



A Generative Approach to the Development of Autonomous Robot Software

Philipp A. Baer, Roland Reichle,
Michael Zapf, Thomas Weise, Kurt Geihs
<lastname>@vs.uni-kassel.de

Distributed Systems Group, Kassel University, Germany

Short Introduction

Development framework for
autonomous robot software

- ◆ Main target: mobile autonomous robots
 - ◆ Message-oriented communication
 - ◆ Unreliable communication media
- ◆ Heterogeneous environments
 - ◆ Operating Systems (Windows, Linux, MacOS, ...)
 - ◆ Hardware Platforms (Intel, PowerPC, ...)
 - ◆ Software Platforms (C++, Java, .NET, ...)



Overview

- I. Case Study: Mobile Robots
- II. Model-Driven Development
- III. Our Approach: Spica
- IV. Lessons Learned
- V. Outlook
- VI. Summary

I. Case Study: Mobile Robots

- ◆ Autonomous mobile robots
 - ◆ Resemble reactive distributed systems
 - ◆ Heterogeneous hardware/software architectures

- ◆ Modular Robot Software
 - ◆ Modules are processes
 - ◆ Modules communicate with each other

- ◆ Unreliable Communication Media
 - ◆ Communication may fail
 - ◆ Thus must be regarded optional

II. Model-Driven Software Development

- ◆ Model-Driven Development (MDD) is here to stay
 - ◆ Modeling is widely-used and accepted
 - ◆ Generalized, proven technique: Model transformation

- ◆ Goals for Software Development
 - ◆ Simplification
 - ◆ Generalization with reusability
 - ◆ Code generation

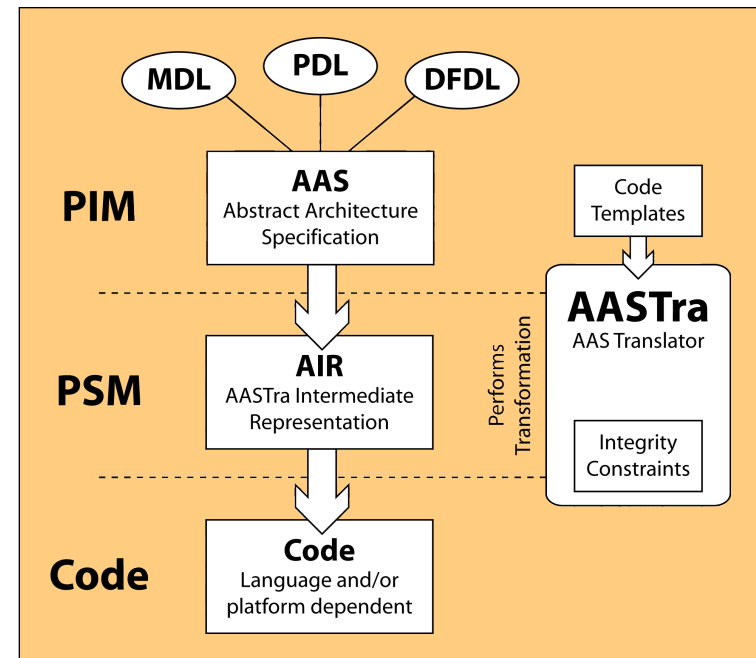
- ◆ Not without controversy
 - ◆ Model Complexity
 - ◆ Correctness (Model/Assumption/Transformation)

III. Spica Approach

- ◆ Abstract Architecture Specification (AAS)
 - ◆ Domain-specific Modeling Languages
 - ◆ Extensible

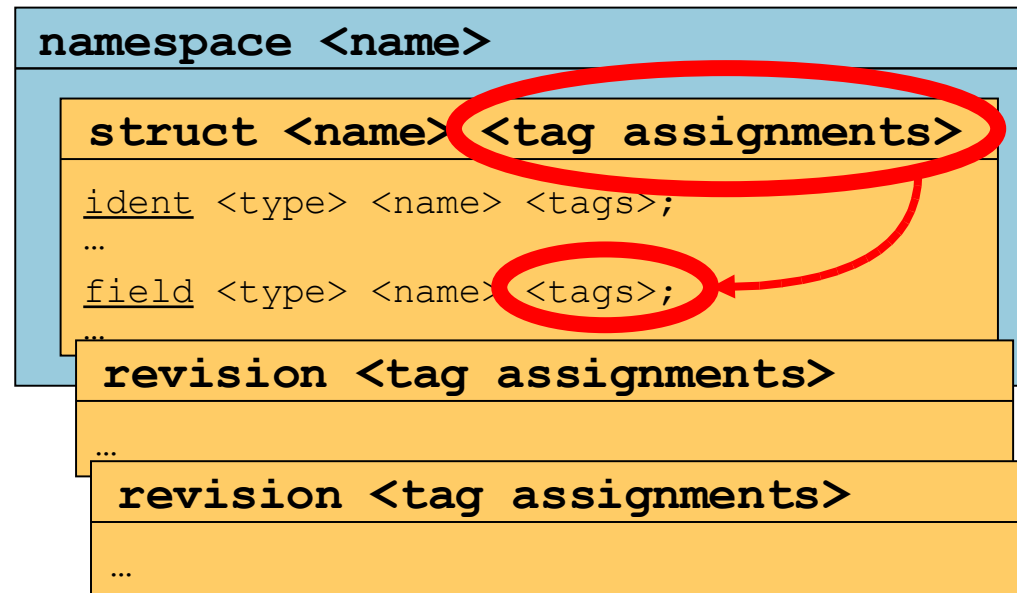
- ◆ Transformation Engine (AASTra)
 - ◆ Model transformation
 - ◆ Code transformation
 - ◆ AIR: in-memory parse-tree

- ◆ Arbitrary code targets
 - ◆ Template-based
 - ◆ Flexible generation rules



III. Message Structures

- ◆ Message Definition Language (MDL)
- ◆ Similar to ASN.1
 - ◆ Domain-specific subset
 - ◆ Different field types
- ◆ Tags
 - ◆ Tags reference fields
 - ◆ Pre-initialize fields
- ◆ Versioning support



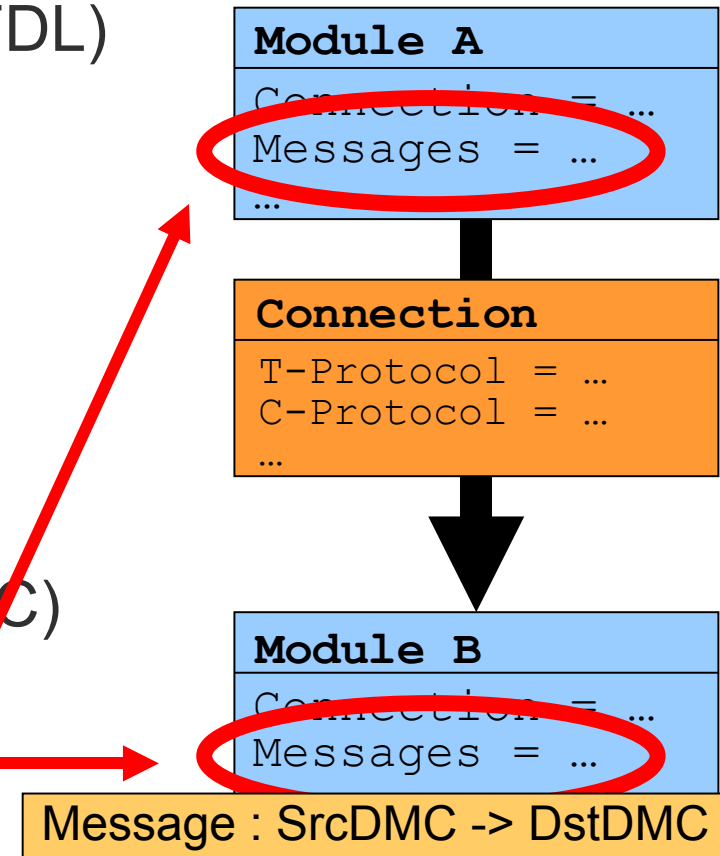
III. Communication Protocols

- ◆ Protocol Definition Language (PDL)
- ◆ Custom protocols
 - ◆ State-machine-based
 - ◆ Use standard transport protocols
- ◆ Logical processing
 - ◆ AND associations: group logically
 - ◆ OR associations: alternatives
- ◆ Extensible through class library-like mechanism
 - ◆ Required for every target language

```
sign(m), send(m, dest) |  
Error("Unable to sign!");
```

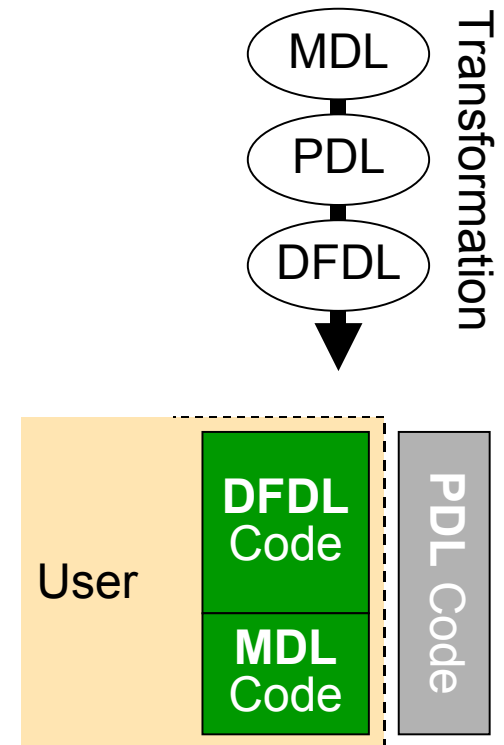
III. Data-Flow

- ◆ Data-Flow Definition Language (DFDL)
- ◆ Communication between Modules
 - ◆ Event-based communication
 - ◆ Connection-specific protocols
 - ◆ Message-specific data management
- ◆ Data Management Containers (DMC)
 - ◆ Message containers
 - ◆ Different semantics available



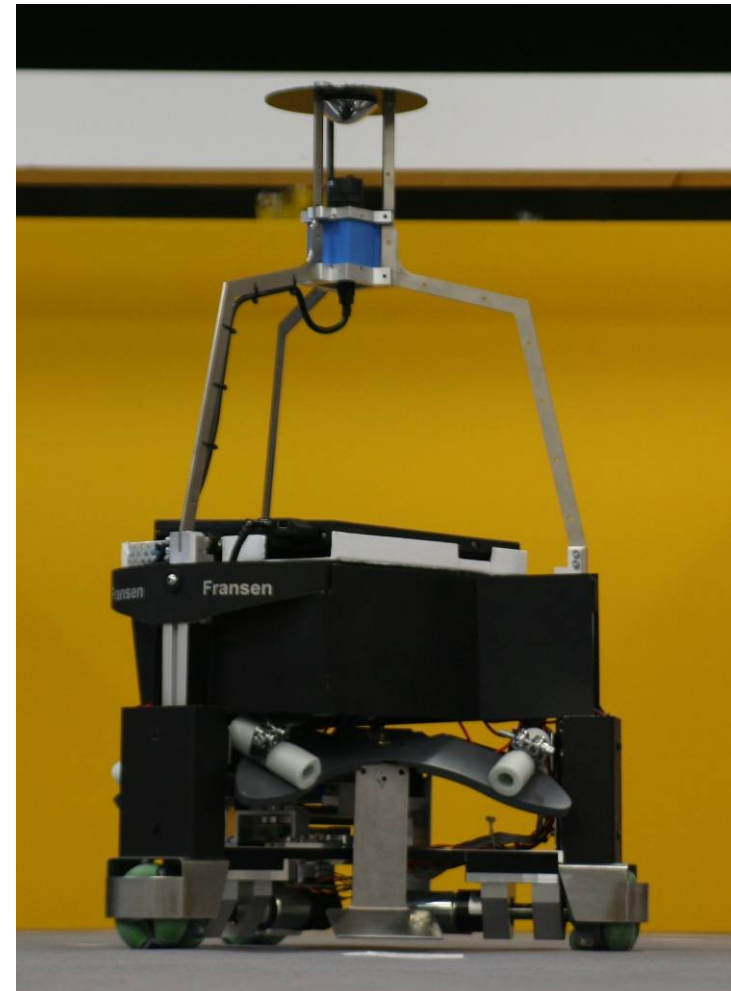
III. Spica User View

- ◆ Transformation tool AASTra creates ready-to-run code
 - ◆ User only accesses DFDL and MDL code
 - ◆ PDL code is not directly accessible
- ◆ DFDL code: communication endpoints
 - ◆ Send operations
 - ◆ Receive callbacks
- ◆ MDL code: message structures



III. Putting it all together: Carpe Noctem

- ◆ Carpe Noctem Robot Soccer
 - ◆ Autonomous mobile systems
 - ◆ Modular software architecture
- ◆ Heterogeneous Platforms
 - ◆ Hardware (Intel, PowerPC)
 - ◆ Software (C#, Java, C++)
- ◆ Entirely based on Spica-Models
 - ◆ 46 Message types
 - ◆ 18 Module models
 - ◆ 54 Connections
 - ◆ Currently no PDL protocols



IV. Lessons Learned

- ◆ AAS languages are specialized subsets/combinations
 - ◆ MDL = subset of ASN.1
 - ◆ PDL = subset of SDL + PL

- ◆ Drawbacks
 - ◆ Early development/research phase
 - ◆ Writing Templates is complex
 - ◆ No accepted standard

- ◆ Advantages
 - ◆ Meets the needs of autonomous mobile systems
 - ◆ Flexible transformation tool
... with customizable in-/output

V. Outlook

- ◆ Extension of AAS languages
 - ◆ MDL: cryptographically secure fields
 - ◆ PDL: integrate capabilities of other languages/techniques
 - ◆ DFDL: dynamic binding (service discovery)

- ◆ Secure dynamic binding is regarded most important

- ◆ Graphical modeling for DFDL (UML, ...)
 - ◆ Not necessary for MDL?
 - ◆ Not suited for PDL?

- ◆ More convenient template creation

VI. Summary

- ◆ Spica: A development framework for autonomous robot systems
 - ◆ Based on the MDD-approach
 - ◆ Domain-specific modelling languages (MDL, PDL, DFDL)
 - ◆ Customizable target languages

- ◆ Spica integration for Soccer Robots
 - ◆ Event-based interaction
 - ◆ Modular software infrastructure



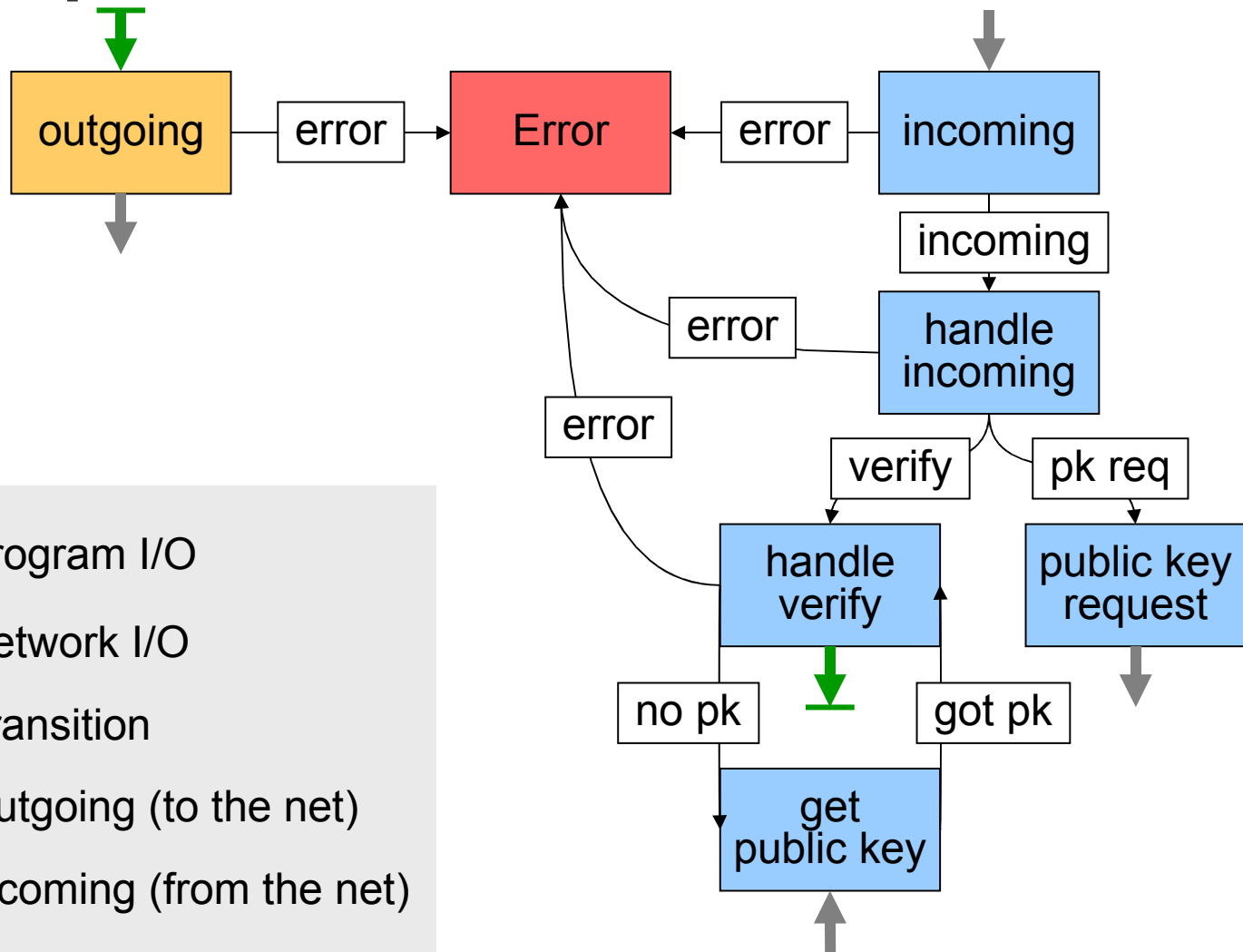
Thank you!

Philipp A. Baer <baer@vs.uni-kassel.de>

References

1. I. T. Union. Information Technology—Abstract Syntax Notation One (ASN.1): Specification of Basic Notation. ITU-T Recommendation X.680, July 2002.
3. J. Ellsberger, D. Hogrefe, A. Sarma. SDL : formal object-oriented language for communicating systems, Prentice Hall, 1997.
5. Carpe Noctem Robocup Team Website. <http://carpenoctem.das-lab.net/>.
7. Object Management Group UML Website. <http://www.uml.org/>.
9. ANTLR Parser Generator. <http://antlr.org/>.
11. StringTemplate Template Engine. <http://www.stringtemplate.org/>.

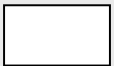
III. Sample Protocol



Program I/O



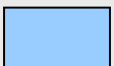
Network I/O



Transition



Outgoing (to the net)



Incoming (from the net)

III. MDL Transformation Example 1

◆ MDL model:

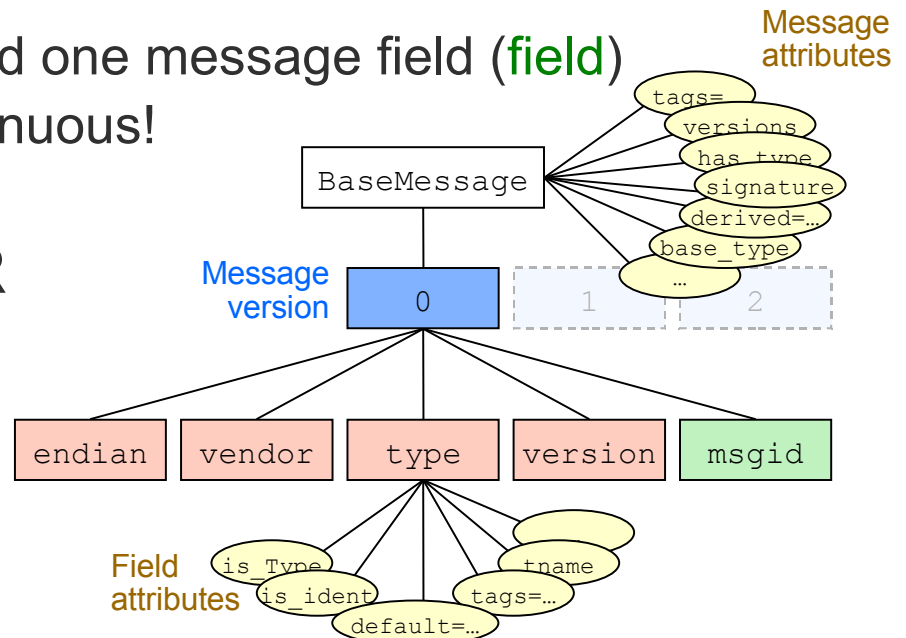
```

struct BaseMessage [Vendor=0,Type=0,Version=0] {
    ident uint8 endian [Endian];
    ident uint8 vendor [Vendor];
    ident uint8 type [Type];
    ident uint8 version [Version];
    field uint32 msgid;
}
    
```

- ◆ Defines a header (**ident**) and one message field (**field**)
- ◆ Header fields must be continuous!

2. AASTra generates the AIR

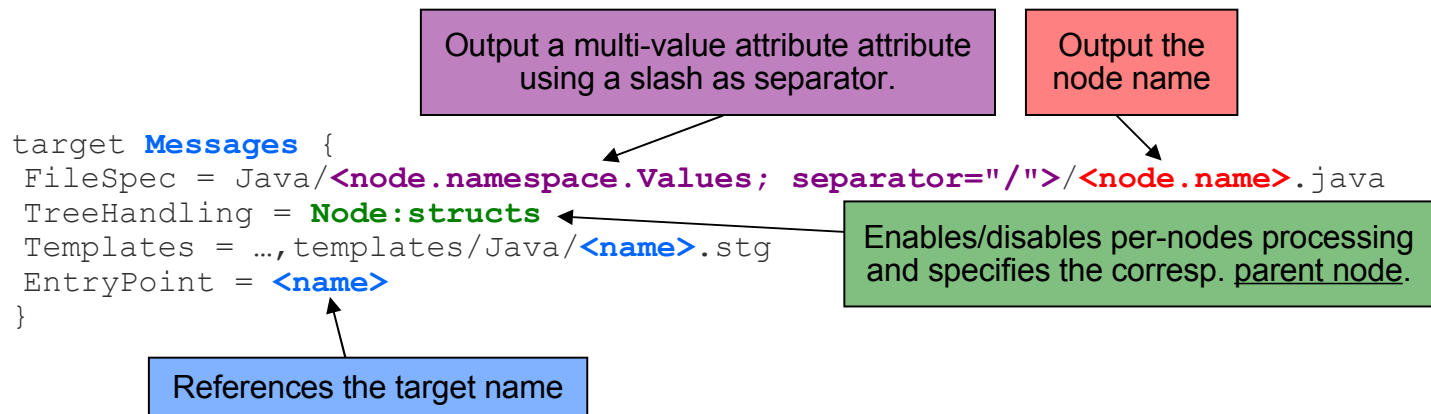
- ◆ Step 1.1: parsing
- ◆ Step 1.2: model checking
- ◆ Step 1.3: cross-referencing



III. MDL Transformation Example 2

1. AASTra then triggers file generation

- ◆ AIR tree processing: whole tree at once or single nodes

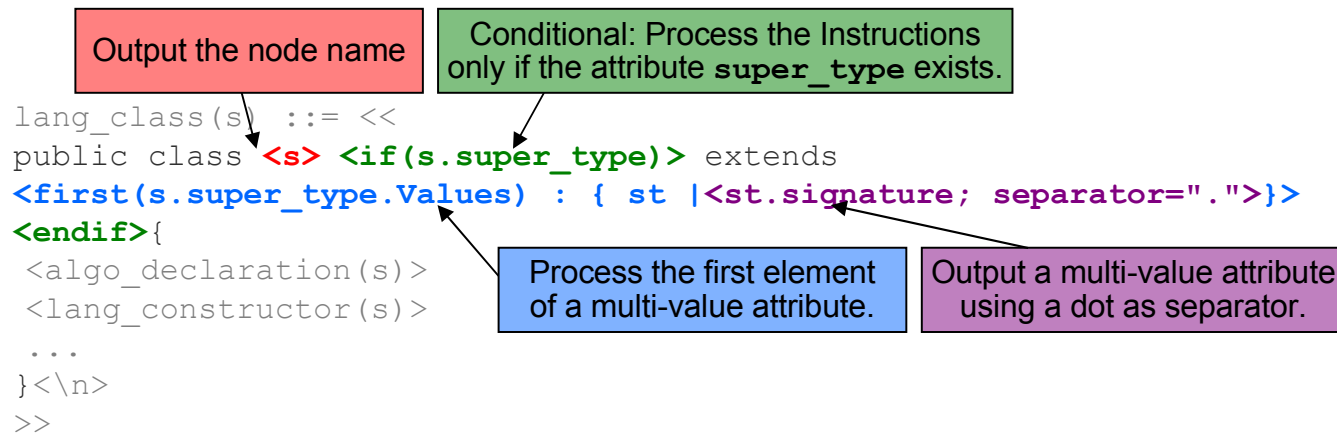


- ◆ FileSpec Path to the target file to generate
- ◆ TreeHandling Specifies how the AIR tree is handled
- ◆ Templates A comma-separated list of template files
- ◆ EntryPoint Name of the main template file

III. MDL Transformation Example 3

1. Code generation

- ◆ Templates access AIR elements



III. MDL Transformation Example 3

- ◆ Code generation
 - ◆ Code skeleton of the generated BaseMessage:

```
...
public class BaseMessage {
    public BaseMessage() {}
    public short getMessageType() { ... }
    public boolean getModified() { ... }
    ...
    public short getType() { ... }
    public void setType(short value) { ... }
    ...
    public void encode(EncodeCNER encoder) throws IOException,Exception { ... }
    public void decode(DecodeCNER decoder) throws IOException, Exception { ... }
    ...
}
```

- ◆ Generated implementation is either
 - ◆ inherited
 - ◆ or instantiated