Constraint-based Platform Variants Specification for Early System Verification

Andreas Burger¹, Alexander Viehl¹, Andreas Braun¹, Finn Haedicke²,⁴, Daniel Große², Oliver Bringmann¹,³ and Wolfgang Rosenstiel¹,³

¹FZI Research Center for Information Technologies
²solvertec GmbH
³University of Tuebingen
⁴Institute of Computer Science, University of Bremen
Design Challenges for Automotive Electronics

- Vehicles have become smarter over the last years
- Significant increase of software in the automotive
  - Multi-sensor data fusion
  - Complex image recognition algorithms
  - Usage of background information (maps, GPS)
  - Situation perception, interpretation & reasoning
Design Challenges for Automotive Electronics

- Increase of reused logic and IP integration in the platform design composition

- Significant increase of platform variant and configuration space, e.g.:
  - 6.4 million valid variants of an automatic gear shifting application in Daimler Trucks
  - $10^{21}$ valid MOST network variants
Design Challenges for Automotive Electronics

- New challenges in verification, exploration and test:
  - Huge variant spaces
  - Verification of IP-Blocks in different platforms
  - Interaction of different IP-Block instances
  - Verification of different platform characteristics (e.g., software versions, component parameter, etc.)

- Therefore virtual prototyping can be used in early verification

- Hence focus is moving away from fixed virtual platforms to variable virtual platforms
Constraint-based Platform Variant Specification

- Platform Variants Specification
  - Structural Templates
  - Platform Constraint Sets

➢ Platform Variants Specification
Constraint-based Platform Variant Specification

- Platform Variants Specification
- Structural Templates
- Platform Constraint Sets

Constraint Transformation / Solving Process

- Platform Variants Specification
- Platform Constraints
Constraint-based Platform Variant Specification

- Platform Variants Specification
- Platform Constraints
- Platform Variant Generation

Platform Variants Specification
- Structural Templates
- Platform Constraint Sets

Constraint Transformation / Solving Process

Platform Variant Generation

Virtual Platform Variants
- XML File
Constraint-based Platform Variant Specification

- Platform Variants Specification
- Platform Constraints
- Platform Variant Generation
- Adaption of Virtual Platform Variants

Platform Variants Specification
- Structural Templates
- Platform Constraint Sets

Constraint Transformation / Solving Process

Virtual Platform Variants
- XML File

Platform Variant Generation

System Simulation
- Platform Adaption
  - VP
- Testbench

1/23/2014
Constraint-based Platform Variant Specification

- Platform Variants Specification
- Platform Constraints
- Platform Variant Generation
- Adaption of Virtual Platform Variants
Platform Variants Specification

- Model-based description approach
- Specification of platform variants structure
  - Hierarchical structured Templates based on UML
    - Platform UML Profile
    - UML Class Diagrams
    - UML Composite Structure Diagrams
- Specification of different configuration possibilities
  - Attached constraint sets to specify feasible variants and configuration parameter
Platform Variants Specification

- **UML Profile** defines different platform types
- **<<vp_property>>** specifies parameter which can be configured
- **<<sc_module>>** virtual prototype modules are defined as UML::Classes
- **<<vp_entity>>** specifies instance of a virtual prototype module

### Example

<table>
<thead>
<tr>
<th>&lt;&lt;sc_module&gt;&gt;</th>
<th>DisplayAdapter</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;&lt;vp_property&gt;&gt;</td>
<td>m_buffer : mt_uint32</td>
</tr>
<tr>
<td></td>
<td>...</td>
</tr>
<tr>
<td>&lt;&lt;vp_property&gt;&gt;</td>
<td>...</td>
</tr>
<tr>
<td>&lt;&lt;vp_property&gt;&gt;</td>
<td>...</td>
</tr>
</tbody>
</table>

Instance of

```
<<vp_entity>>
My_DA:DisplayAdapter
```
Platform Variants Specification

- UML Profile defines different platform types
- <<vp_template>> abstracts a part of a system for simplicity, variability or structural reasons
- <<vp_template>> contains entities and even other templates and can be used in variants specification
Constraint-based Platform Variant Specification

- Platform Variants Specification
- Platform Constraints
- Platform Variant Generation
- Adaption of Virtual Platform Variants
Platform Constraints

- Constraints are specified by an extended subset of the Object Constraint Language (OCL)
- OCL commonly defines constraints at the M1 layer of Meta Object Facility (MOF)

<table>
<thead>
<tr>
<th>Standard MOF Layer</th>
<th>Platform Meta Layer</th>
</tr>
</thead>
<tbody>
<tr>
<td>M3</td>
<td>Meta Meta Model</td>
</tr>
<tr>
<td></td>
<td>UML Meta Model</td>
</tr>
<tr>
<td>M2</td>
<td>UML Meta Model</td>
</tr>
<tr>
<td></td>
<td>Platform Templates / Profile</td>
</tr>
<tr>
<td>M1</td>
<td>User-defined UML- / Object-models</td>
</tr>
<tr>
<td></td>
<td>Platform Variant Specification</td>
</tr>
<tr>
<td>M0</td>
<td>Distinctive Data</td>
</tr>
<tr>
<td></td>
<td>Platform Variant Space</td>
</tr>
</tbody>
</table>
Platform Constraints – P-OCL

- OCL subset supports:
  - Boolean operators:
    \(<, >, \leq, \geq, \>
    \text{and, or, if} - \text{then} - \text{else}, ...
  - OCL Collection operators:
    \text{includes()}, \text{size()}, ...

- OCL Extensions:
  - Probability distribution operators for OCL Collection-Type \text{Sequence}:
    \text{gaussian()}, ...
  - Special Template-Operators:
    \text{active()}, ...
Constraint transformation in Boolean formulas to use SMT/SAT Solver (metaSMT, Z3, PicoSAT, etc.)

Automatically transformation in Quantifier-free bit-vector (QF-BV) logic
Platform Constraints – Transformation

- Transformation of P-OCL constraints in Quantifier-free bit-vector (QF-BV) logic
- Numbers are converted in bit-vectors
- QF-BV logic is expressed in metaSMT Python code
- P-OCL Example:
Platform Constraints – Transformation

- Transformation of P-OCL constraints in Quantifier-free bit-vector (QF-BV) logic
- Numbers are converted in bit-vectors
- QF-BV logic is expressed in metaSMT Python code
- P-OCL Example:

\[
\text{Sequence}\{2..16\} \rightarrow \text{select}\,(e \mid e/2 = 0) \rightarrow \\
\text{includes}\,(\text{self.allInstances}() \rightarrow \text{size}())
\]

- Boolean formula:

\[
y \geq \vec{a} \ \& \ \ y \leq \vec{b} \ \& \ \ (y - \vec{a}) \ % \ \vec{s} = = 0
\]

- Whereby:

\[
\vec{a} \equiv \text{conv}^{-1}(2), \ \vec{b} \equiv \text{conv}^{-1}(16) \ \text{and} \ \vec{s} \equiv \text{conv}^{-1}(2)
\]
Platform Constraints – Transformation

- Transformation of P-OCL constraints in Quantifier-free bit-vector (QF-BV) logic
- Numbers are converted in bit-vectors
- QF-BV logic is expressed in metaSMT Python code
- P-OCL Example:

\[
\text{Sequence}\{2..16\} \rightarrow \text{select}\( e \mid e/2 = 0\)\rightarrow \\
\text{includes(}\text{self\_allInstances()}\rightarrow\text{size()}\)
\]

- Boolean formula:

\[
y \geq \tilde{a} \land y \leq \overrightarrow{b} \land (y - \tilde{a}) \% \tilde{s} == 0
\]

- Whereby:

\[
\tilde{a} \equiv \text{conv}^{-1}(2), \overrightarrow{b} \equiv \text{conv}^{-1}(16) \text{ and } \tilde{s} \equiv \text{conv}^{-1}(2)
\]
Platform Constraints – Transformation

- Transformation of P-OCL constraints in Quantifier-free bit-vector (QF-BV) logic
- Numbers are converted in bit-vectors
- QF-BV logic is expressed in metaSMT Python code
- P-OCL Example:

```
Sequence{2..16} -> select(e | e/2 = 0) -> includes(self.allInstances() -> size())
```

- metaSMT Syntax:

```
y >= bv_uint(2)[bw]
y <= bv_uint(16)[bw]
(y - bv_uint(2)[bw]) %
  bv_uint(2)[bw] == bv_uint(0)[bw]
```
Constraint-based Platform Variant Specification

- Platform Variants Specification
- Platform Constraints
- Platform Variant Generation
- Adaption of Virtual Platform Variants
Platform Variant Generation

- Solver solutions are provided as matrix

\[
\begin{pmatrix}
x_0 & x_1 & x_2 & x_3 & \ldots & x_n \\
2 & 3 & 54 & 235 & \ldots & i
\end{pmatrix}
\quad n \in \mathbb{N}; \ i \in \mathbb{N}
\]

- Each variable represents a module, template or parameter specification
Platform Variant Generation

- Variant generation example:
  - Ring-Topology
  - Template $T1$ is specified variable by constraint:
    \[
    \text{Sequence}\{1..3\} \rightarrow \text{includes}(\text{self}.\text{allInstances()} \rightarrow \text{size}())
    \]
    - Whereby $\text{self}$ refers to $T1$
Platform Variant Generation

- Variant generation example:
  - Ring-Topology
  - Constraint is formalized in:
    
    \[
    \begin{align*}
    x_0 & \geq \text{bv\_uint}(1)[\text{bw}] \\
    x_0 & \leq \text{bv\_uint}(3)[\text{bw}]
    \end{align*}
    \]

- Solver solution:
  \[
  \begin{pmatrix}
  x_0 \\
  3 \\
  \vdots
  \end{pmatrix}
  \]
Platform Variant Generation

- Variant generation example:
  - Ring-Topology
  - Constraint is formalized in:
    \[ x_0 \geq \text{bv\_uint}(1)[\text{bw}] \]
    \[ x_0 \leq \text{bv\_uint}(3)[\text{bw}] \]

- Solver solution:
  \[
  \begin{pmatrix}
  x_0 \\
  3 \\
  \end{pmatrix}
  \]
Constraint-based Platform Variant Specification

- Platform Variants Specification
- Platform Constraints
- Platform Variant Generation
- Adaption of Virtual Platform Variants
Configuration of platform variants as virtual prototypes
   - Linking and instantiation of SystemC modules regarding the generated platform variant specification

Dynamical reconfiguration during the simulation without recompilation
Use Cases

- **Media Oriented Systems Transport-Bus (MOST)**
  - Simulation-based verification of implementation against specification:
    - Ring Break Diagnosis (RBD) Application
    - Central Component Application

- **FlexRay**
  - Exploration of a Camera and Recognize Module:
    - Traffic Sign Recognition (TSR)

- **Verification Flow:**
Experimental Results – RBD Verification scenarios

- Six evaluation scenarios are turned out to be suggestive: **Error Free, Ring Break, Excessive Attenuation, Multi Master, All Slave, Combination** [1]

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Variants</th>
<th>Templates</th>
<th>Constraints</th>
</tr>
</thead>
<tbody>
<tr>
<td>Error Free</td>
<td>25133</td>
<td>8</td>
<td>38</td>
</tr>
<tr>
<td>Ring Break</td>
<td>24478</td>
<td>8</td>
<td>37</td>
</tr>
<tr>
<td>Excessive Attenuation</td>
<td>24564</td>
<td>8</td>
<td>37</td>
</tr>
<tr>
<td>Multi Master</td>
<td>25231</td>
<td>8</td>
<td>37</td>
</tr>
<tr>
<td>All Slave</td>
<td>25117</td>
<td>7</td>
<td>29</td>
</tr>
<tr>
<td>Combination</td>
<td>24756</td>
<td>8</td>
<td>38</td>
</tr>
</tbody>
</table>
Top level of the variant specification for scenario „Ring Break“
Each SlaveTemplate contains Or-Template to inject Ring Break Channels
Only one Ring Break can be diagnosed by RBD algorithm
  - Ensured by P-OCL If constraint
Experimental Results –
Ring Break Platform Variant Specification

- Each SlaveTemplate contains Or-Template to inject Ring Break Channels
- Only one Ring Break can be diagnosed by RBD algorithm
  - Ensured by P-OCL If constraint

```plaintext
«vp_constraint»
Constraint5
{if self.orTemp.ChannelRB.active() then
slaveTemp1.orTemp.allInstances->forall(e | !e.ChannelRB.active()) and
slaveTemp2.orTemp.allInstances->forall(e | !e.ChannelRB.active())
else ...
endif
endif}
```
Experimental Results – Central Component Verification Scenarios

- Eight evaluation scenarios turned out to be suggestive.

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Variants</th>
<th>Templates</th>
<th>Constraints</th>
</tr>
</thead>
<tbody>
<tr>
<td>No SSO</td>
<td>15334</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>One timing slave SSO</td>
<td>119271</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>Timing master SSO</td>
<td>1625</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>More than one timing slave reports SSO</td>
<td>746616</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>No CU</td>
<td>20243</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>One timing slave reports CU</td>
<td>4744</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>Timing master CU</td>
<td>1465</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>More than one timing slave reports CU</td>
<td>56294</td>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>
FlexRay – Traffic Sign Recognition (TSR) Scenario

- Heterogeneous system, virtual prototypes and target code
- Virtual prototype modules
- Target code implementation for Tilera board
Experimental Results

- Evaluated against frame rate and recognized traffic signs

- Exploration of the Camera Module regarding:
  - Display resolution
  - Greyscale- or colored-camera
  - Scale factor

- Exploration of different hardware parallelization options:
  - Number of used cores (up to 54 cores)
  - Range definition for circle detection

- 71150 valid Variants are generated
Conclusion

- Constraint- and Model-based variants specification approach
  - High structural flexibility
  - Reuse of already modeled templates and variants
  - Enables to handle huge variants spaces
  - Precise, plausible and comprehensive specification of valid variants

- Automatically generation and simulation of platform variants
Conclusion

- Constraint- and Model-based variants specification approach
  - High structural flexibility
  - Reuse of already modeled templates and variants
  - Enables to handle huge variants spaces
  - Precise, plausible and comprehensive specification of valid variants

- Automatically generation and simulation of platform variants

- Benefits are:
  - Reduction of manual effort in verification, exploration and test
  - Highly automatic generation of virtual prototype variants
  - Reusability of the variants specifications
Thank you for your attention!

Contact person
Andreas Burger
FZI Research Center for Information Technologies
Haid-und-Neu-Str. 10-14
76131 Karlsruhe, Germany
aburger@fzi.de
www.fzi.de/ispe
References