s c c h

software competence center
hagenberg

A Comparison of Tool Support for Textual Domain-Specific Languages

Michael Pfeiffer and Josef Pichler

Dr. Josef Pichler

+43 7236 3343 867
josef.pichler@scch.at
www.scch.at

Das SCCH ist eine Initiative der

JOHANNES KEPLER
UNIVERSITÄT LINZ
Netzwerk für Forschung, Lehre und Praxis

Das SCCH befindet sich im

softwarepark
hagenberg

Motivation

s c c h
software competence center
hagenberg

- Domain-specific languages are languages tailored to a specific application domain
- Notation (textual, graphical, tabular) depends on the application domain
- Focus on textual languages following source-to-source transformation; host languages are Java or C#
- Successful application of DSL depends on provided tool support
- Tool support by *language workbenches* (Fowler)
  - oAW, MPS, MontiCore, IMP, TCS, TEF, CodeWorker, ...
- Feature model for expressing variations on DSL and DSL tools in general (not only textual languages) (Langlois et. al 2007)
- Reuse of an existing criteria catalog facilitates comparison of results

Overview of Tools

scch
software competence center
hagenberg

- openArchitectureWare (oAW, 4.3)
  - Eclipse, modular MDA/MDD generator framework, xText
- Meta Programming System (MPS, early access)
  - JetBrains
- MontiCore (1.1.5)
  - Academic, TU Braunschweig
- IDE Meta-Tooling Platform (IMP, 0.1.74)
  - Eclipse
- Textual Concrete Syntax (TCS, 0.0.1)
  - Eclipse, very similar to oAW
- Textual Editing Framework (TEF, 1.0.3)
  - Academic, Humboldt-Universität zu Berlin, very similar to oAW
- CodeWorker (3.5)
  - parsing tool and source code generator, no editor

Example

- Example (finite state machine) implemented with 4 tools
- Grammar productions for textual FSM

```
FSM = "inputAlphabet" string "outputAlphabet" string {State}.
State = ["start"] "state" id {Transition}.
Transition = "transition" char ["/" char] "->" id.
```

- Example: determine if a binary number has an odd or even number of zero digits

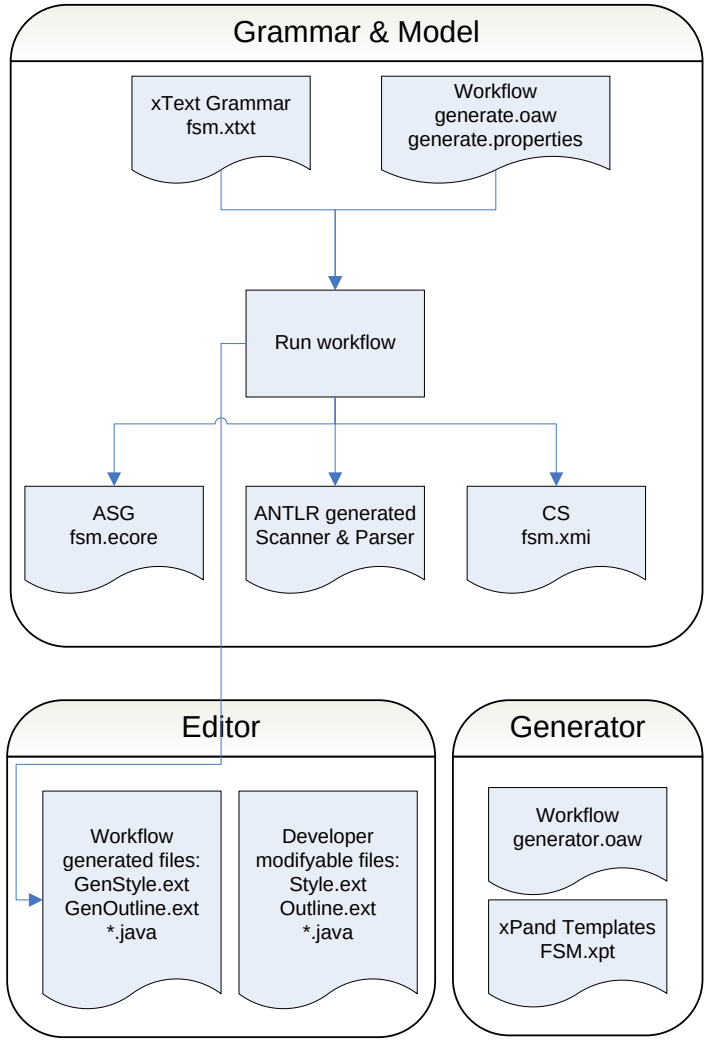
```
inputAlphabet "01"  // Digits of a binary number
outputAlphabet "eo" // Char 'e' for even and 'o' for odd

start state Even
transition 0 / o -> Odd
transition 1 / e -> Even
state Odd
transition 0 / e -> Even
transition 1 / o -> Odd
```

openArchitectureWare (xText)

s c c h
software competence center
hagenberg

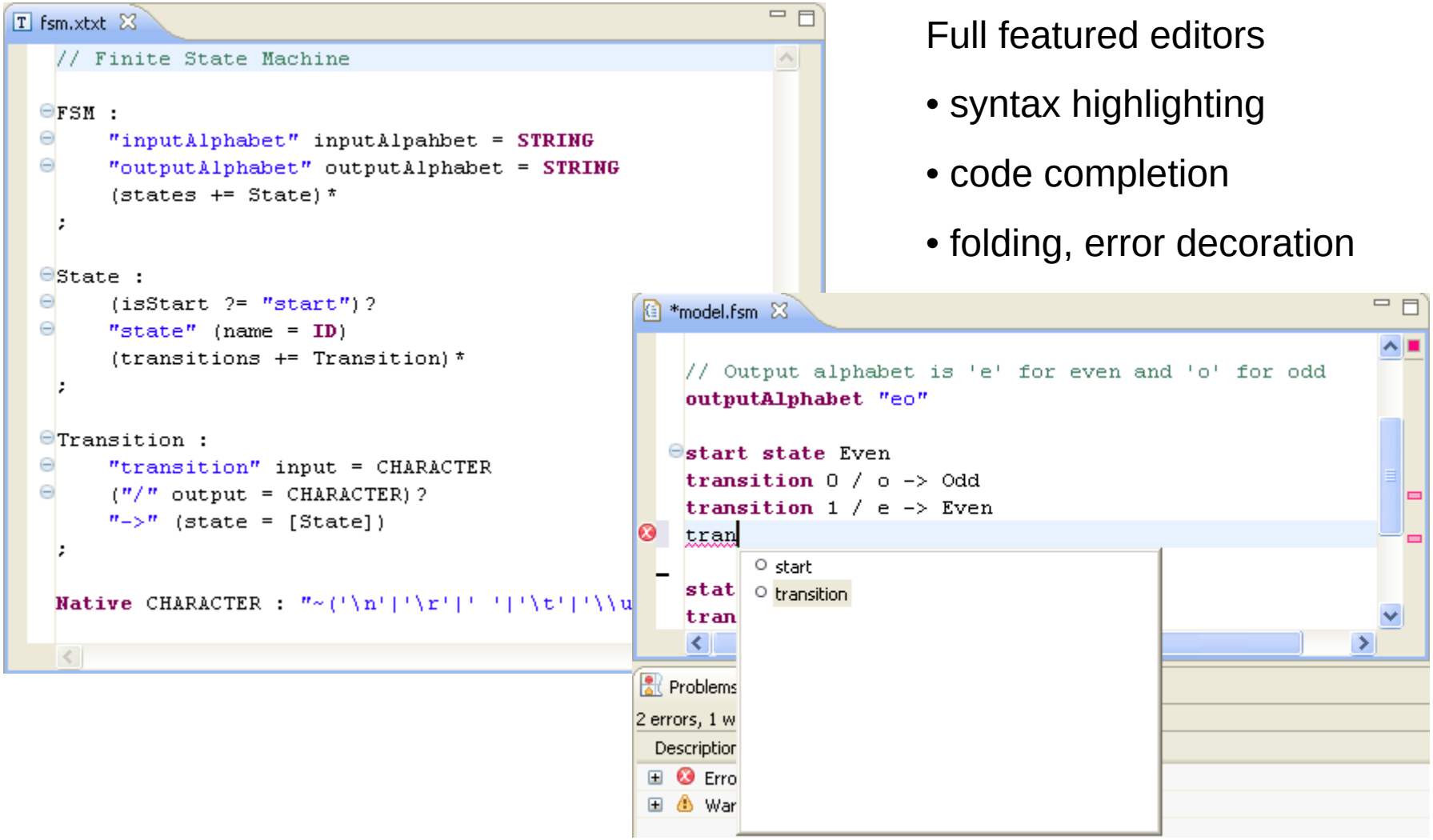
- Open source project
- Model Driven Development
- Eclipse, EMF
- Subcomponent xText for textual DSL

- Grammar defines ASG in form of dynamic EMF model
- Validation language
- Template language for xPand



openArchitectureWare (xText)$_2$

scch
software competence center
hagenberg



Full featured editors

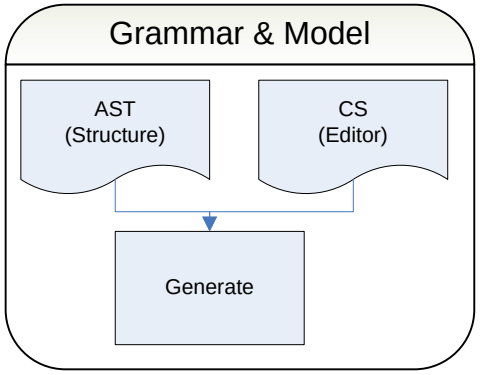
- syntax highlighting
- code completion
- folding, error decoration

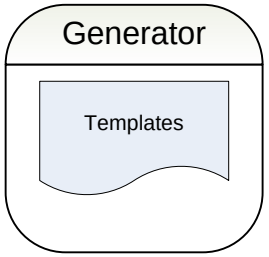
Meta Programming System

- JetBrains (IntelliJ), no release yet, early access program
- Cell-based editing model (no free text!)
- Powerful but complex

- Abstract syntax is specified by concepts (instead of production rules)
- All concepts are the structure of the language

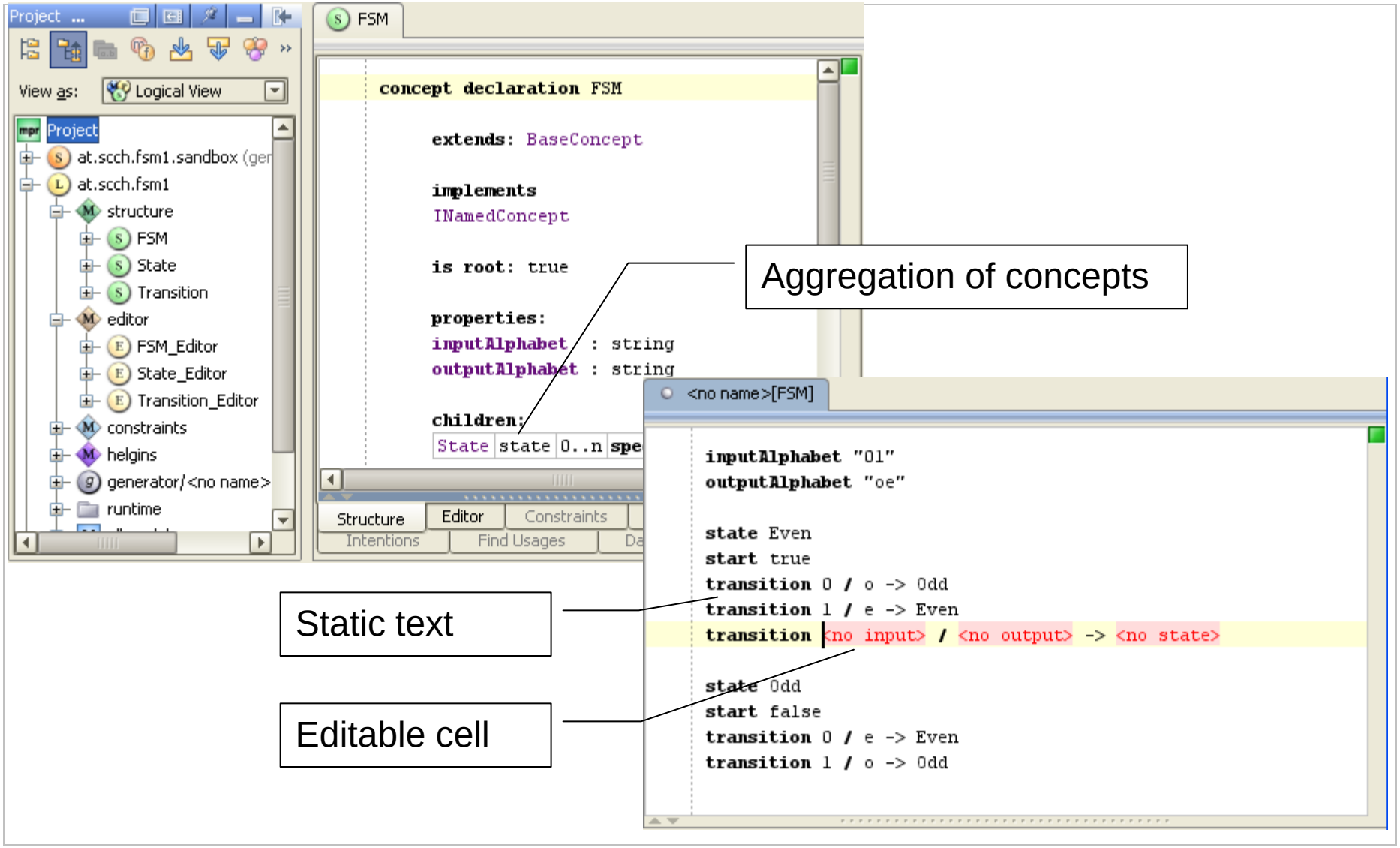
- Concrete syntax is static text (editor layout) for every concept (AS)





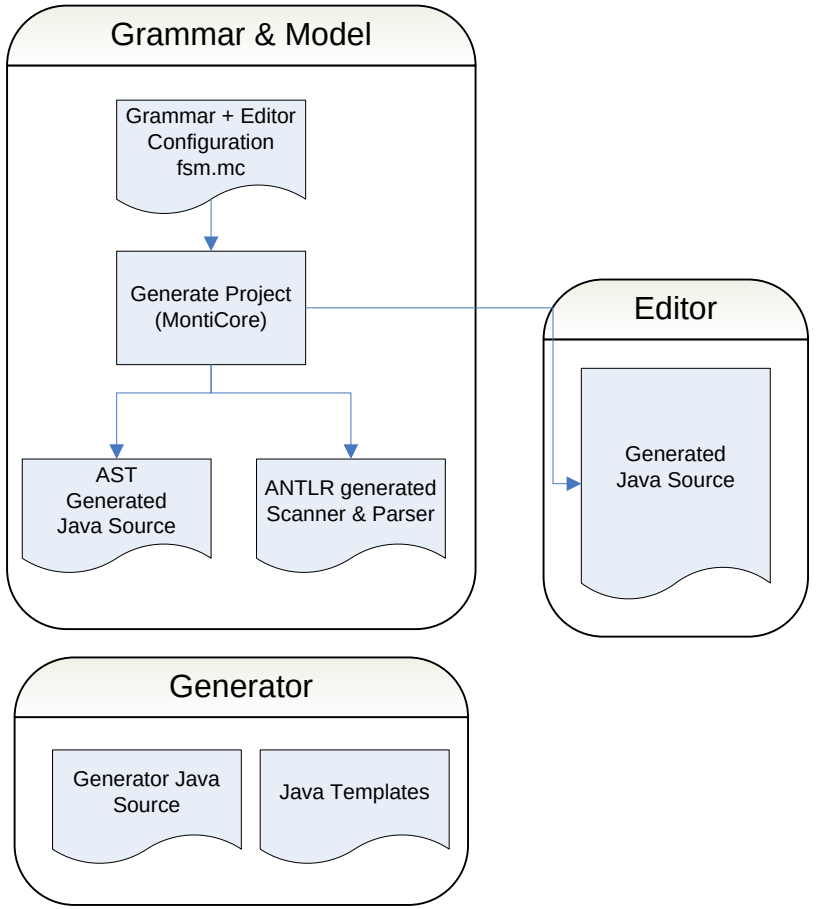
Meta Programming System$_2$

scch
software competence center
hagenberg



MontiCore

- Academic project, TU Braunschweig
- Eclipse based
- Grammar for concrete syntax and abstract syntax (similar to input format of ANTLR)
- Generated AST
- Generated compiler-frontend (ANTLR)
- Transformation
  - Visitors on AST
  - Template engine



MontiCore$_2$

scch
software competence center
hagenberg

FSM.mc

```
package at.scch.fsm;

grammar FSM {

    FSM =
        "inputAlphabet" inputAlphabet:STRING
        "outputAlphabet" outputAlphabet:STRING
        (states:State)*
    ;

    State =
        (start:["start"])?
        "state" name:IDENT
        (transitions:Transition)*
    ;

    Transition =
        "transition" input:CHARACTER
        ("/" output:CHARACTER)?
        "->" state:IDENT
    ;

    ident CHARACTER =
        '\''! ('a'..'z' | 'A'..'Z' | '0'..'9') '\
        : char
```

*example.fsm

```
 7 inputAlphabet "01"
 8 outputAlphabet "oe"
 9
10 start state Even
11 transition '0' / 'o' -> Odd
12 transition '1' / 'e' -> Even
13 tran
14
15 state Odd
16 transition '0' / 'e' -> Even
17 transition '1' / 'o' -> Odd
```

Editors

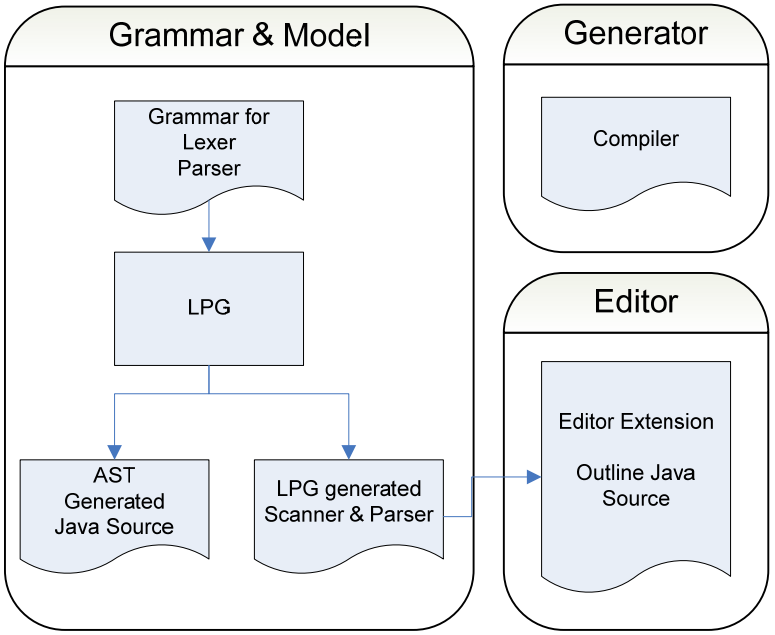
- syntax highlighting
- folding
- error decoration

IDE Meta-Tooling Platform

- Open source, IBM Watson Research
- Eclipse
- Wizard to generate code skeletons for a large range of IDE features

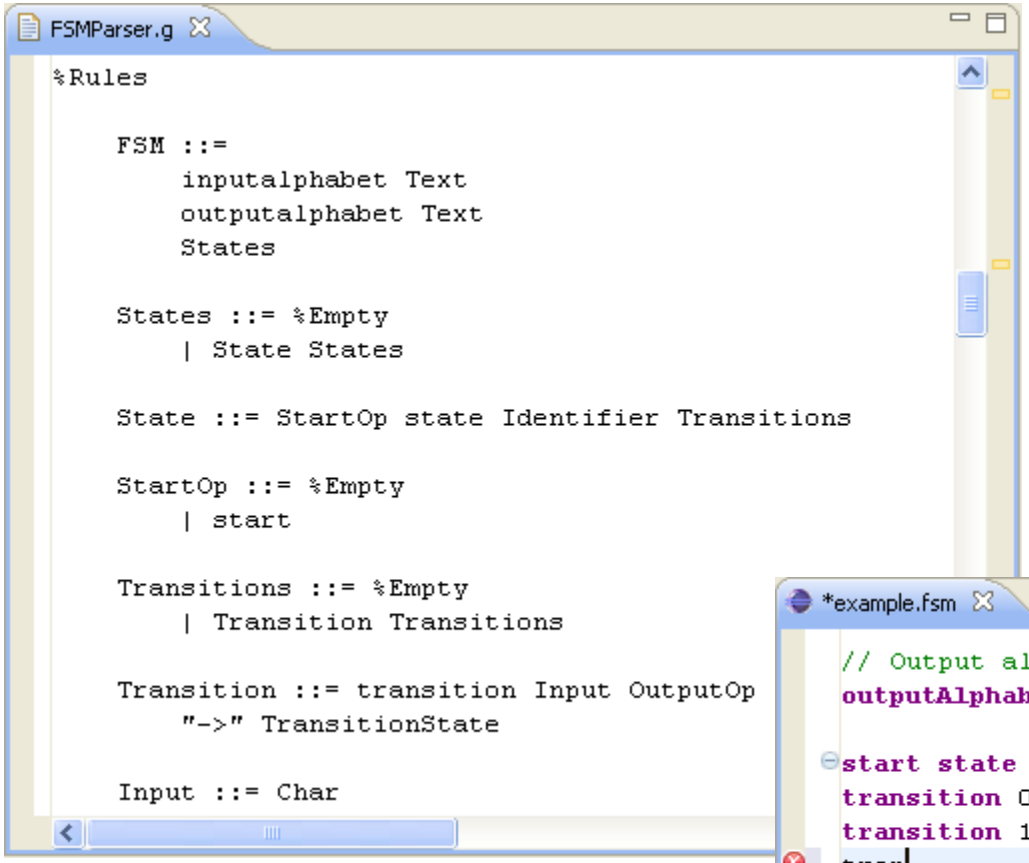
- Grammar for concrete syntax
- LPG generates scanner, parser, and data structures for AST

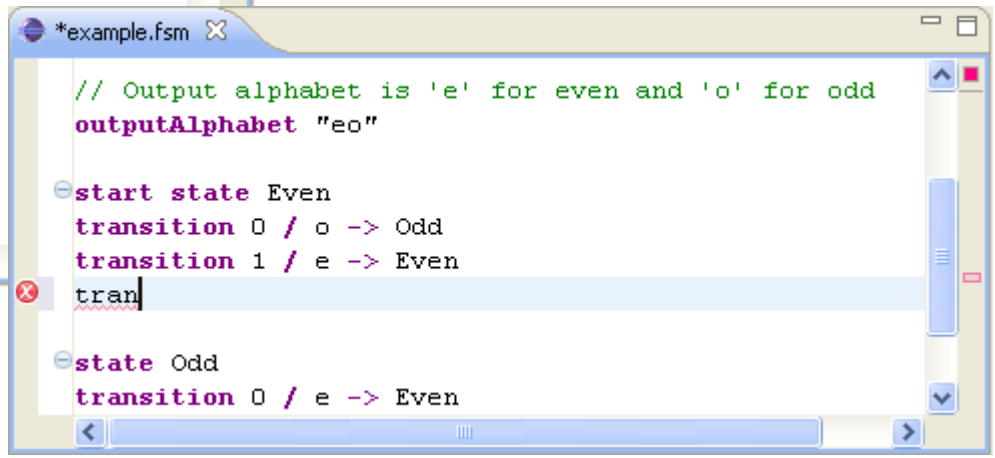
- Transformation by visitors, no template engine



IDE Meta-Tooling Platform$_2$

scch
software competence center
hagenberg





Editors

- syntax highlighting
- folding, formatting
- code completion

Criteria

- Criteria for our comparison are a subset derived from feature model
- Groups: language (LA), transformation (TR), and tools (TO)

Language (abstract and concrete syntax)

- LA-AS1. abstract syntax tree or abstract syntax graph
- LA-AS2. grammar or meta-model
- LA-AS3. can be composed

- LA-CS1. technique to map abstract syntax to concrete syntax
- LA-CS2. representation (text, graphic, ...) for the concrete syntax
- LA-CS3. declarative or imperative style

Comparison - Language

scch
software competence center
hagenberg

	oAW	MontiCore	MPS	IMP
LA-AS1	Graph	Tree	Graph	Tree
LA-AS2	Meta-model (ECore)	Java Classes	Meta-model	Java Classes
LA-AS3	Composition	Composition	Composition	No built-in support
LA-CS1	Explicit. (1)	Explicit. (1)	Explicit. (2)	Implicit. (3)
LA-CS2	Text	Text	Text/XML	Text

- Abstract syntax: graph and trees; generated Java classes or models
- Mapping between abstract syntax and concrete syntax
  - Def. of concrete syntax is mixed with def. of abstract syntax (1)
  - Explicit definition of editor layout (CS) for each AS elements (2)
  - Implicit definition of abstract syntax by concrete syntax (3)
- Textual representation (XML only in MPS)

Criteria

Transformation (target asset and operational translation)

- TR-TA1. target asset (model, text, graphic, binary)
- TR-TA2. destructive or incremental update
- TR-TA3. kind of support for integration of target assets

- TR-OT1. transformation techniques (M→M, M→T, T→T, T→M)
- TR-OT2. transformation by compilation or interpretation
- TR-OT3. internal or external environment for transformation
- TR-OT4. implicit or explicit scheduling
- TR-OT5. internal or external location
- TR-OT6. automation level (manual or automated)

Comparison - Transformation

scch
software competence center
hagenberg

	oAW	MontiCore	MPS	IMP
TR-TA1	Text	Text	Model or Text	No built-in support.
TR-TA2	Destructive	Destructive	Destructive	No built-in support
TR-TA3	No support for integration with target assets available.			

- Different levels of tool support for generating target assets (TA)
- Template engines for model-to-text transformation
  - outstanding (oAW), rudimentary (MontiCore)
  - MPS requires generation of target model that is transformed to text
- All tools (except IMP) provide destructive update (overwrites target assets)
- No support for integration of generated target assents

Comparison - Transformation$_2$

scch
software competence center
hagenberg

	oAW	MontiCore	MPS	IMP
TR-OT1	M2T (xPand)	M2M (visitor) M2T (visitor/template engine)	M2M (generator) M2T (indirect)	No built-in support.
TR-OT2	Interpretation. Templates filled at runtime.			Compilation.
TR-OT3	External. Runtime workbench must be launched.	External. Runtime workbench must be launched.	Internal. Editor and Transformation in MPS.	External. Runtime workbench must be launched.
TR-OT4	Explicit. Workflow triggered by user.	Explicit. Triggered by user.	Explicit. Triggered by user.	Implicit. Eclipse builder runs after change.
TR-OT5	Internal. Runs in runtime workbench (Eclipse) or in MPS			
TR-OT6	Manual. Triggered by user.			Automatically after change.

- Concerning operational translation, tools are very similar

Criteria

Tool (respect of abstraction, assistance)

- TO-RA1. respect of abstraction (intrusive or seamless)
- TO-AS1. kind of assistance (static or adaptive)
- TO-AS2. process guidance (step or workflow)
- TO-AS3. checking (completeness or consistency)

Omitted

- Quality factors
- Process features

Comparison - Tool

	oAW	MontiCore	MPS	IMP
TO-RA1	Depends on DSL. Our example DSL is seamless.			
TO-AS1	Adaptive. Code completion. Validation.	Adaptive. Validation.	Adaptive. Code completion. Validation.	Adaptive. Validation.
TO-AS2	Neither step nor workflow process guidance is supported.			
TO-AS3	Completeness. Using constraint language. Consistency. Ensured by grammar validation.	Completeness. Needs implementation. Consistency. Ensured by grammar validation.	Completeness. Using constraint language. Consistency. Grammar validation.	Completeness. Needs implementation. Consistency. Ensured by grammar validation.

- High variance ranging from
  - plain text editor for CS definition (IMP)
  - to an editor with syntax coloring (MontiCore)
  - code completion and validation while typing (oAW and MPS)

Conclusion

s c c h
software competence center
hagenberg

- Language workbenches are driven by existing IDE
  - JetBrains (IntelliJ) for MPS
  - Eclipse Platform (JDT) for all others
- Eclipse technology (platform, EMF, ...) predominant for textual DSLs
- Free text editing including features like code completion but missing refactoring, searching, references, ...
- MPS with unique cell based editing model
- Reuse of compiler generators (ANTLR, LGP) to generate scanner/parser for text-to-model transformation
- Template engine or visitor pattern for model-to-text