML-inconsistent
Model is OCL-inconsistent

Running Example

Register

name: String
purpose: RegisterType
bitwidth: Integer

1. register

Processor

bitwidth: Integer

1 processor

content Processor inv bv:
register->forall(r|r.bitwidth = bitwidth)
Running Example

Is there a valid system state consisting of one processor and three registers?

<table>
<thead>
<tr>
<th>r0 : Register</th>
<th>name</th>
<th>purpose</th>
<th>bitwidth</th>
</tr>
</thead>
<tbody>
<tr>
<td>r1 : Register</td>
<td>name</td>
<td>purpose</td>
<td>bitwidth</td>
</tr>
<tr>
<td>r2 : Register</td>
<td>name</td>
<td>purpose</td>
<td>bitwidth</td>
</tr>
</tbody>
</table>

| p0 : Processor | bitwidth |

Encoding Attributes

How many bits are needed for the bit-vectors?

- \[ \lceil \log_2 (n + 1) \rceil \] bits are needed. The additional value is for representing undefined attributes. Determination of \( n \) by type of attribute:
  - Boolean: \( n = 2 \)
  - Enumeration: \( n = \text{Number of fields} \)
  - String: \( n = \text{Number of all possible strings} \)
  - Integer: \( n = 2^l - 1 \) for \( l \)-bit integers
Encoding Links
• $\lambda_{\text{Register}}$ is a bit-mask and each bit enables or disables a possible link to a register
• $\lambda_{\text{Register}} = \lambda_0\lambda_1\lambda_2$

Running Example
• $r_0 : \text{Register}$
  - $\alpha_{\text{name}} = 0$
  - $\alpha_{\text{purpose}} = 1$
  - $\alpha_{\text{bitwidth}} = 32$
• $r_1 : \text{Register}$
  - $\alpha_{\text{name}} = 1$
  - $\alpha_{\text{purpose}} = 0$
  - $\alpha_{\text{bitwidth}} = 32$
• $r_2 : \text{Register}$
  - $\alpha_{\text{name}} = 3$
  - $\alpha_{\text{purpose}} = 4$
  - $\alpha_{\text{bitwidth}} = 32$
• $p_0 : \text{Processor}$
  - $\alpha_{\text{bitwidth}} = 32$

Encoding Attributes
How many bits are needed for the bit-vectors?

$\lceil \log(n + 1) \rceil$ bits are needed. The additional value is for representing undefined attributes. Determination of $n$ by type of attribute:

- **Boolean** $n = 2$
- **Enumeration** $n = \text{Number of fields}$
- **String** $n = \text{Number of all possible strings}$
- **Integer** $n = 2^l - 1$ for $l$-bit integers

Mathias Soeken et al. Formal Verification of UML-based Specifications 16/29
Encoding Invariants

context Processor inv bw:

\[ \forall \text{r} \in \text{Register} \Rightarrow (\text{r}.\text{bitwidth} = \text{bitwidth}) \]

• Can easily be converted to a CNF using Boolean transformations
• Other OCL functions can be translated in a similar way

Running Example

\begin{align*}
\text{r0} & : \text{Register} \\
\alpha_{\text{r0}} & = \lambda_0 \lambda_1 \lambda_2 \\
\text{name} & \in B^2 \\
\text{purpose} & \in B^3 \\
\text{bitwidth} & \in B^8 \\
\lambda_{\text{processor}} & = 1 \\
\text{r1} & : \text{Register} \\
\alpha_{\text{r1}} & = \lambda_0 \lambda_1 \lambda_2 \\
\text{name} & \in B^2 \\
\text{purpose} & \in B^3 \\
\text{bitwidth} & \in B^8 \\
\lambda_{\text{processor}} & = 0 \\
\text{r2} & : \text{Register} \\
\alpha_{\text{r2}} & = \lambda_1 \lambda_2 \\
\text{name} & \in B^2 \\
\text{purpose} & \in B^3 \\
\text{bitwidth} & \in B^8 \\
\lambda_{\text{processor}} & = 0 \\
\end{align*}

\begin{align*}
\text{p0} & : \text{Processor} \\
\alpha_{\text{bitwidth}} & \in B^8 \\
\lambda_{\text{register}} & = 100
\end{align*}

Encoding Links

• \( \lambda_{\text{register}} \) is a bit-mask and each bit enables or disables a possible link to a register
• \( \lambda_{\text{register}} = \lambda_0 \lambda_1 \lambda_2 \)
• There is a link between \( p0 \) and \( ri \) iff \( \lambda_i = 1 \)

\begin{align*}
\text{context Processor inv bw:} \\
\text{register->forAll(r|r.bitwidth=bitwidth)}
\end{align*}
Verification Tasks

- Let $I = \{i_1, \ldots, i_n\}$ be a set of invariants
- Let $\sigma$ denote a system state, i.e. a concrete assignment of attributes and links
- Let $\sigma(i) \in B$ with $i \in I$ be the evaluation of $i$ in $\sigma$

Consistency

$\exists \sigma : \bigwedge_{i \in I} \sigma(i)$

Independence If the model is consistent, an invariant $j \in I$ is independent, if

$\exists \sigma : \bigwedge_{i \in I \setminus \{j\}} \sigma(i) \land \neg \sigma(j)$
**Experimental Results**

- Consistency check for consistent models
- Enumerative approach is USE
- UML2Alloy converts UML models to an Alloy model which is solved by a SAT solver

![Graph showing consistency check results](image)


---

**Running Example**

```
VERIFICATION OF DYNAMIC ASPECTS

Button
counter: Integer
requesting() 

 propriety: 
pre: t1.pedLight = false 
post: t1.request = true 
post: counter = counter@pre + 1 

TrafficLight

context requesting() 

propriety:
pre: request = true 
post: counter = counter@pre + 1 
```

---

```
context switchPedLight() 

propriety:
pre: request = true 
post: pedLight != pedLight@pre 
```

```
context switchCarLight() 

propriety:
pre: request = true 
post: carLight != carLight@pre 
```

```
inv: not(pedLight = true and carLight = true) 
```
Running Example

Time: 0
context Button::requesting()
pre: tl.pedLight = false
post: tl.request = true
post: counter ... 0
b2: Button
counter = 0

Car Pedestrian

Mathias Soeken et.al. Formal Verification of UML-based Specifications 23/29

Running Example

Time: 1
context Button::requesting()
pre: tl.pedLight = false
post: tl.request = true
post: counter ... 1
b2: Button
counter = 0

Car Pedestrian

Mathias Soeken et.al. Formal Verification of UML-based Specifications 23/29

Running Example

Time: 2
context Button::requesting()
pre: tl.pedLight = false
post: tl.request = true
post: counter ... 1
b2: Button
counter = 0

Car Pedestrian

Mathias Soeken et.al. Formal Verification of UML-based Specifications 23/29
Encoding

- Introduce time model, i.e. we consider $k$ steps (operation calls)
- This leads to $k + 1$ system states $s_0, \ldots, s_k$

Let $OP$ be the set of all possible operation calls

- Create $\bar{\alpha}$ and $\bar{\lambda}$ variables for each time step

Running Example

Time: 3

context Button::requesting()
pre: tl.pedLight = false
post: tl.request = true
post: counter = 0
post: tl.request = true
b1: Button
counter = 1
Car
Pedestrian
b2: Button
counter = 0

context TrafficLight
pre: tl.pedLight = true
post: carLight = false
pre: tl.request = true
post: pedLight = true
b1: TrafficLight
pedLight = true
carLight = false
pre: request = true
post: request = false
b2: TrafficLight
pedLight = false
carLight = true
pre: request = true
post: request = false

Mathias Soeken et al.
Formal Verification of UML-based Specifications
**Encoding**

- Introduce time model, i.e. we consider $k$ steps (operation calls)
- This leads to $k + 1$ system states $s_0, \ldots, s_k$
- Create $\alpha$ and $\lambda$ variables for each time step
- Let $OP$ be the set of all possible operation calls
- Let $op_i$ with $i \in \{0, \ldots, k-1\}$ the operation call executed in system state $i$
- Besides the invariants, let $\prec_{op}$ and $\succ_{op}$ with $op \in OP$ the pre- and post-conditions of the operation associated to the operation call $op$

**Experimental Results**

<table>
<thead>
<tr>
<th>Name</th>
<th>Task</th>
<th>#Obj</th>
<th>Depth</th>
<th>Status</th>
<th>Run-time</th>
</tr>
</thead>
<tbody>
<tr>
<td>switch</td>
<td>Reachability</td>
<td>25</td>
<td>23</td>
<td>sat</td>
<td>49.70</td>
</tr>
<tr>
<td>switch</td>
<td>Reachability</td>
<td>25</td>
<td>22</td>
<td>unsat</td>
<td>50.00</td>
</tr>
<tr>
<td>switch</td>
<td>Reachability</td>
<td>25</td>
<td>50</td>
<td>sat</td>
<td>621.70</td>
</tr>
<tr>
<td>switch</td>
<td>Reachability</td>
<td>9</td>
<td>103</td>
<td>sat</td>
<td>147.50</td>
</tr>
<tr>
<td>switch</td>
<td>Reachability</td>
<td>9</td>
<td>102</td>
<td>unsat</td>
<td>90.40</td>
</tr>
<tr>
<td>simple-cpu2</td>
<td>State Gen.</td>
<td>13</td>
<td>100</td>
<td>sat</td>
<td>1.30</td>
</tr>
<tr>
<td>simple-cpu2</td>
<td>State Gen.</td>
<td>13</td>
<td>100</td>
<td>unsat</td>
<td>0.40</td>
</tr>
<tr>
<td>traffic-control</td>
<td>Reachability</td>
<td>6</td>
<td>5</td>
<td>sat</td>
<td>0.00</td>
</tr>
<tr>
<td>traffic-control</td>
<td>Reachability</td>
<td>24</td>
<td>10</td>
<td>sat</td>
<td>1.60</td>
</tr>
<tr>
<td>traffic-control</td>
<td>State Gen.</td>
<td>9</td>
<td>30</td>
<td>unsat</td>
<td>0.10</td>
</tr>
<tr>
<td>traffic-control</td>
<td>State Gen.</td>
<td>9</td>
<td>100</td>
<td>unsat</td>
<td>0.40</td>
</tr>
</tbody>
</table>
References

Formal Verification of
UML-based Specifications

Mathias Soeken, Robert Wille, and Rolf Drechsler
Institute of Computer Science, University of Bremen
Bremen, Germany
msoeken@informatik.uni-bremen.de
January 31, 2011