



Software
Systems
Engineering

MontiCore

A Framework for DSL-Development

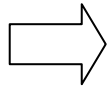
Efficient Editor Generation for
Compositional DSLs in Eclipse



Holger Krahn, Bernhard Rumpe, Steven Völkel
Software Systems Engineering
Technische Universität Braunschweig

<http://www.sse-tubs.de/>

Efficient Editor Generation for Compositional DSLs in Eclipse



1.

MontiCore

2.

Example: MSCs with Java

3.

Editor generation

4.

Live-Demo

5.

Conclusion

MontiCore – Design goals

3

vor/03/01

- **Compact grammar-based definition** of a language in a unified format for **abstract and concrete syntax**
 - Avoids inconsistencies
 - Reduces effort

- **Generative development**
 - Easily accessible, **strongly typed, heterogeneous** data structures
 - Generation of Java components with published interfaces and combination by configuration

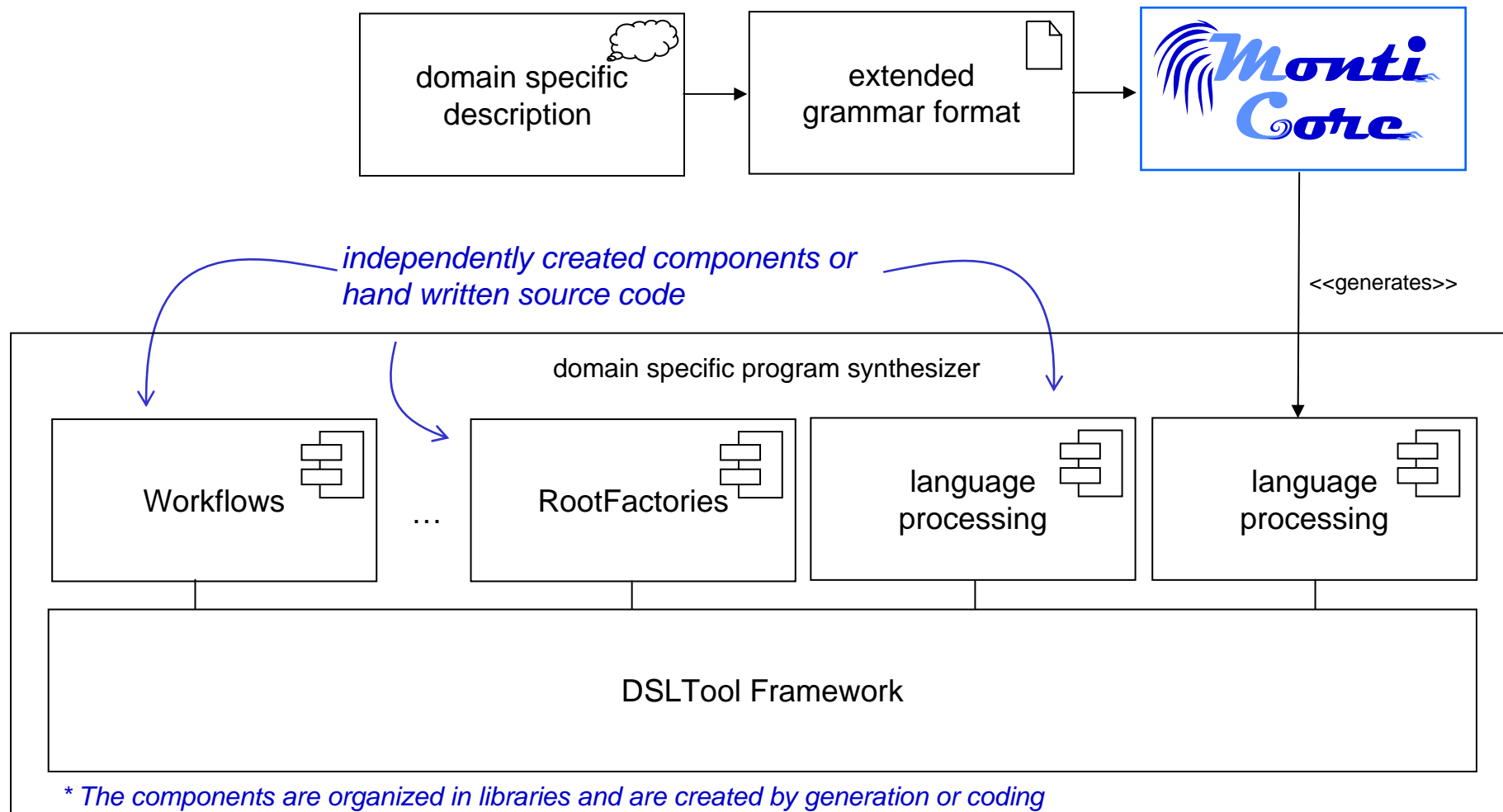
- **Open solution**
 - **Programming of additional components**

- **Framework-based solution**
 - **Structure of processing** is predetermined for easier development of program synthesizers

- **Independency**
 - Available as command line tool, Eclipse-Plugin, Online-Service

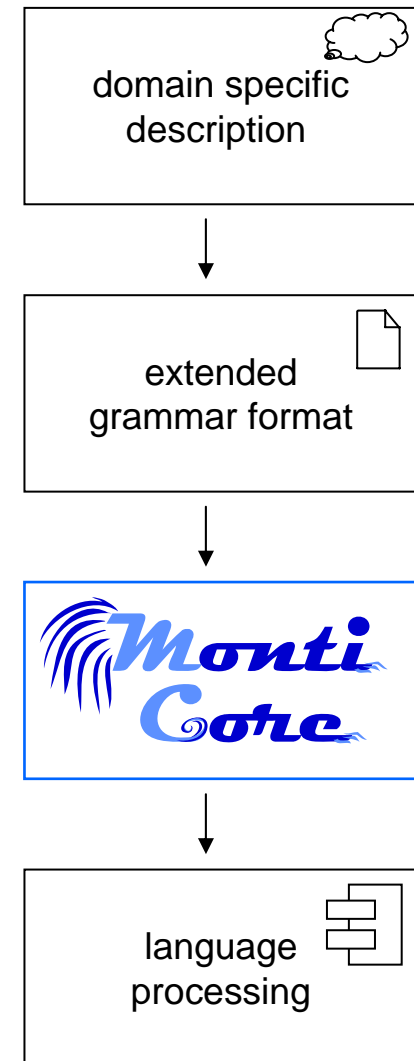
MontiCore – Creation of domain specific program synthesizers (DSLTools)

- MontiCore can be used to create domain specific program synthesizers



MontiCore – Generation of components for language processing

- The generated components are e.g.
 - Recursive-descent LL-Parser
 - Strongly typed AST classes with optional associations
 - Simple Symboltables
 - [Editor for Eclipse](#) (Syntaxhighlighting, Outline, etc.)
 - Documentation (e.g., class diagram for AST, Grammar in EBNF)
- Languages realized with MontiCore:
 - UML/P (CD, SD, OD, SC)
 - Finite Automata
 - Java 5
 - ...



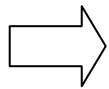
Reuse of languages

6

vor/03/01

- MontiCore supports two types of reuse of languages
 - **Embedding**
 - Specifying grammars with **holes** (grammar fragments)
 - Combine multiple grammars to form new language **without re-compilation**
 - Mechanism suitable to **embed** (possibly multiple) action languages in models
 - **Inheritance**
 - Inherit from an existing grammar
 - Add new rules or change existing ones
 - Extend an existing language by specifying the delta only

Efficient Editor Generation for Compositional DSLs in Eclipse



1.

MontiCore

2.

Example: MSCs with Java

3.

Editor generation

4.

Live-Demo

5.

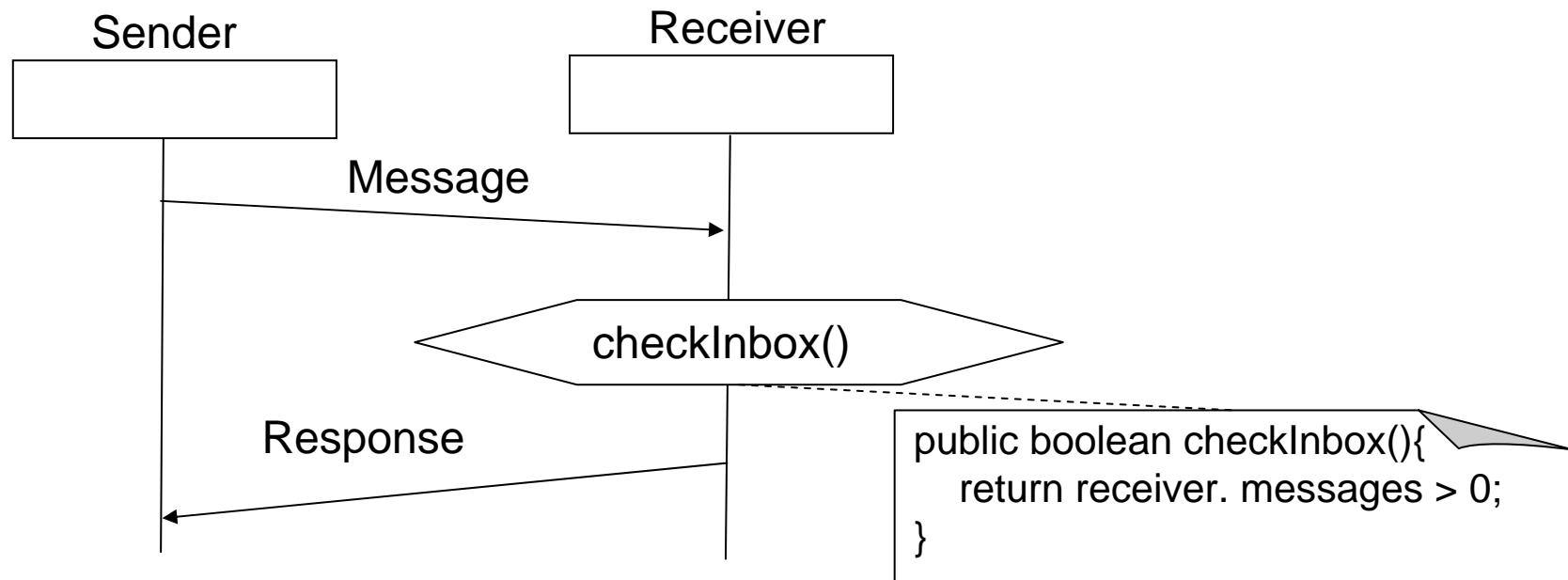
Conclusion

Overview: Example

8

vor/03/01

- Message Sequence Charts
 - Conditions are expressions
 - Methods help to express complex conditions
- Textual Concrete Syntax from ITU-TS Recommendation Z.120



Example: MSCs with Java Definition as MontiCore grammar

```
grammar MSC
options{ compilationunit MSC }

MSC = "msc" name:IDENT "{" ( Instance | Method )* "}";

Instance = "instance" name:IDENT "{" Event* "}";

interface Event;
SendEvent implements Event =
  "out" message:IDENT "to" receiver:IDENT ";";

ReceiveEvent implements Event =
  "in" message:IDENT "from" sender:IDENT ";";

Condition implements Event =
  "condition" name:IDENT
  ( shared:["shared"]
  ( sharedWithAll:["all"] |
  sharedWith:IDENT ("," sharedWith:IDENT)* ) )?
  ( "{" Cond}" | ";" );

external Cond;
external Method;
// ...
}
```

*Event =
SendEvent
| ReceiveEvent
| Condition*

"Holes" in grammar fragment

Example: MSCs with Java

Abstract syntax

10

vor/03/01

```

grammar MSC
  options{ compilationunit MSC }

  MSC = "msc" name:IDENT "{" ( Instance | Method )* "}";

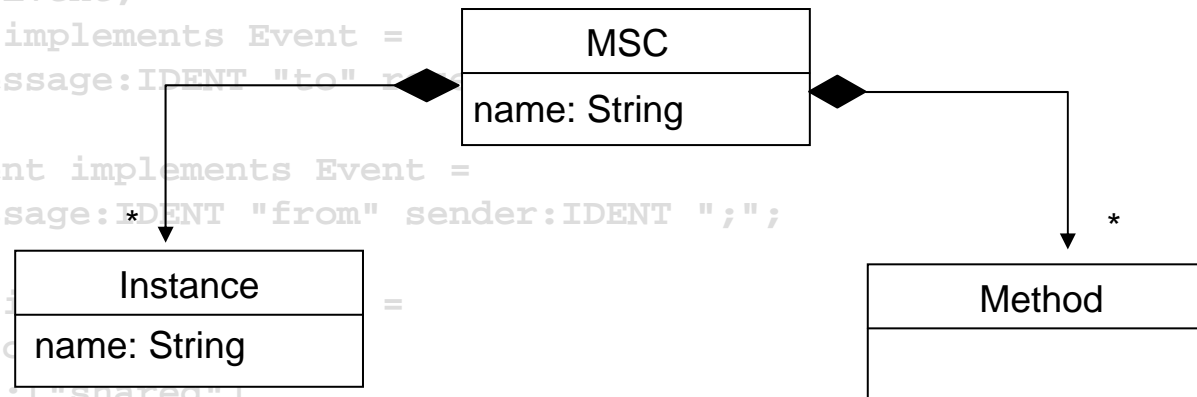
  Instance = "instance" name:IDENT "{" Event* "}";

  interface Event;
  SendEvent implements Event =
    "out" message:IDENT "to" r
  ReceiveEvent implements Event =
    "in" message:IDENT "from" sender:IDENT ";";

  Condition :
    "condition" name:IDENT =
    ( shared:["shared"]
    ( sharedWithAll:["all"] |
      sharedWith:IDENT ("," sharedWith:IDENT)* ) )?
    ( "{" Cond "}" | ";" );

  external Cond;
  external Method;
  // ...
}

```



Example: MSCs with Java

Definition of associations

11

vor/03/01

```

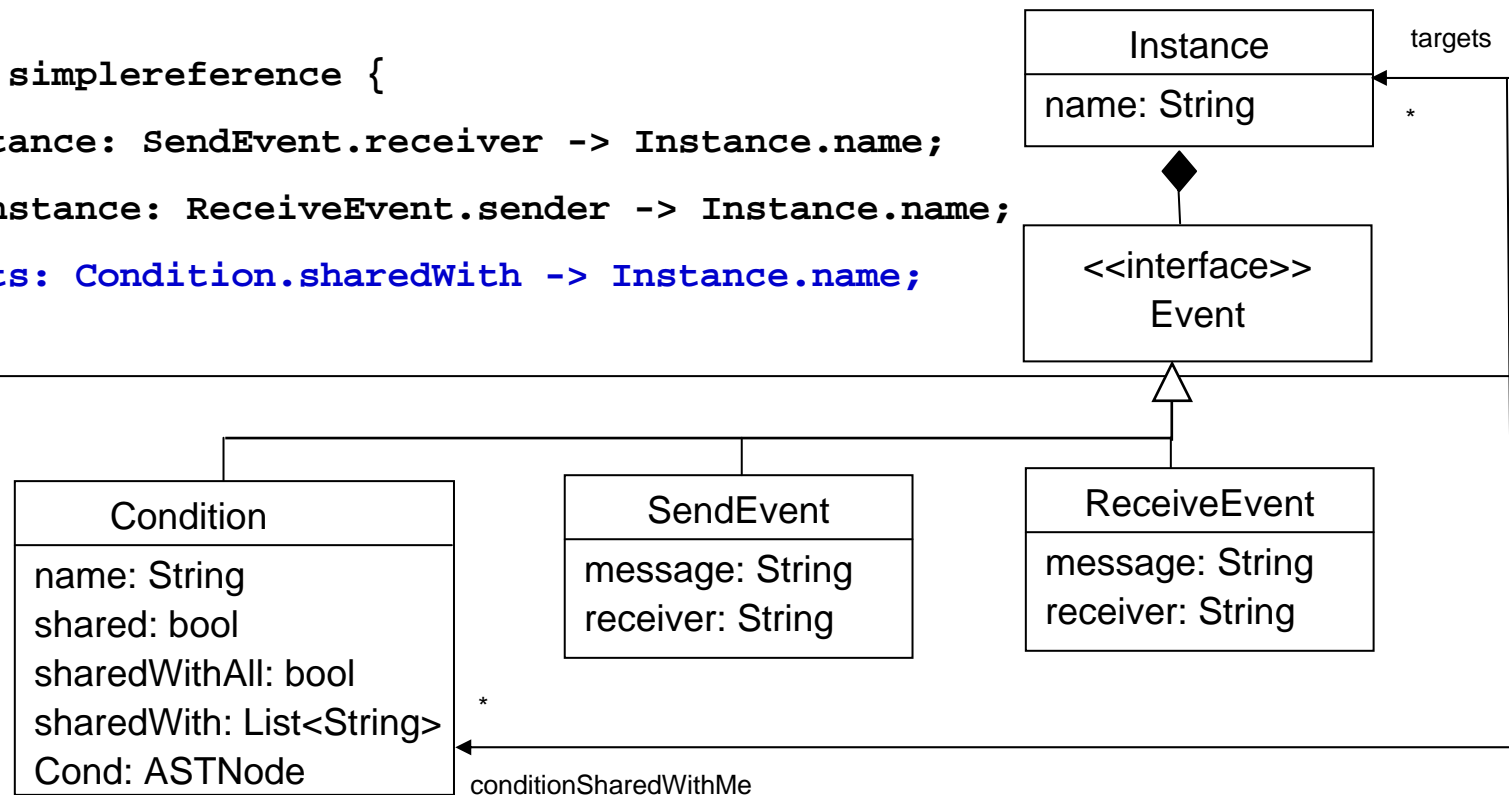
associations {
  SendEvent.toInstance * <-> 1 Instance.messagesToBeReceived;
  ReceiveEvent.fromInstance * <-> 1 Instance.messagesToBeSend;
  Condition.targets * <-> * Instance.conditionsSharedWithMe;
}

```

```

concept simplereference {
  ToInstance: SendEvent.receiver -> Instance.name;
  FromInstance: ReceiveEvent.sender -> Instance.name;
  Targets: Condition.sharedWith -> Instance.name;
}

```



Embedding of Java-Grammar

```
grammar MSC {  
  external Cond;  
  // ...  
}
```

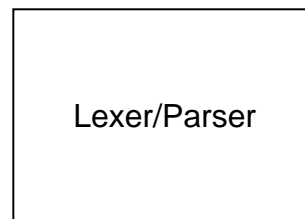
```
grammar Java {  
  interface Expression =  
  // ...  
}
```

The non-terminal Action is marked as external (extension point)

configure by Java-API or configuration script

```
grammar MSC {  
  external  
  Cond  
}
```

generation

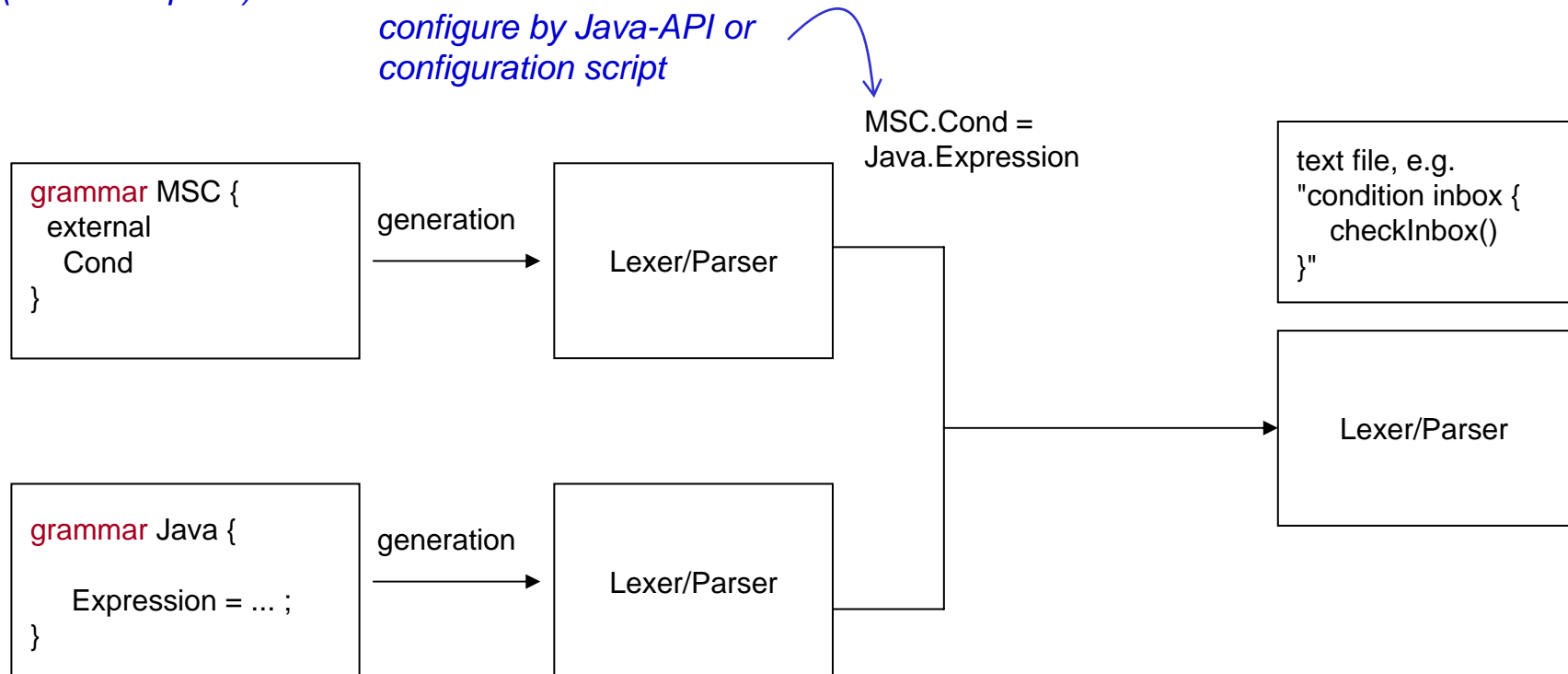
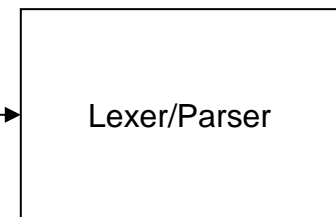
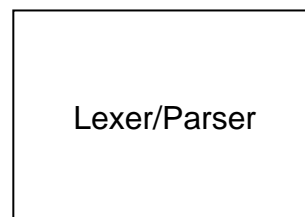


MSC.Cond =
Java.Expression

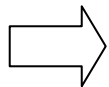
text file, e.g.
"condition inbox {
 checkInbox()
}"

```
grammar Java {  
  Expression = ... ;  
}
```

generation



Efficient Editor Generation for Compositional DSLs in Eclipse



1.

MontiCore

2.

Example: MSCs with Java

3.

Editor generation

4.

Live-Demo

5.

Conclusion

Elements of editors in Eclipse

The screenshot displays the Eclipse IDE interface with a code editor on the left and an outline view on the right. A context menu is open over the code editor, listing various actions such as 'Undo Typing', 'Save', 'Cut', 'Copy', 'Paste', 'Run As', and 'Generate Trace'. The code editor shows a sequence diagram in UML notation with syntax highlighting for keywords like 'instance', 'condition', and 'public'. The outline view shows a tree structure of the diagram elements, including 'sender', 'receiver', and 'inbox'. Annotations with arrows point to specific features: 'Code and comment regions' points to the code lines; 'Foldings' points to the folded state of the 'condition' block; 'Syntaxhighlighting of keywords' points to the highlighted keywords; 'Format action' points to the 'Format' option in the context menu; 'Editor action' points to the 'Generate Trace' option; and 'outline' points to the outline view.

```
1 msc mail{
2
3 instance sender{
4 //send a message
5 out message to receiver;
6 in response from receiver;
7 }
8
9 instance receiver{
10 in message from sender;
11 condition inbox {
12     checkInbox()
13 }
14 out response to sender;
15 }
16
17 public boolean checkInbox() {
18     return receiver.messages > 0;
19 }
20 }
21
```

Code and comment regions

Foldings

Syntaxhighlighting of keywords

Format action

Editor action

outline

Elements of editors in Eclipse

The screenshot displays the Eclipse IDE interface. On the left, the Package Explorer shows a project named 'Test' containing files 'mail.msc' and 'mail2.msc'. A context menu is open over 'mail.msc', listing actions like 'Open', 'Copy', 'Paste', 'Delete', 'Build Path', 'Refactor', 'Import...', 'Export...', 'Refresh', 'OSTP', 'Compose', 'Run As', 'Debug As', 'Team', 'Compare With', and 'Replace With'. An arrow labeled 'Navigator action' points to the 'Compose' option. The main editor window shows the content of 'mail.msc' with the following code:

```
msc mail {  
    instance sender {  
        // send a message  
        out message to receiver;  
        in response from receiver;  
    }  
  
    instance receiver {  
        in message from sender;  
        condition inbox {  
            checkInbox()  
        }  
  
        // produce a problem report  
        // out response to sender;  
    }  
  
    public boolean checkInbox() {  
        return receiver.messages > 0;  
    }  
}
```

A red 'x' icon in the left margin of the code editor indicates an error. An arrow points from this icon to the 'Problems' view at the bottom of the IDE. The 'Problems' view shows a table with one error:

Description	Resource	Path	Location
Message response is nowhere sent!	mail.msc	Test	line 6

An arrow labeled 'Problem Reports' points from the 'Problems' view back to the error icon in the code editor.

Navigator action

Problem Reports

Where shall certain elements be defined?

16

vor/03/01

- Fragments
 - Java, MSC
- Language
 - MSC with Java
- Tool
 - Multiple cooperating languages
- Syntaxhighlighting
- Foldable elements
- Elements of outline
- Context menu items
- Error messages
- Manifest.mf
- Plugin.xml

Where shall certain elements be defined?

Fragments

```
concept editorattributes {  
  
keywords:  
  msc, instance, in, out, to, from,  
  action, condition, shared, all;  
  
foldable:  
  MSC, Instance, Condition;  
  
segment:  
  MSC ("pict/m.gif") show: "MSC " name;  
segment:  
  Instance ("pict/i.gif") show: name;  
segment:  
  SendEvent ("pict/arrow.gif")  
  show: "Send to " receiver ":" message;  
segment:  
  ReceiveEvent ("pict/arrow.gif")  
  show: "Receive from " sender ":" message;  
segment:  
  Condition ("pict/c.gif") show: name;  
  
}
```

- Syntaxhighlighting
- Foldable elements
- Elements of outline

- Context menu items
- Error messages

- Manifest.mf
- Plugin.xml

Where shall certain elements be defined? Language

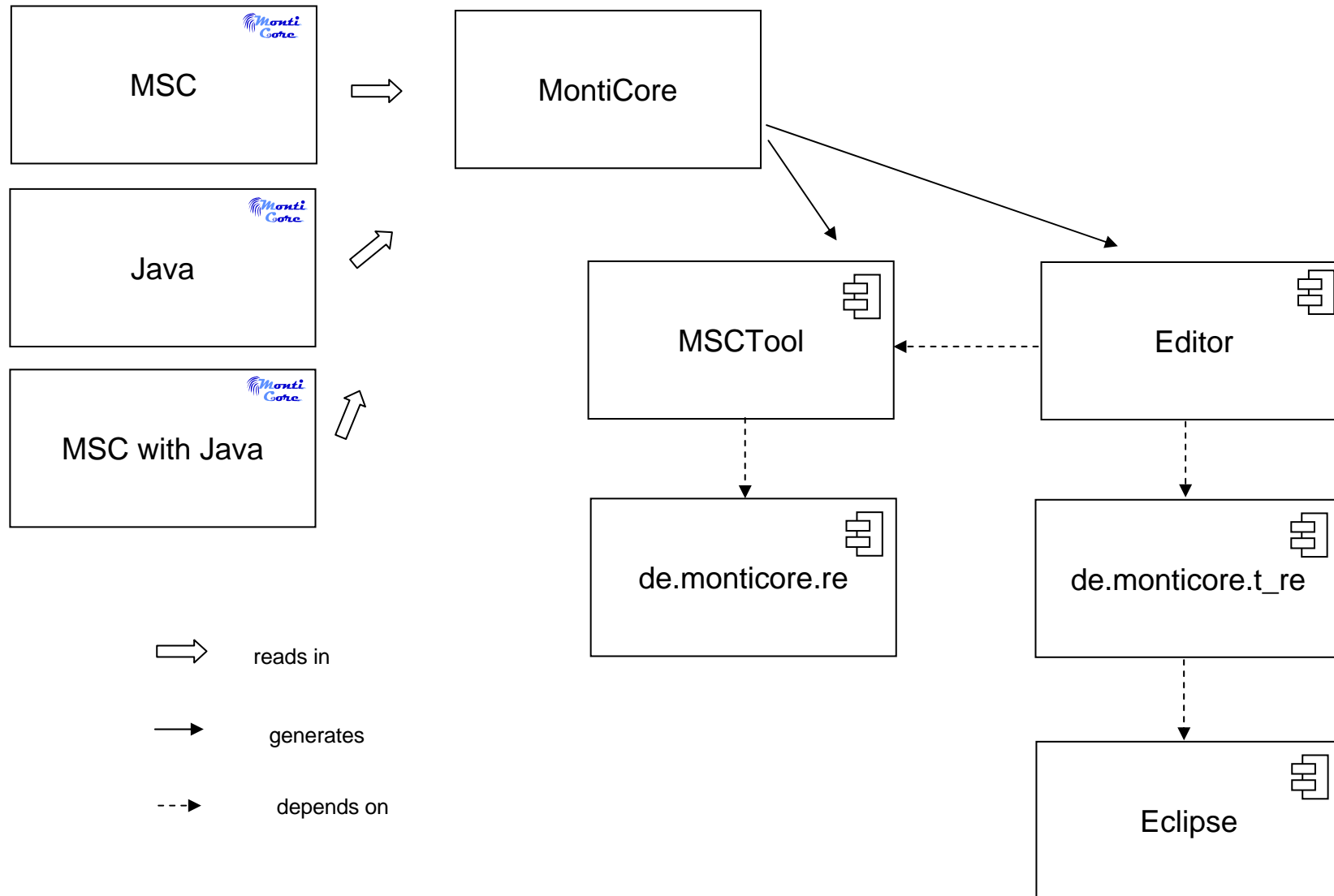
```
// context menu item for the editor:  
// there will be an item "Generate code".  
// When user selects this item,  
// CodegenAction will be invoked  
  
menumitem Generate Trace  
("mc.examples.msc.msc.action.GenerateTraceAction")  
;  
  
// popups in the package explorer:  
// select 1..n *.msc-files and press  
// right button. There will be an "Compose" item.  
// ComposeAction will be called if  
// the user selects it  
  
popup Compose  
("mc.examples.msc.msc.compose.ComposeAction");
```

- Syntaxhighlighting
- Foldable elements
- Elements of outline

- Context menu items
- Error messages

- Manifest.mf
- Plugin.xml

Overview editor generation



Efficient Editor Generation for Compositional DSLs in Eclipse

1.

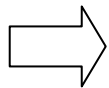
MontiCore

2.

Example: MSCs with Java

3.

Editor generation



4.

Live-Demo

5.

Conclusion

Efficient Editor Generation for Compositional DSLs in Eclipse

1.

MontiCore

2.

Example: MSCs with Java

3.

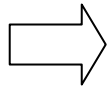
Editor generation

4.

Live-Demo

5.

Conclusion



- MontiCore is a framework for the development of domain specific languages (DSLs)
 - Language definition by context-free grammar
 - Composable use of languages by using language inheritance and **embedding**
 - Creation of generators **structured and simplified by framework** and workflows
 - **Generation of Eclipse-based editors**
 - Elements defined at fragment and language level
 - Fragments (included partial editors) can be compiled and combined later

- Available as an online service in the sse-lab:
www.monticore.de