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/

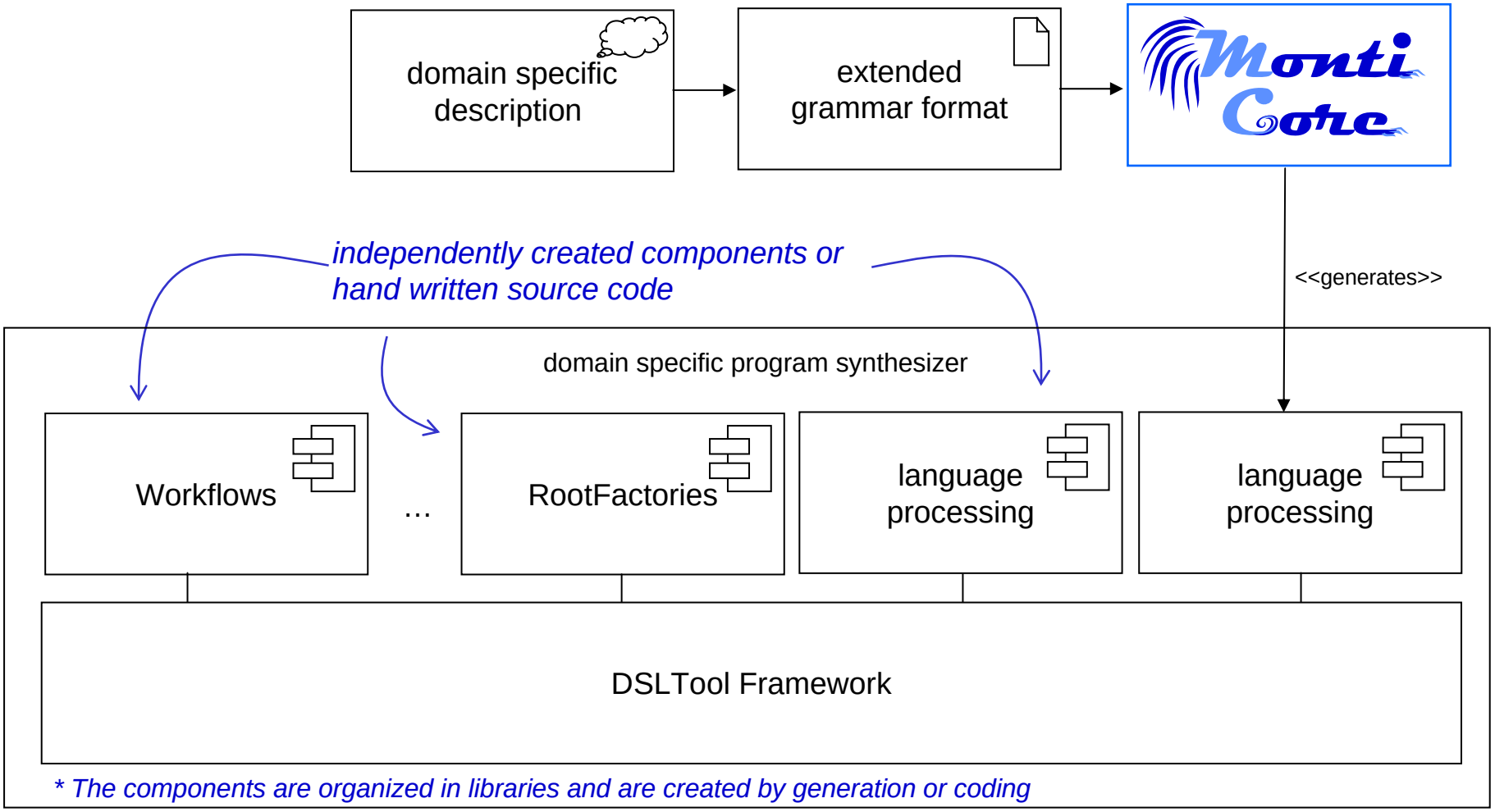
Efficient Editor Generation for Compositional DSLs in Eclipse

MontiCore – Design goals

- 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)

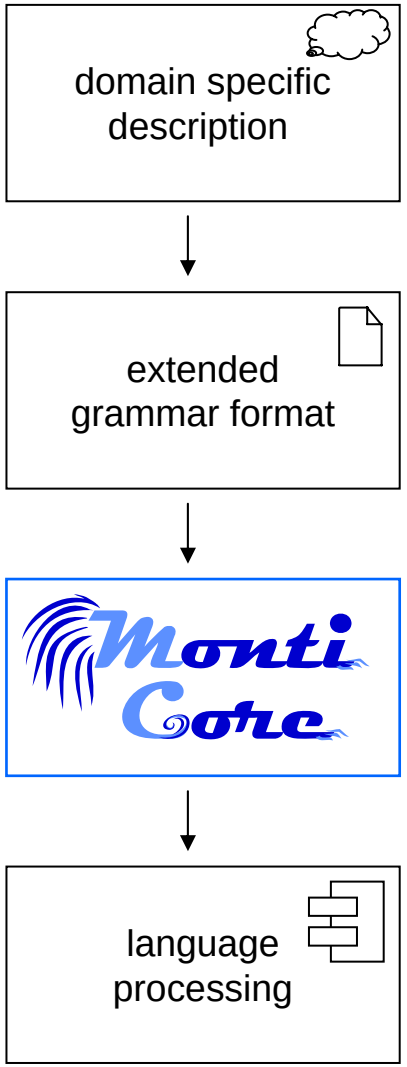
- MontiCore can be used to create domain specific program synthesizers



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

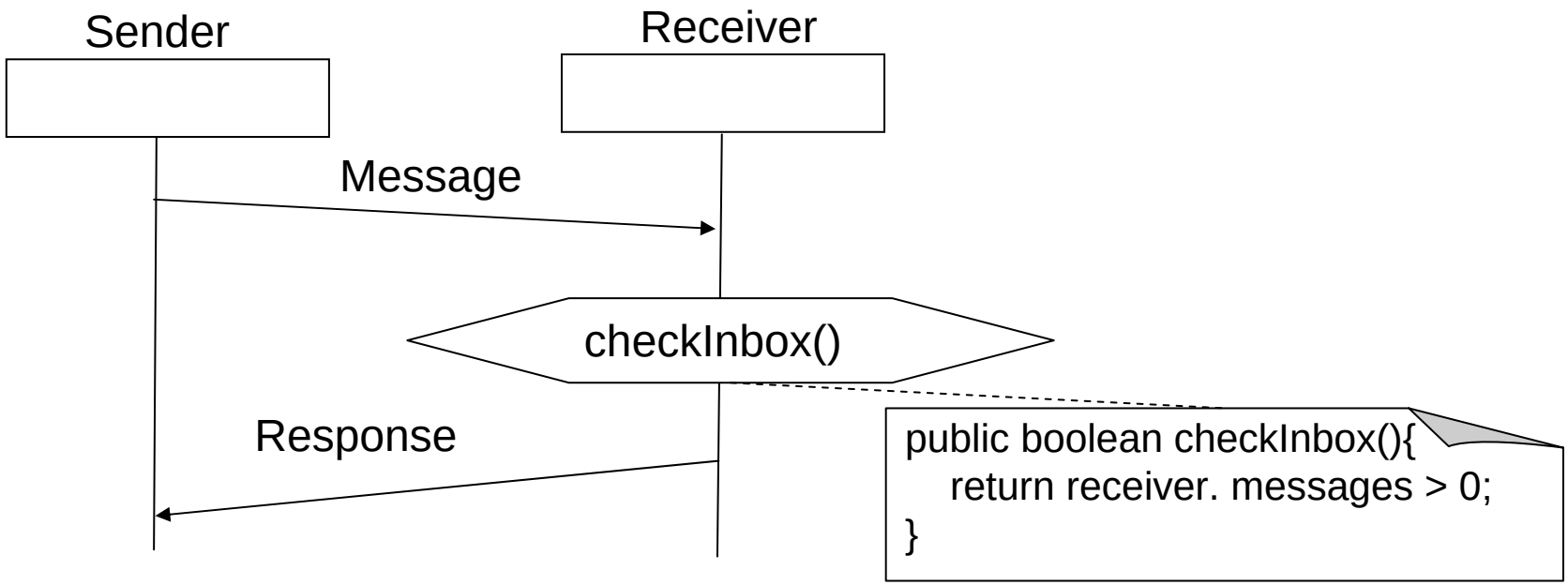
- MontiCore supports two types of reuse of languages
  - Embedding
    - Specifying grammars with holes (grammar fragments)
    - Combine multiple grammars to form new language without re-compilation
    - Mechanism suitable to embed (possibly multiple) action languages in models
  - Inheritance
    - 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

- 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

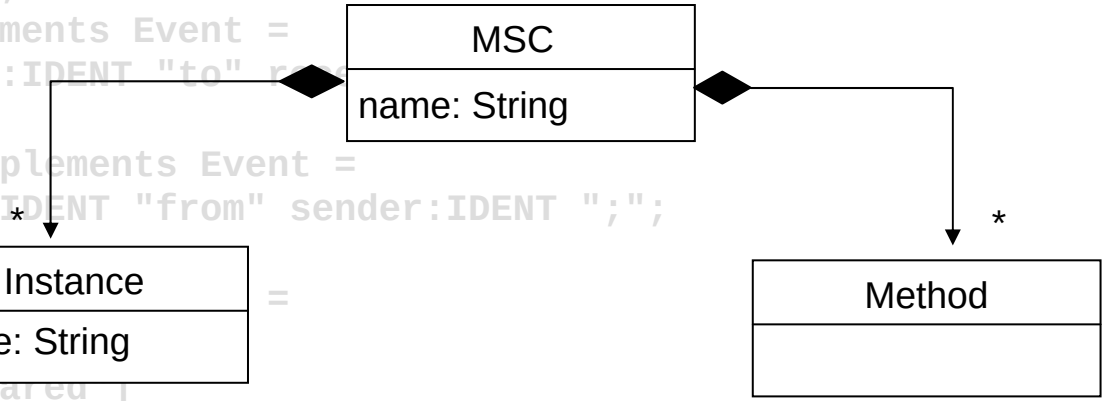
```
grammar MSC
  options{ compilationunit MSC }

  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

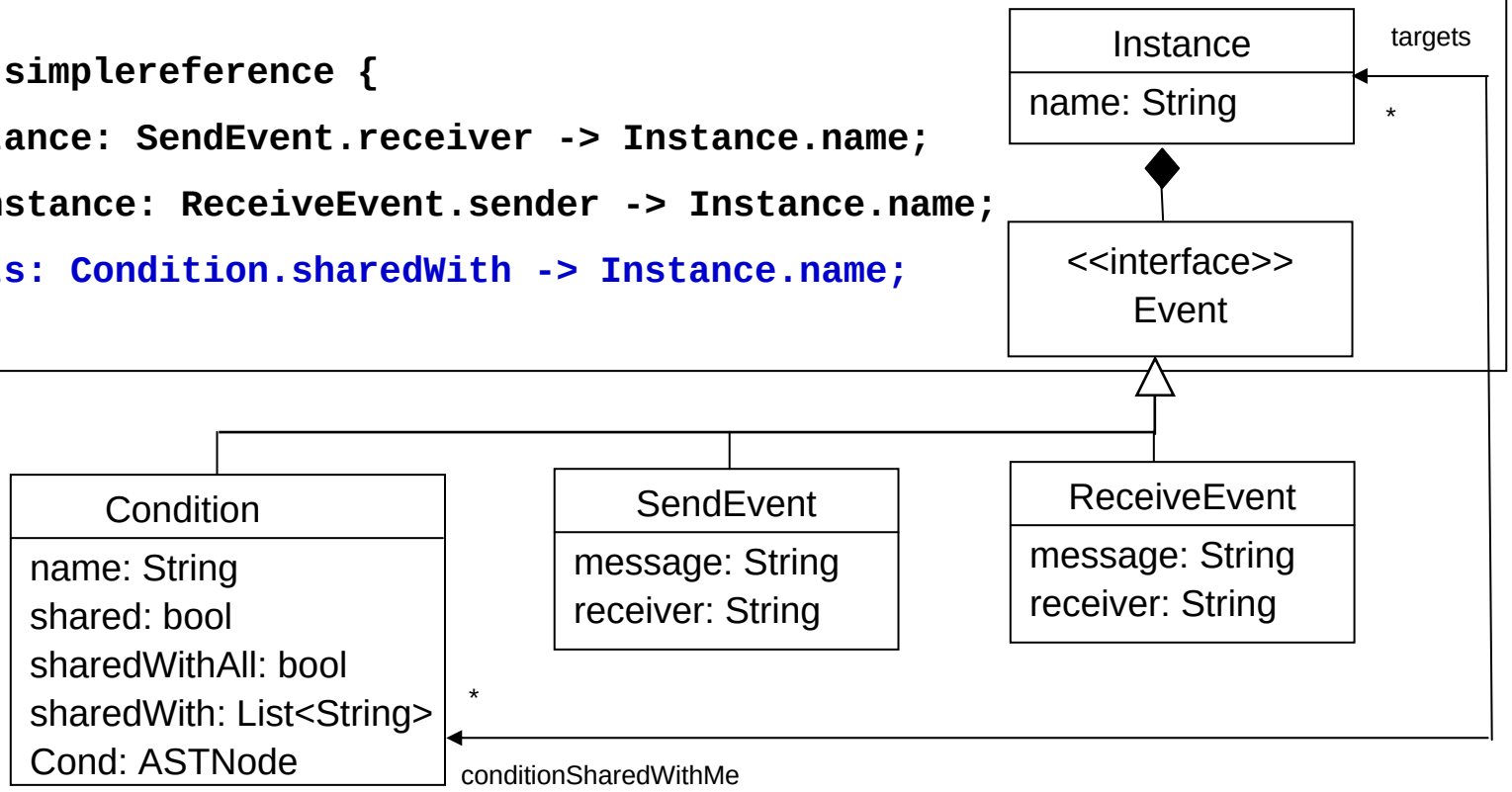
Example: MSCs with Java
Abstract syntax

```
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" re

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

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

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



Example: MSCs with Java
Definition of associations

```
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

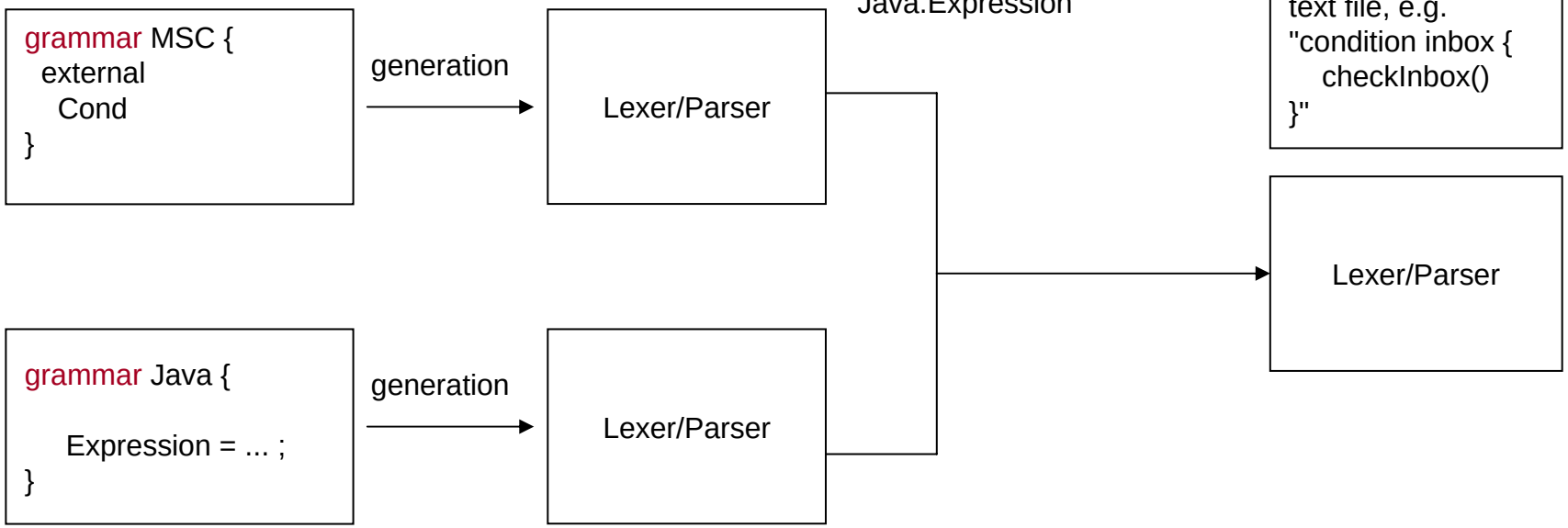
```
grammar MSC {
    external Cond;
    // ...
}
```

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

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

configure by Java-API or configuration script

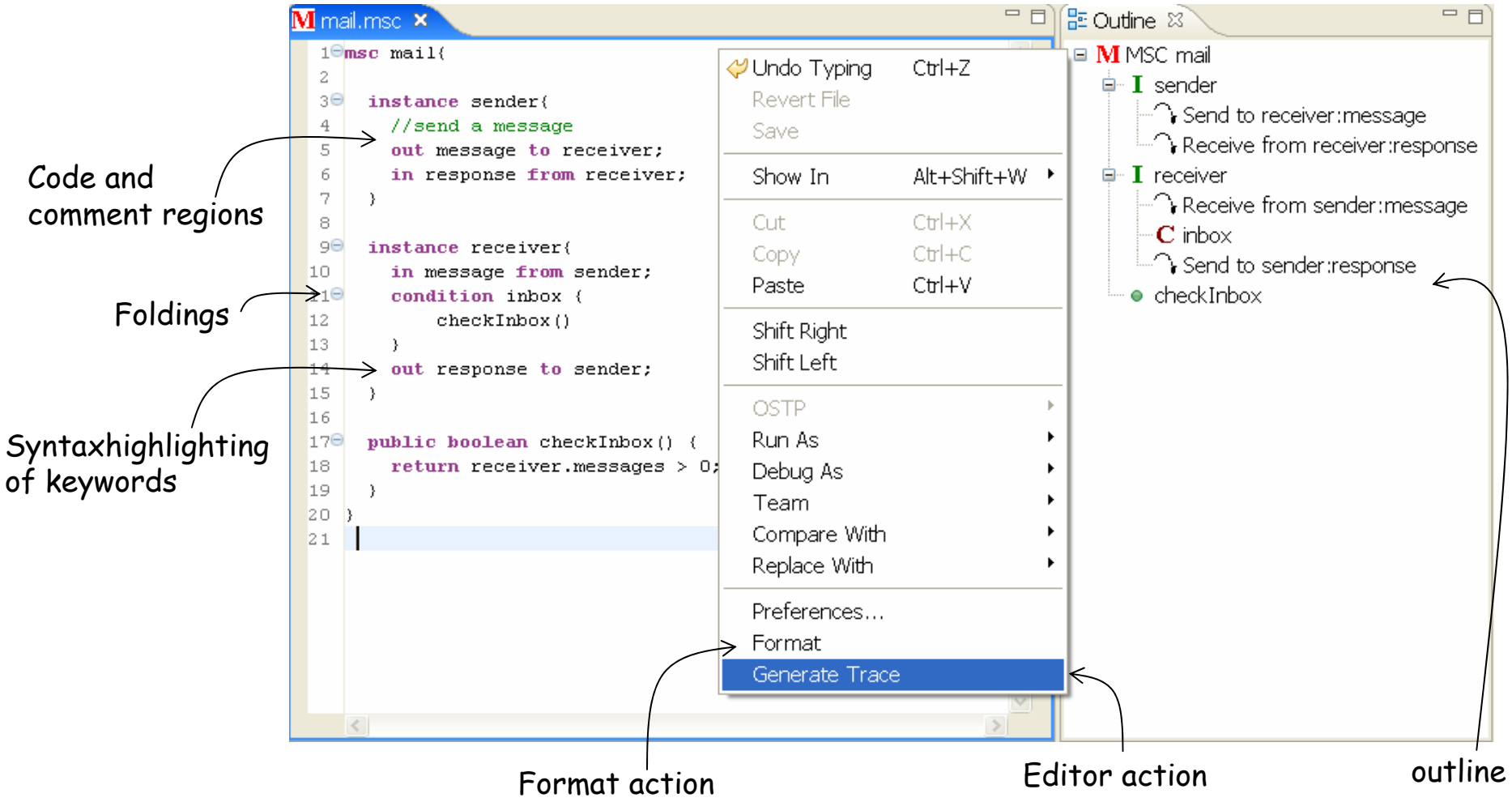
MSC.Cond = Java.Expression



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



Elements of editors in Eclipse

Package Explorer | Hierarchy | Outline

- Test
  - gen
  - JRE System Library [jre1.5.0_06]
  - mail.msc
  - mail2.msc
  - Second.fa

Open	F3
Copy	Ctrl+C
Copy Qualified Name	
Paste	Ctrl+V
Delete	Delete
Build Path	▸
Refactor	Alt+Shift+T ▸
Import...	
Export...	
Refresh	F5
OSTP	▸
Compose	
Run As	▸
Debug As	▸
Team	▸
Compare With	▸
Replace With	▸

Navigator action

mail.msc | mail2.msc

```
msc mail {

  instance sender {
    // send a message
    out message to receiver;
    in responce from receiver;
  }

  instance receiver {
    in message from sender;
    condition inbox {
      checkInbox()
    }

    // produce a problem report
    // out responce to sender;
  }

  public boolean checkInbox() {
    return receiver.messages > 0;
  }
}
```

Problems | Javadoc | Declaration

1 error, 0 warnings, 0 infos

Description	Resource	Path	Location
Errors (1 item)			
Message responce is nowhere sent!	mail.msc	Test	line 6

Problem Reports

Where shall certain elements be defined?

- Fragments
  - Java, MSC
- Syntaxhighlighting
- Foldable elements
- Elements of outline

- Language
  - MSC with Java
- Context menu items
- Error messages

- Tool
  - Multiple cooperating languages
- 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

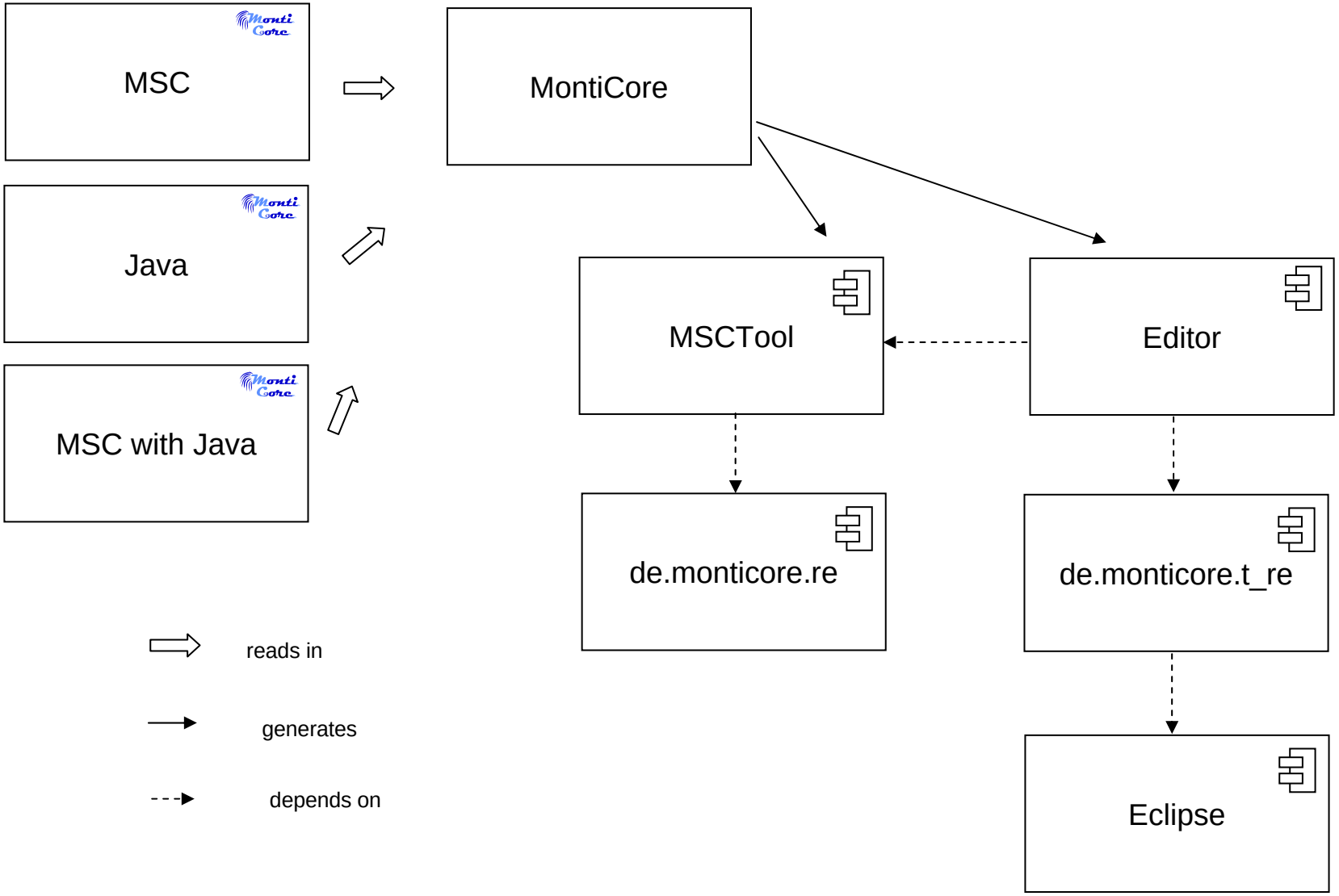
```
// context menu item for the editor:
// there will be an item "Generate code".
// When user selects this item,
// CodegenAction will be invoked

menuitem 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

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