Analyzing Reconfigurable Component-Based Systems Using Attribute Grammars

Thomas Noll

Software Modeling and Verification Group
RWTH Aachen University
noll@cs.rwth-aachen.de

FACS 2011; September 14, 2011; Oslo, Norway
Overview

1. Introduction

2. Specifying Reconfigurable Systems in AADL

3. Attribute Grammars

4. Analyzing AADL Specifications Using Attribute Grammars
Overall objective

Develop a model-based approach to system-software co-engineering while focusing on a coherent set of modeling and analysis techniques for evaluating system-level correctness, safety, dependability, and performance of on-board computer-based aerospace systems.

Derived objectives

2. Verification methodology based on state-of-the-art formal methods
3. Toolset supporting the analysis of SLIM models
4. Evaluation on industrial-size case studies from aerospace domain
# The ESA COMPASS Project

([http://compass.informatik.rwth-aachen.de/](http://compass.informatik.rwth-aachen.de/))

## Overall objective

Develop a model-based approach to system-software co-engineering while focusing on a coherent set of modeling and analysis techniques for evaluating system-level correctness, safety, dependability, and performance of on-board computer-based aerospace systems.

## Derived objectives

1. **Modeling formalism**: System-Level Integrated Modeling Language (SLIM; “extended subset” of AADL)
2. **Verification methodology** based on state-of-the-art formal methods
3. **Toolset** supporting the analysis of SLIM models
4. **Evaluation** on industrial-size case studies from aerospace domain
System = hierarchy of interacting (HW/SW) components

Ports provide interaction interfaces
- event ports: for (multi-way) hand-shaking communication
- data ports: for continuous exchange of values

Connections define interaction topology
- event port connections
- (inter-component) data port connections
- (intra-component) data flows

Mode transition system specifies component nominal behavior
(≈ timed automaton)

Error occurrence and handling

Dynamic reconfiguration: mode-dependent (de-)activation of components and connections
A Reconfigurable Data Acquisition System

Monitor

Sensors
sensor1
sensor2

Filters
filter1
filter2

Switches
switchS
switchF
switch

Output
value

Acquisition

Analyzing Component-Based Reconfigurable Systems
FACS 2011
system Acquisition
    features
        value: out data port real;
    end Acquisition;

system implementation Acquisition.Impl
subcomponents
    sensors: system Sensors;
    filters: system Filters;
    monitor: system Monitor;
connections
    data port sensors.output -> filters.input;
    data port sensors.output -> monitor.valueS;
    data port filters.output -> value;
    data port filters.output -> monitor.valueF;
    event port monitor.switchS -> sensors.switch;
    event port monitor.switchF -> filters.switch;
end Acquisition.Impl;
```
system Sensors
features
  output: out data port real;
  switch: in event port;
end Sensors;

system implementation Sensors.Impl
subcomponents
  sensor1: device Sensor in modes (Primary);
  sensor2: device Sensor in modes (Backup);
connections
  data port sensor1.output -> output in modes (Primary);
  data port sensor2.output -> output in modes (Backup);
modes
  Primary: initial mode;
  Backup: mode;
transitions
  Primary -[switch]->_ Backup;
end Sensors.Impl;
device Sensors.Impl
features
  output: out data port real;
end Sensor;
device implementation Sensor.Impl
...
end Sensor.Impl;
```
Cyclic Data Port Dependencies

- Mode transitions and event port connections yield **configuration transition system**
  - **configuration** = current mode of each active component + data values
- Data port connections/flows induce **equation system over data port values**
- Required: **unique solution** in each system configuration
  \[ \Rightarrow \text{Data port dependencies (as imposed by data port connections/flows) must always be acyclic} \]

**Definition (Circularity of AADL specifications)**

An AADL specification is called **circular** if there exists a mode configuration such that the corresponding data port dependency graph has a (directed) cycle. Otherwise it is called **noncircular**.

**Note:** restriction to reachable mode configuration undecidable
  \[ \Rightarrow \text{Approximation by considering all combinations of modes} \]
Cyclic Data Port Dependencies

- Mode transitions and event port connections yield configuration transition system
  - configuration = current mode of each active component + data values
- Data port connections/flows induce equation system over data port values
- Required: unique solution in each system configuration
  ⇒ Data port dependencies (as imposed by data port connections/flows) must always be acyclic

Definition (Circularity of AADL specifications)

An AADL specification is called circular if there exists a mode configuration such that the corresponding data port dependency graph has a (directed) cycle. Otherwise it is called noncircular.

Note: restriction to reachable mode configuration undecidable

⇒ Approximation by considering all combinations of modes
Cyclic Data Port Dependencies

- Mode transitions and event port connections yield configuration transition system
  - configuration = current mode of each active component + data values
- Data port connections/flows induce equation system over data port values
- Required: unique solution in each system configuration
  \[ \Rightarrow \text{Data port dependencies (as imposed by data port connections/flows) must always be acyclic} \]

Definition (Circularity of AADL specifications)

An AADL specification is called circular if there exists a mode configuration such that the corresponding data port dependency graph has a (directed) cycle. Otherwise it is called noncircular.

Note: restriction to reachable mode configuration undecidable
  \[ \Rightarrow \text{Approximation by considering all combinations of modes} \]
system Cyclic
end Cyclic;

system implementation Cyclic_Impl
subcomponents
  inc1: system Inc;
  inc2: system Inc;
connections
  data port inc1.output -> inc2.input;
  data port inc2.output -> inc1.input;
end Cyclic_Impl;

system Inc
features
  input: in data port int;
  output: out data port int;
end Inc;

system implementation Inc_Impl
flows
  output := input + 1;
end Inc_Impl;
Origininally devised by D. Knuth to define semantics of context-free languages

Idea: enrich context-free grammar by semantic rules which annotate syntax tree with attribute values

Attributes attached to nonterminal symbols
  synthesized: bottom-up computation (from the leaves to the root)
  inherited: top-down computation (from the root to the leaves)

With every production a set of semantic rules is associated
  define values of inner attributes
    (= synthesized/inherited of LHS/RHS) ...
  in dependence of outer attributes
    (= synthesized/inherited of RHS/LHS)
Example (Knuth’s binary numbers)

### Numbers

- **S → L**
  - $v.0 = v.1$
  - $p.1 = 0$

- **S → L.L**
  - $v.0 = v.1 + v.3$
  - $p.1 = 0$
  - $p.3 = -l.3$

### Lists

- **L → B**
  - $v.0 = v.1$
  - $l.0 = 1$
  - $p.1 = p.0$

- **L → LB**
  - $v.0 = v.1 + v.2$
  - $l.0 = l.1 + 1$
  - $p.1 = p.0 + 1$
  - $p.2 = p.0$

### Bits

- **B → 0**
  - $v.0 = 0$

- **B → 1**
  - $v.0 = 2^{p.0}$

**Synthesized attributes of** $S, L, B$: $v$ (value; domain $V_v := \mathbb{Q}$)

**Inherited attribute of** $L$: $l$ (length; domain $V_l := \mathbb{N}$)

**Inherited attribute of** $L, B$: $p$ (position; domain $V_p := \mathbb{Z}$)
Example: Knuth’s Binary Numbers

Example (Knuth’s binary numbers)

| Numbers | S → L | v.0 = v.1 |
|         |      | p.1 = 0 |
|         | S → L.L | v.0 = v.1 + v.3 |
|         |      | p.1 = 0 |
|         |      | p.3 = −l.3 |
| Lists   | L → B | v.0 = v.1 |
|         |      | l.0 = 1 |
|         |      | p.1 = p.0 |
|         | L → LB | v.0 = v.1 + v.2 |
|         |      | l.0 = l.1 + 1 |
|         |      | p.1 = p.0 + 1 |
|         |      | p.2 = p.0 |
| Bits    | B → 0 | v.0 = 0 |
| Bits    | B → 1 | v.0 = 2^{p.0} |

Synthesized attributes of \( S, L, B \): \( v \) (value; domain \( V_v := \mathbb{Q} \))

of \( L \): \( l \) (length; domain \( V_l := \mathbb{N} \))

Inherited attribute of \( L, B \): \( p \) (position; domain \( V_p := \mathbb{Z} \))
An Attributed Derivation Tree

Example (Knuth’s binary numbers)
Again: unique solvability of equation system required
⇒ avoid cyclic dependencies

Definition (Circularity of AGs)
An attribute grammar is called circular if there exists a syntax tree $t$ such that the attribute equation system of $t$ is recursive (i.e., some attribute variable of $t$ depends on itself). Otherwise it is called noncircular.
Again: unique solvability of equation system required
⇒ avoid cyclic dependencies

**Definition (Circularity of AGs)**

An attribute grammar is called **circular** if there exists a syntax tree \( t \) such that the attribute equation system of \( t \) is recursive (i.e., some attribute variable of \( t \) depends on itself). Otherwise it is called **noncircular**.
Decidability of Circularity

**Definition (Attribute dependence)**

Given: AG with underlying CFG $G = \langle N, \Sigma, P, S \rangle$.

- If $t$ is a syntax tree with root label $A \in N$ and root node $k$, $\alpha \in \text{syn}(A)$, and $\beta \in \text{inh}(A)$ such that $\beta.k \rightarrow^+_t \alpha.k$, then $\alpha$ is dependent on $\beta$ below $A$ in $t$ (notation: $\beta \xrightarrow{A} \alpha$)
- For every syntax tree $t$ with root label $A \in N$,
  $$\text{is}(A, t) := \{ (\beta, \alpha) \in \text{inh}(A) \times \text{syn}(A) \mid \beta \xrightarrow{A} \alpha \text{ in } t \}$$
- For every $A \in N$,
  $$\text{IS}(A) := \{ \text{is}(A, t) \mid t \text{ syntax tree with root label } A \} \subseteq 2^{\text{Inh} \times \text{Syn}}$$

**Example (Knuth's binary numbers)**

- $\text{is}(L, L \Rightarrow B \Rightarrow 0) = \emptyset$
- $\text{is}(L, L \Rightarrow B \Rightarrow 1) = \{(p, v)\}$
- $l$ never dependent on any inherited attribute
  $\Rightarrow \text{IS}(L) = \{\emptyset, \{(p, v)\}\}$
Decidability of Circularity

Definition (Attribute dependence)

Given: AG with underlying CFG $G = \langle N, \Sigma, P, S \rangle$.

- If $t$ is a syntax tree with root label $A \in N$ and root node $k$, $\alpha \in \text{syn}(A)$, and $\beta \in \text{inh}(A)$ such that $\beta.k \xrightarrow{+} t \alpha.k$, then $\alpha$ is dependent on $\beta$ below $A$ in $t$ (notation: $\beta \xleftrightarrow{A} \alpha$)
- For every syntax tree $t$ with root label $A \in N$,
  $$\text{is}(A, t) := \{(\beta, \alpha) \in \text{inh}(A) \times \text{syn}(A) | \beta \xleftrightarrow{A} \alpha \text{ in } t\}$$
- For every $A \in N$,
  $$\text{IS}(A) := \{\text{is}(A, t) | t \text{ syntax tree with root label } A\} \subseteq 2^{\text{Inh} \times \text{Syn}}$$

Example (Knuth’s binary numbers)

- $\text{is}(L, L \Rightarrow B \Rightarrow 0) = \emptyset$
- $\text{is}(L, L \Rightarrow B \Rightarrow 1) = \{(p, v)\}$
- $l$ never dependent on any inherited attribute
  $$\Rightarrow \text{IS}(L) = \{\emptyset, \{(p, v)\}\}$$
### The Circularity Test

#### Algorithm (Circularity test)

1. **Iterative computation of IS sets**
2. AG circular iff exists \( \pi = A_0 \rightarrow A_1 \ldots A_n \in P \) and \( D_i \in IS(A_i) \) (\( \forall i \in [n] \)) such that \( D(\pi) \cup \bigcup_{i=1}^{n} D_i \) has a cycle

#### Theorem

The time complexity of the circularity test is **exponential** in the size of the attribute grammar (\( = \) maximal length of right-hand sides of productions).

#### Proof.

The Circularity Test

Algorithm (Circularity test)

1. Iterative computation of $IS$ sets
2. AG circular iff exists $\pi = A_0 \rightarrow A_1 \ldots A_n \in P$ and $D_i \in IS(A_i)$ ($\forall i \in [n]$) such that $D(\pi) \cup \bigcup_{i=1}^{n} D_i$ has a cycle

Theorem

The time complexity of the circularity test is exponential in the size of the attribute grammar (\(=\) maximal length of right-hand sides of productions).

Proof.

### AADL Specifications vs. Attribute Grammars

<table>
<thead>
<tr>
<th>AADL</th>
<th>Attribute grammars</th>
</tr>
</thead>
<tbody>
<tr>
<td>System configuration</td>
<td>Derivation tree</td>
</tr>
<tr>
<td>Mode</td>
<td>Production</td>
</tr>
<tr>
<td>Active component</td>
<td>Nonterminal symbol</td>
</tr>
<tr>
<td>Inactive component</td>
<td>Terminal symbol</td>
</tr>
<tr>
<td>Incoming data port</td>
<td>Inherited attribute</td>
</tr>
<tr>
<td>Outgoing data port</td>
<td>Synthesized attribute</td>
</tr>
<tr>
<td>Flow/data port connection</td>
<td>Semantic rule</td>
</tr>
</tbody>
</table>

#### Example (Data acquisition system)

```
Acquisition

Sensors
- Sensor
  - sensor

Filters
- iFilter
  - filter

Monitor
- vF
- vS
```

---

**Analyzing Component-Based Reconfigurable Systems**

FACS 2011
Definition (Context-free grammar of AADL specification)

\[ G = \langle N, \Sigma, P, S \rangle \] is given as follows:

- \( N := Cmp \)
- \( \Sigma := Cmp^\dagger \) (\( c^\dagger \) denotes inactive component)
- \( P := \{ \pi_{c, m} \mid c \in Cmp, m \in Mod(c) \} \) where
  - \( \pi_{c, m} := c \rightarrow c'_1 \ldots c'_n \) for
  - \( \bigcup_{m \in Mod(c)} Cmp(c, m) = \{ c_1, \ldots, c_n \} \) and
  - \( c'_i := \begin{cases} c_i & \text{if } c_i \in Cmp(c, m) \\ c_i^\dagger & \text{otherwise} \end{cases} \)
- \( S := \text{main} \)
Example

Acquisition $\rightarrow$ Sensors Filters Monitor

Sensors $\rightarrow$ Sensor sensor
Sensors $\rightarrow$ sensor Sensor
Sensor $\rightarrow$ $\varepsilon$

Filters $\rightarrow$ Filter filter
Filters $\rightarrow$ filter Filter
Filter $\rightarrow$ $\varepsilon$
Monitor $\rightarrow$ $\varepsilon$
Definition (Attribution scheme of AADL specification)

- $Inh := \bigcup_{c \in Cmp} IPrt(c)$
- $Syn := \bigcup_{c \in Cmp} OPrt(c)$
- For every $c \in Cmp$:
  - $inh(c) := IPrt(c)$
  - $syn(c) := OPrt(c)$
- For every $m \in Mod(c)$:

$$E_\pi := \{ q.j = p.i \mid (p.i, q.j) \in Con(c, m) \} \cup \{ q.0 = e[p \mapsto p.0; p \in IPrt(c)] \mid (e, q) \in Flw(c, m) \}.$$
### Example

| Acquisition → Sensors Filters Monitor : | input.2 = output.1  
valueS.3 = output.1  
valueF.3 = output.2  
value.0 = output.2 |
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Sensors → Sensor sensor :</td>
</tr>
<tr>
<td>Sensors → sensor Sensor :</td>
</tr>
<tr>
<td>Sensor → ε</td>
</tr>
</tbody>
</table>
| Filters → Filter filter : | input.1 = input.0  
output.0 = output.1 |
| Filters → filter Filter : | input.2 = input.0  
output.0 = output.2 |
| Filter → ε : | output.0 = 2.0 * input.0 |
| Monitor → ε | |
An AADL specification is circular iff the corresponding attribute grammar is circular.
Problem: exponential complexity of circularity test
Solution: switch to strong noncircularity

Definition (Attribute dependence (modified))

Given: AG with underlying CFG \( G = \langle N, \Sigma, P, S \rangle \).

- Reminder: if \( t \) is a syntax tree with root label \( A \in N \) and root node \( k \), \( \alpha \in \text{syn}(A) \), and \( \beta \in \text{inh}(A) \) such that \( \beta.k \xrightarrow{+} t \alpha.k \), then \( \alpha \) is dependent on \( \beta \) below \( A \) in \( t \) (notation: \( \beta \xleftarrow{A} \alpha \)).
- For every \( A \in N \),
  \[
  IS'(A) := \{(\beta, \alpha) \mid \beta \xleftarrow{A} \alpha \text{ in some syntax tree with root label } A\}
  \subseteq Inh \times Syn
  \]

- Implemented in COMPASS Toolset
A (weakly) noncircular, but not strongly noncircular AADL specification:

```aadl
system Super
  features
    value: out data port real;
end Super;

system implementation Super.Impl
  subcomponents
    sub1, sub2: system Sub;
  connections
    data port sub1.out1 -> sub2.in2, sub1.out2 -> sub2.in1;
    data port sub2.out1 -> sub1.in1, sub2.out2 -> sub1.in2;
end Super.Impl;

system Sub
  features
    in1, in2: in data port int;
    out1, out2: out data port int;
end Sub;

system implementation Sub.Impl
  flows
    out1 := in2 in modes (m0); out1 := 1 in modes (m1);
    out2 := 2 in modes (m0); out2 := in1 in modes (m1);
  modes
    m0: initial mode;
    m1: mode;
  transitions ...
end Sub.Impl;
```
A Strong Cycle

Super

in1 in2 Sub out1 out2

in1 in2 Sub out1 out2