SE2 FACHVORTRAG | NIKE

DOMÄNSPEZIFISCHE SPRACHEN

GLIEDERUNG

- 1. Was ist eine DSL?
- 2. Arten von DSL
  2.1 interne DSL/eingebettete DSL
  2.2 externe DSL
- 3. Werkzeuge zur Erstellung von DSL
- 4. Vor - und Nachteile von DSL
- 5. Quellen

1.

WAS SIND DSL?

1. WAS SIND DSL?

- „Eine **domänenspezifische Sprache** oder **anwendungsspezifische Sprache** ist eine formale Sprache die zur Interaktion zwischen Menschen und digital arbeitenden Computern („Computersprache“ für ein bestimmtes Problemfeld (die sogenannte Domäne) entworfen und implementiert wird. Beim Entwurf einer DSL wird man bemüht sein, einen hohen Grad an Problemspezifität zu erreichen: die Sprache soll alle Probleme der Domäne darstellen können und nichts darstellen können, was außerhalb der Domäne liegt. Dadurch ist sie durch Domänenspezialisten ohne besonderes Zusatzwissen bedienbar.“

1. WAS SIND DSL?

- Sprachen mit Spezialisierung auf eine Domäne
- Ziel: Eine Sprache zur effizienteren Lösung des Problems, leichte Bedienbarkeit und Lesbarkeit
- Außerhalb der Domäne nicht verwendbar
- Beispiel: CSS,SQL, xUnit Frameworks,Rake

2.

ARTEN VON DSL

2.1 INTERNE SPRACHEN / EINGEBETTETE DSL

- verwenden einer Hostsprache als Basis der eigenen Sprache
- keine eigene Syntax
- Beispiel: Rake, xUnit Frameworks

2.1 XUNIT / FRAMEWORKS

- verschiedene Frameworks für Modultests
- überprüfen verschiedener Elemente (Units)
- erstes XUNIT Framework von Kent Beck für „Smalltalk“ entwickelt

- **Beispiele:**
  - JUNIT für Java
  - CPPUnit für C++
  - PHPUnit für PHP

  (für weitere siehe: https://en.wikipedia.org/wiki/List_of_unit_testing_frameworks)

2.1 JUNIT

- **JUnit** ist Java Test-Framework für automatisierte Regressionstests
- **Testklassen** definieren Testfälle für bestimmte Zielklassen einer Anwendung
- **Testfälle** sind Methoden, die auf Tests auf den Zielklassen ausführen

```java
public class MyTests {

    @Test
    public void multiplicationOfZeroIntegersShouldReturnZero() {
        MyClass tester = new MyClass(); // MyClass is tested

        // assert statements
        assertEquals("10 x 0 must be 0", 0, tester.multiply(10, 0));
        assertEquals("0 x 10 must be 0", 0, tester.multiply(0, 10));
        assertEquals("0 x 0 must be 0", 0, tester.multiply(0, 0));
    }
}
```

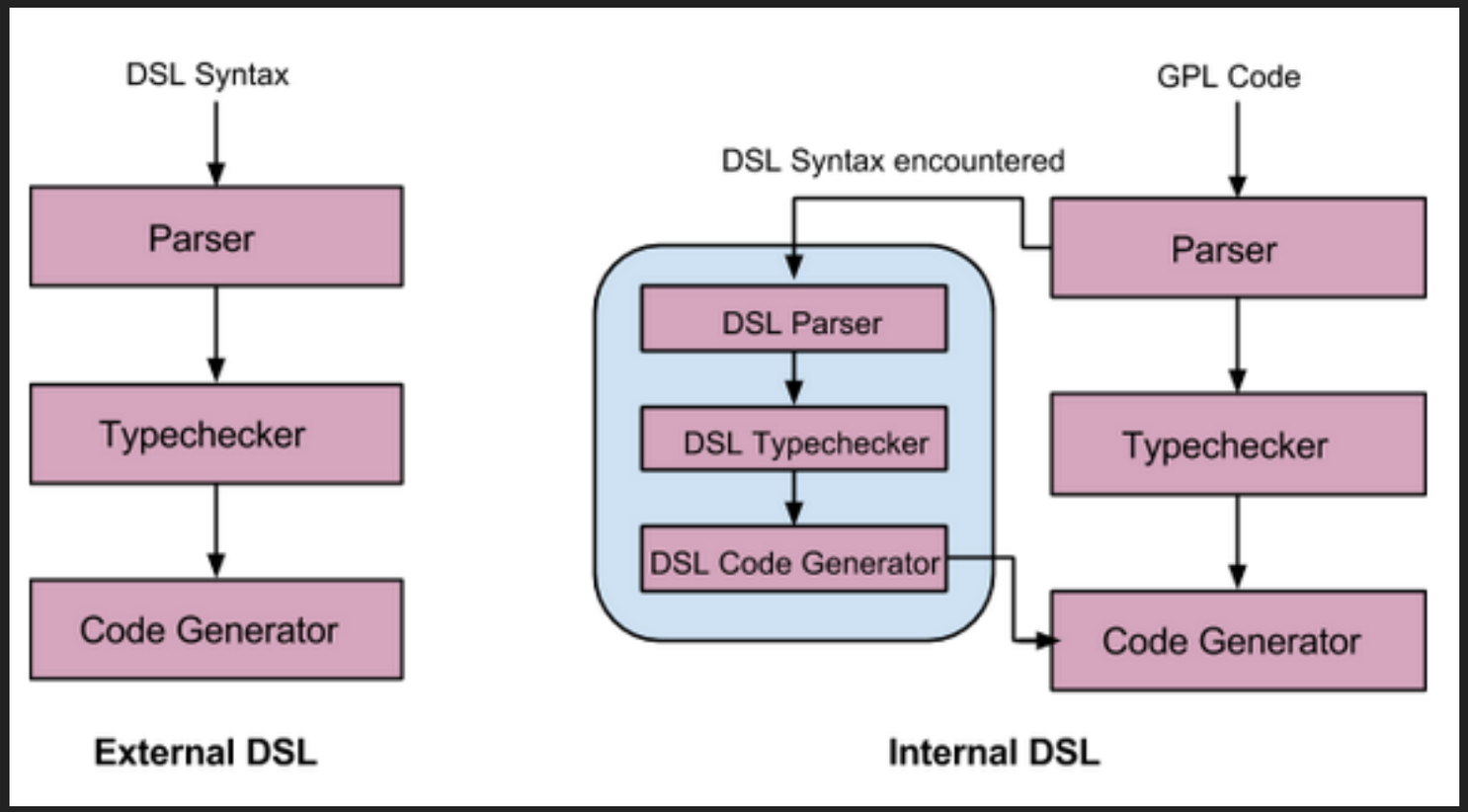
2.1 JUNIT/ZUSICHERUNGEN

Zusicherung	Ist wahr, wenn ...
assertTrue(String message, boolean condition)	die Bedingung wahr ist
assertFalse(boolean condition)	die Bedingung falsch ist
assertNotNull(Object object)	der Wert von *object* nicht null ist
assertNull(Object object)	der Wert von *object* null ist
assertEquals(Object expected, Object actual)	die Objekte gleich sind (prüft mit equals)
assertEquals(long expected, long actual)	die Werte gleich sind (int oder long)
assertEquals(double expected, double actual)	die Werte gleich sind (float oder double)

Quelle: Prog2 Strippgen

2.2 EXTERNE SPRACHEN

- komplett eigene Syntax
- benötigt eigenen Compiler
- hohe Flexibilität, ausdrucksstark
- Beispiele: reguläre Ausdrücke, SQL, CSS
- effiziente Implementierung durch Werkzeuge



2.2 EXTERNE SPRACHEN

- SQL

```sql
CREATE TABLE Employee (
  id INT NOT NULL IDENTITY (1,1) PRIMARY KEY,
  name VARCHAR(50),
  surname VARCHAR(50),
  address VARCHAR(255),
  city VARCHAR(60),
  telephone VARCHAR(15),
)
```

- CSS

```css
body {
  text-align: left;
  font-family: helvetica, sans-serif;
}
h1 {
  border: 1px solid #b4b9bf;
  font-size: 35px;}
```

3.

WERKZEUGE ZUR ERSTELLUNG VON DSL

3. ALLGEMEINES

- man benötigt Werkzeuge für Spezifikation, Transformation, syntaxgestütztes Editieren
- Sprachentwicklungssysteme und Toolkits generieren:
  - Compiler/Interpreter
  - Konsistenzprüfer
  - Editor mit integrierter Syntaxüberprüfung
  - Analysewerkzeuge
  - Entwicklungsumgebungen
  - Applikations-und Codegenerator

3. TEXTUELLE DSL

- textuelle sind deklarativ und in reiner Textform
- Modellierung benötigt Werkzeuge zur:
  - lexikalischen und syntaktischen Analyse
  - semantischen Transformation in eine andere DSL

3. GRAFISCHE DSL

- visuelle DSL werden grafisch modelliert
- Modellelemente werden in Metamodellen spezifiziert
- visuell = textuell, da grafische Modelle in Textform transformierbar sind

3. TEXTUELLE VS GRAFISCHE DSL

- Grafische DSL:
  - Übersicht
  - Weglassen von Details
  - Ausschnitte
  - Präsentation, Diskussionsgrundlage
- Textuelle DSL:
  - Genaue Details
  - Ausführliche und umfassende Spezifikation

3. LISTE VON WERKZEUGEN

- AMMI - FAW GmbH
- DEViL - Universität Paderborn
- DSL Tools für Visual Studio
- **Eclipse Xtext (Teil des EMF Projekts)**
- Eclipse GMF
- EMFText
- Microsoft Oslo für textuelle und graphische DSLs
- OOMEGA TMK (Textual Modelling Kit)
- **MetaEdit+ - MetaCase**
- JetBrains Meta Programming System

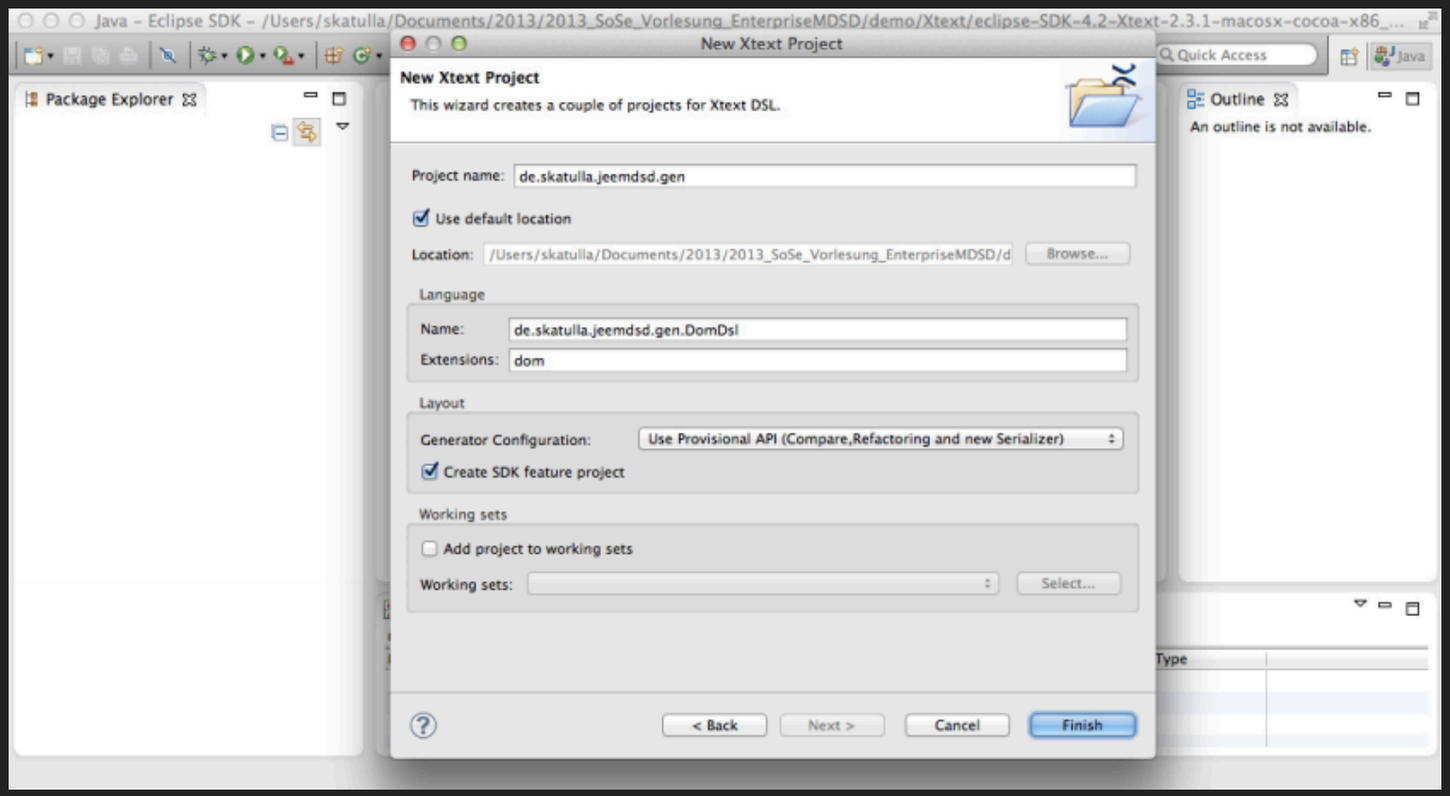
3. BEISPIEL: XTEXT (TEXTUELL)

- Open-Source Framework für Entwicklung von Programmiersprachen & DSL
- Syntaxdefinition mit Grammatiken in EBNF
- Generierung eines textuellen Editors für definierte Sprache als Eclipse Plug-in
- plattformübergreifend

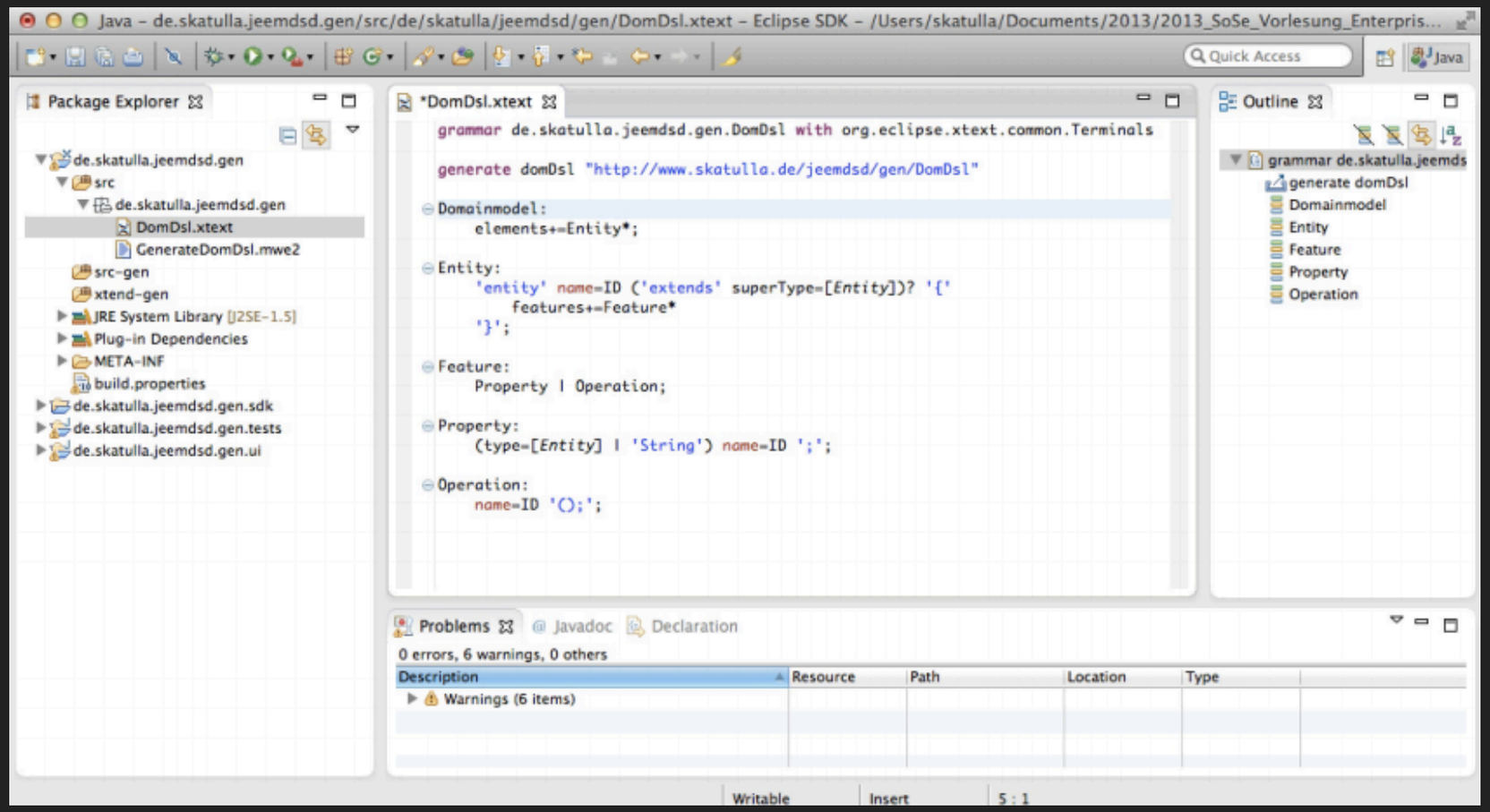
3. FUNKTIONALITÄTEN VON XTEXT

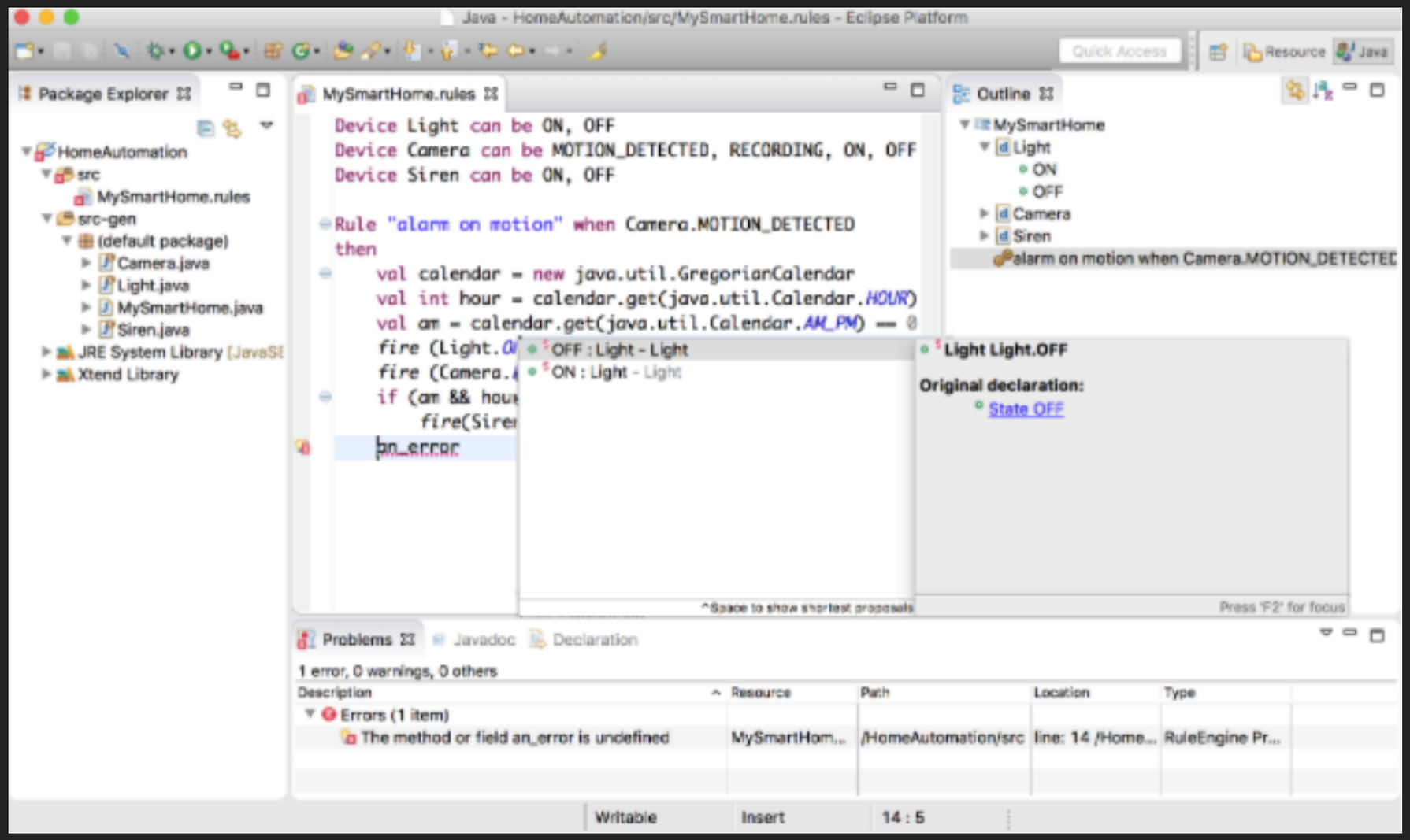
- Workflow
- Compilerfunktionen
  - Parser/Grammatik der Sprache
  - Code-Generierung
- IDE-Konzepte:
  - Editor
  - Syntax-Hightlighting
  - Autoformatierung

Projekt anlegen



Grammatik definieren





Nach Generierung von Metamodell, Ecore-Klassen und Editor

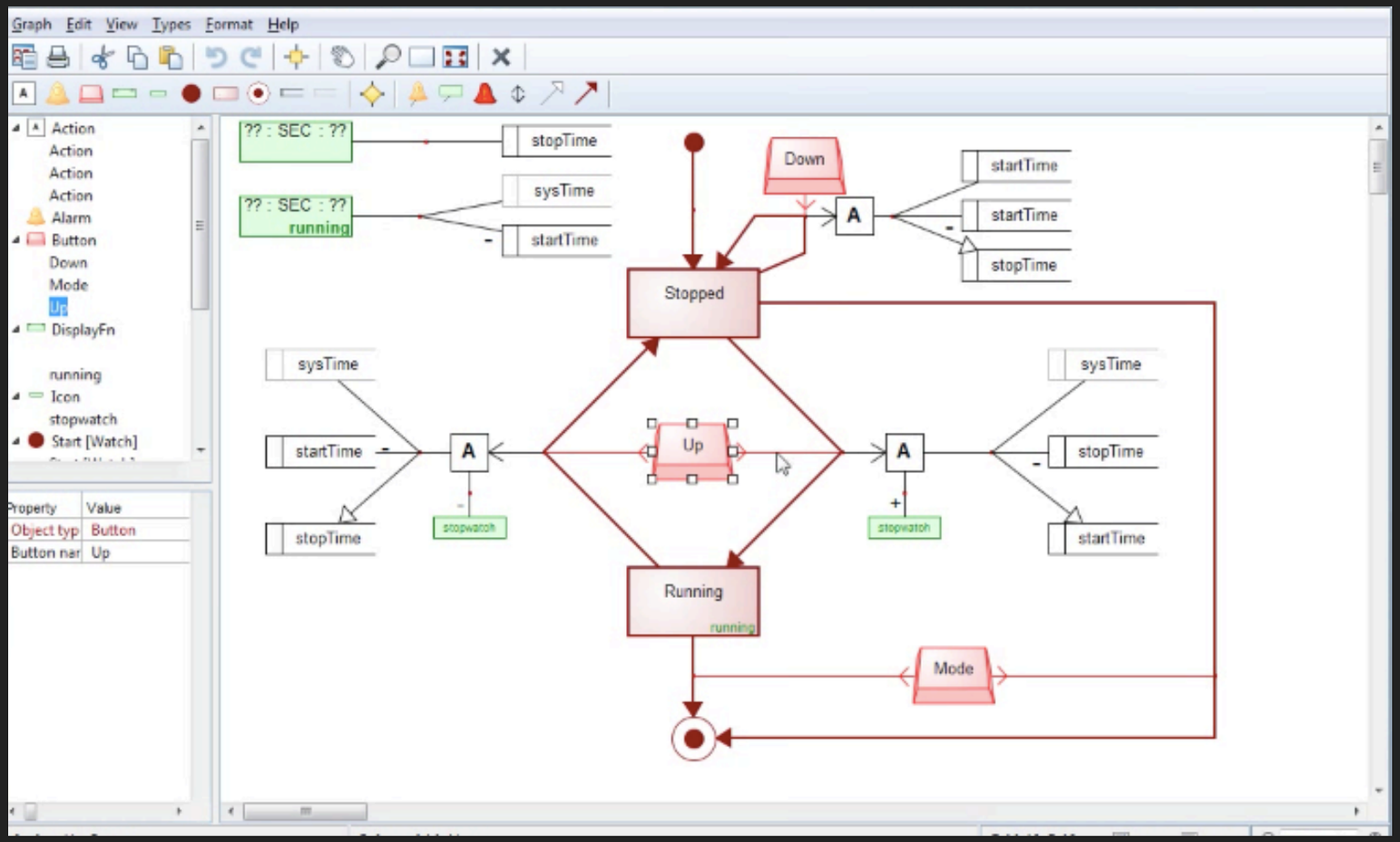
- rechte Seite Outline View: Inhalt des Dokuments kompakt dargestellt
- linke Seite Package Explorer: zeigt selbsterstellte und codegenerierte Dateien

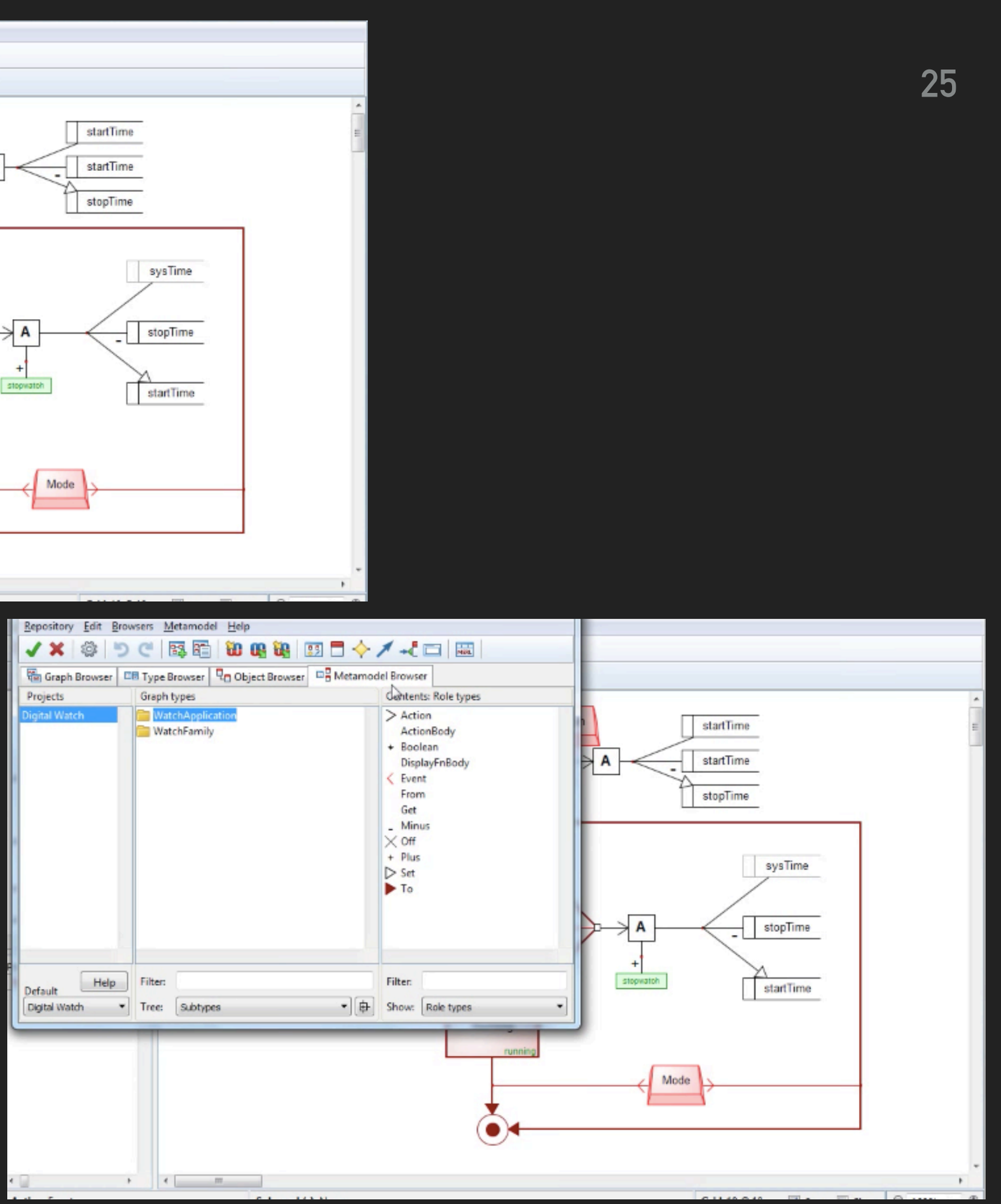
3. BEISPIEL: METAEDIT+ (GRAFISCH) MetaCase

- Fokus liegt auf grafischen DSLs und Codegenerierung
- Erlaubt mit Editor Symbole zu erstellen und mit Metamodell zu verbinden
- geeignet für Modellmigration
- finnische Firma MetaCase betreibt MetaEdit+

3. BEISPIEL: METAEDIT+ (GRAFISCH) MetaCase

- Metatypen sind:
  - Graph: visuelle Darstellung aller Metatypen
  - Object: Hauptelement der Modellierung
  - Property: enthält Beschreibung eines Objects
  - Port: alle Instanzen des erweiterten Objects teilen gleichen Port
  - Relationship: Explizite Beziehungen zwischen Gruppen von Objekten (mit Rollenspezifikation gebunden)


Graph Edit View Types Format Help
Action
Action
Action
Action
Alarm
Button
Down
Mode
Up
DisplayFn
running
Icon
stopwatch
Start [Watch]
Property
Value
Object typ
Button
Button nar
Up
?? : SEC : ??
stopTime
?? : SEC : ??
running
sysTime
startTime
Down
A
startTime
startTime
stopTime
Stopped
sysTime
startTime
stopTime
A
Up
A
sysTime
stopTime
startTime
stopwatch
stopwatch
Running
running
Mode


Repository Edit Browsers Metamodel Help
Graph Browser
Type Browser
Object Browser
Metamodel Browser
Projects
Digital Watch
Graph types
WatchApplication
WatchFamily
Contents: Role types
Action
ActionBody
Boolean
DisplayFnBody
Event
From
Get
Minus
Off
Plus
Set
To
Default
Help
Digital Watch
Filter:
Tree:
Subtypes
Filter:
Show:
Role types
startTime
startTime
stopTime
A
sysTime
stopTime
startTime
stopwatch
running
Mode

4.

VOR- UND NACHTEILE VON DSL

4. VOR-UND NACHTEILE VON DSL

- Vorteile:
  - gute Lesbarkeit
  - bessere Abbildung der Domäne
  - gute Wartbarkeit und Portabilität
  - nutzbar für Spezialisten mit wenig IT-Kenntnissen
  - interne DSL: Tools sind vorhanden, wenig Einarbeitungszeit

4. VOR-UND NACHTEILE VON DSL

- Nachteile:
  - lange Entwicklungszeit
  - hohe Entwicklungskosten
  - genaues Wissen über die Domain erforderlich
  - durch Verwendung von DSL steigt Anzahl an verwendeten Sprachen

ABSCHLIEßENDE WORTE:

Die Grenze zwischen normalen Programmiersprachen und domänspezifischen Sprachen ist nicht immer festgelegt.

NIKE

5.

QUELLEN

5. QUELLEN

- https://de.wikipedia.org/wiki/Dom%C3%A4nenspezifische_Sprache
- http://www.itwissen.info/DSL-domain-specific-language-Domaenenspezifische-Sprache.html
- http://www.informatik.uni-jena.de/dbis/lehre/ss2013/skat/2013_SoSe_EnterpriseMDSD_2_Modellierung-V01.pdf
- http://www-i3.informatik.rwth-aachen.de/files/seminar-ws0607/thema06.pdf
- https://msdn.microsoft.com/de-de/library/bb126278.aspx
- http://www.itwissen.info/DSL-domain-specific-language-Domaenenspezifische-Sprache.html

5. QUELLEN

- https://wiki.eclipse.org/Graphical_Modeling_Framework/Tutorial/Part_1
- http://www.informatik.uni-jena.de/dbis/lehre/ss2013/skat/2013_SoSe_EnterpriseMDSD_2_Modellierung-V01.pdf
- http://voelter.de/data/articles/ToolsFuerDSLs.pdf
- https://www.heise.de/developer/artikel/Werkzeuge-fuer-domaenenspezifische-Sprachen-227190.html
- https://www.youtube.com/watch?v=w2GfLlXe4zw
- https://de.wikipedia.org/wiki/Modellgetriebene_Softwareentwicklung
- https://de.wikipedia.org/wiki/Metamodell

5. QUELLEN

- https://de.wikipedia.org/wiki/Xtext
- https://de.wikipedia.org/wiki/Eclipse_Modeling_Framework
- https://de.wikipedia.org/wiki/Dom%C3%A4nenspezifische_Sprache
- http://www.informatik.uni-jena.de/dbis/lehre/ss2010/skatulla/MDSD2010_1-4.pdf
- http://www.dannyfleming.uk/wp-content/uploads/2015/03/internal-vs-external.png
- http://www.metacase.com/de/mwb/

5. QUELLEN

- https://www.uni-marburg.de/fb12/arbeitsgruppen/swt/lehre/files/mddma1415/MDD4MA141119.pdf
- http://java-pro.de/language-engineering-mit-eclipse-xtext/
- https://www-old.cs.uni-paderborn.de/fileadmin/Informatik/FG-Kastens/Lehre/Seminare/SLE_WS_2009_10/Vortraege/Schumacher.pdf
- https://eclipse.org/Xtext/documentation/301_grammarlanguage.html
- http://www-i3.informatik.rwth-aachen.de/files/seminar-ws0607/thema06.pdf