Analyzing Reconfigurable Systems Using Attribute Grammars

Thomas Noll

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

MOVES Seminar; February 17, 2011
Overview

1. Introduction

2. Specifying Reconfigurable Systems in AADL

3. Attribute Grammars

4. Analyzing AADL Specifications Using Attribute Grammars
1. Introduction

2. Specifying Reconfigurable Systems in AADL

3. Attribute Grammars

4. Analyzing AADL Specifications Using Attribute Grammars
Main 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
Main 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 Architecture Analysis and Design Language

1998  SAE AS-2C

2004  AADL 1.0

2006  Error Annex 1.0

2009  AADL 2.0

2010  Error Annex 2.0
AADL’s Main Features

- 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
1. Introduction

2. Specifying Reconfigurable Systems in AADL

3. Attribute Grammars

4. Analyzing AADL Specifications Using Attribute Grammars
A Reconfigurable Data Acquisition System

Acquisition

Monitor

Sensors

sensor1

sensor2

Filters

filter1

filter2

switch

switchS

switchF

output

output

value

MOVES Seminar 8/32
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 Sensor
features
  output: out data port real;
end Sensor;
device implementation Sensor.Impl
...
end Sensor.Impl;
Cyclic Data Port Dependencies

- Data port connections/flows induce **equation system over data port values**
- Required: **unique solution** in each system configuration
  
  (= components \(\mapsto\) modes)
- \(\implies\) Data port dependencies (as imposed by data port connections/flows) must 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

\(\implies\) Approximation by considering all combinations of modes
Cyclic Data Port Dependencies

- Data port connections/flows induce equation system over data port values
- Required: unique solution in each system configuration (= components $\mapsto$ modes)
- Data port dependencies (as imposed by data port connections/flows) must 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 $\implies$ Approximation by considering all combinations of modes
Cyclic Data Port Dependencies

- Data port connections/flows induce equation system over data port values
- Required: unique solution in each system configuration (= components ↦→ modes)
- Data port dependencies (as imposed by data port connections/flows) must 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
A Simple Circular Specification

```plaintext
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;
```

Analyzing Reconfigurable Systems
Formalizing Data Port Dependencies

- **Components** \( Cmp \) with main component \( \text{main} \in Cmp \)
- For each \( c \in Cmp \):
  - Modes \( \text{Mod}(c) \)
  - Data ports: \( \text{Prt}(c) := \text{IPrt}(c) \uplus \text{OPrt}(c) \)
    - incoming: \( \text{IPrt}(c) \)
    - outgoing: \( \text{OPrt}(c) \)
- For each \( m \in \text{Mod}(c) \): activities determined by in modes clauses
  - active subcomponents \( \text{Cmp}(c, m) \subseteq Cmp \)
  - active flows:
    \[
    \text{Flw}(c, m) \subseteq \{(e, p) \mid e \text{ expression over } \text{IPrt}(c), p \in \text{OPrt}(c)\}
    \]
- active data port connections for \( \text{Cmp}(c, m) = \{c_1, \ldots, c_n\} \):
  \[
  \text{Con}(c, m) \subseteq \{(p.i, q.j) \mid i = 0, j \in [n], p \in \text{IPrt}(c), q \in \text{IPrt}(c_j), \text{ or }
  i, j \in [n], p \in \text{OPrt}(c_i), q \in \text{IPrt}(c_j), \text{ or }
  i \in [n], j = 0, p \in \text{OPrt}(c_i), q \in \text{OPrt}(c)\}.
  \]
Example (Formalization of data acquisition system)

- \( Cmp = \{\text{Acquisition, Sensors, Sensor, Filters, Filter, Monitor}\} \)
- \( \text{main} = \text{Acquisition} \)
- \( Mod(\text{Acquisition}) = \{\text{default}\} \)
  \( Cmp(\text{Acquisition, default}) = \{\text{Sensors, Filters, Monitor}\} \)
  \( Mod(\text{Sensors}) = \{\text{Primary, Backup}\} \)
  \( Cmp(\text{Sensors, Primary}) = \{\text{Sensor}\} \)
  \( Cmp(\text{Sensors, Backup}) = \{\text{Sensor}\} \) 
  : 

- \( IPrt(\text{Acquisition}) = \emptyset \)
- \( OPrt(\text{Acquisition}) = \{\text{value}\} \)
  \( IPrt(\text{Filters}) = \{\text{input}\} \)
  \( OPrt(\text{Filters}) = \{\text{output}\} \) 
  : 

The Data Acquisition System Revisited I
Example (continued)

- \( \text{Con}(\text{Acquisition, default}) = \{(\text{output.1, input.2}), \) 
  \( (\text{output.1, valueS.3}), \) 
  \( (\text{output.2, value.0}), \) 
  \( (\text{output.2, valueF.3})\} \)

- \( \text{Con}(\text{Sensors, Primary}) = \{(\text{output.1, output.0})\} \)

- \( \text{Con}(\text{Sensors, Backup}) = \{(\text{output.2, output.0})\} \)

- \( \text{Flw}(\text{Acquisition, default}) = \emptyset \)

- \( \text{Flw}(\text{Filter, default}) = \{(2.0 * \text{input, output})\} \)
1. Introduction

2. Specifying Reconfigurable Systems in AADL

3. Attribute Grammars

4. Analyzing AADL Specifications Using Attribute Grammars
Origininallly 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 (and terminal) 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

#### Example (Knuth’s binary numbers)

<table>
<thead>
<tr>
<th>Type</th>
<th>Rule</th>
<th>Equations</th>
</tr>
</thead>
</table>
| Numbers    | $S \rightarrow L$ | $v.0 = v.1$  
                         $p.1 = 0$  
                         $S \rightarrow L.L$  
                         $v.0 = v.1 + v.3$  
                         $p.1 = 0$  
                         $p.3 = -l.3$ |
| Lists      | $L \rightarrow B$ | $v.0 = v.1$  
                         $l.0 = 1$  
                         $p.1 = p.0$  
                         $L \rightarrow LB$  
                         $v.0 = v.1 + v.2$  
                         $l.0 = l.1 + 1$  
                         $p.1 = p.0 + 1$  
                         $p.2 = p.0$ |
| Bits       | $B \rightarrow 0$ | $v.0 = 0$ |
| Bits       | $B \rightarrow 1$ | $v.0 = 2p.0$ |

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

Inherited attributes 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)

<table>
<thead>
<tr>
<th>Numbers</th>
<th>$S \to L$</th>
<th>$v.0 = v.1$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$p.1 = 0$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$S \to L.L$</td>
<td>$v.0 = v.1 + v.3$</td>
</tr>
<tr>
<td></td>
<td>$p.1 = 0$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$p.3 = -l.3$</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Lists</th>
<th>$L \to B$</th>
<th>$v.0 = v.1$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$l.0 = 1$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$p.1 = p.0$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$L \to LB$</td>
<td>$v.0 = v.1 + v.2$</td>
</tr>
<tr>
<td></td>
<td>$l.0 = l.1 + 1$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$p.1 = p.0 + 1$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$p.2 = p.0$</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Bits</th>
<th>$B \to 0$</th>
<th>$v.0 = 0$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$B \to 1$</td>
<td>$v.0 = 2^{p.0}$</td>
</tr>
</tbody>
</table>

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 $V_p := \mathbb{Z}$)
An Attributed Derivation Tree

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

⇒ avoid cyclic dependencies
Again: unique solvability of equation system required

\[ \Rightarrow \] 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 $\beta.k \mathbin{\rightarrow^+_t} \alpha.k$, then $\alpha$ is dependent on $\beta$ below $A$ in $t$ (notation: $\beta \overset{A}{\rightarrow} \alpha$)

- For every syntax tree $t$ with root label $A \in N$,
  
  $$\text{is}(A, t) := \{(\beta, \alpha) \in \text{inh}(A) \times \text{syn}(A) \mid \beta \overset{A}{\rightarrow} \alpha \text{ in } t\}$$

- 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
- $\implies \text{IS}(L) = \{\emptyset, (p, v)\}$
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 $\beta.k \rightarrow^+_t \alpha.k$, then $\alpha$ is dependent on $\beta$ below $A$ in $t$ (notation: $\beta \rightarrow^A \alpha$)

- For every syntax tree $t$ with root label $A \in N$, $\text{is}(A, t) := \{(\beta, \alpha) \in \text{inh}(A) \times \text{syn}(A) | \beta \rightarrow^A \alpha \text{ in } t\}$

- 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

$\implies \text{IS}(L) = \{\emptyset, (p, v)\}$
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.

### 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.

1. Introduction

2. Specifying Reconfigurable Systems in AADL

3. Attribute Grammars

4. Analyzing AADL Specifications Using Attribute Grammars
## AADL 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>Active component</td>
<td>Nonterminal symbol</td>
</tr>
<tr>
<td>Inactive component</td>
<td>Terminal symbol</td>
</tr>
<tr>
<td>Mode</td>
<td>Production</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>
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 $\rightarrow$ $\epsilon$

Filters $\rightarrow$ Filter Filter†

Filters $\rightarrow$ Filter† Filter

Filter $\rightarrow$ $\epsilon$

Monitor $\rightarrow$ $\epsilon$
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:**
- $\text{input}.2 = \text{output}.1$
- $\text{valueS}.3 = \text{output}.1$
- $\text{value}.0 = \text{output}.2$
- $\text{valueF}.3 = \text{output}.2$

** Sensors → Sensor Sensor$^\dagger$:**
- $\text{output}.0 = \text{output}.1$

** Sensors → Sensor$^\dagger$ Sensor:**
- $\text{output}.0 = \text{output}.2$

** Sensor → $\varepsilon$:**

** Filters → Filter Filter$^\dagger$:**
- $\text{input}.1 = \text{input}.0$
- $\text{output}.0 = \text{output}.1$

** Filters → Filter$^\dagger$ Filter:**
- $\text{input}.2 = \text{input}.0$
- $\text{output}.0 = \text{output}.2$
- $\text{output}.0 = 2.0 \times \text{input}.0$

** Filter → $\varepsilon$:**

** Monitor → $\varepsilon$:**
An AADL specification is circular iff the corresponding attribute grammar is circular.
Implementation in the COMPASS Toolset

- 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 $\beta.k \rightarrow^+_t \alpha.k$, then $\alpha$ is dependent on $\beta$ below $A$ in $t$ (notation: $\beta \xrightarrow{A} \alpha$).

- For every $A \in N$,

$$IS'(A) := \{(\beta, \alpha) \mid \beta \xrightarrow{A} \alpha \text{ in some syntax tree with root label } A\} \subseteq \text{Inh} \times \text{Syn}$$

- Implemented in COMPASS Toolset
Strong vs. Weak Noncircularity

A (weakly) noncircular, but not strongly noncircular AADL specification:

```plaintext
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

Analyzing Reconfigurable Systems