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

- **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

![Diagram showing the process of creating a domain specific program synthesizer using MontiCore.](image)

*The components are organized in libraries and are created by generation or coding.*
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

```java
public boolean checkInbox()
{
    return receiver.messages > 0;
}
```
Example: MSCs with Java
Definition as MontiCore grammar

```monticore
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;
// ...
```
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;
// ...
}
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;
}
```

```
Instance
name: String

Condition
name: String
shared: bool
sharedWithAll: bool
sharedWith: List<String>
Cond: ASTNode

<<interface>> Event
message: String
receiver: String

SendEvent
message: String
receiver: String

ReceiveEvent
message: String
receiver: String
```

```
conditionSharedWithMe

targets

* 
```
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

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

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

```
grammar 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

Code and comment regions
Foldings
Syntax highlighting of keywords
Format action
Editor action
Outline
Elements of editors in Eclipse

Problem Reports

Navigator action
Where shall certain elements be defined?

- **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;
}
```

- Syntax highlighting
- Foldable elements
- Elements of outline
- Context menu items
- Error messages
- Manifest.mf
- Plugin.xml
Where shall certain elements be defined?

Language

- Syntax highlighting
- Foldable elements
- Elements of outline

---

// 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");

- Context menu items
- Error messages

- Manifest.mf
- Plugin.xml
Overview editor generation

MSC → MontiCore

MSC with Java

Java

MontiCore reads in MSC

MontiCore generates de.monticore.re

de.monticore.t_re depends on Eclipse

Eclipse
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 (including partial editors) can be compiled and combined later

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