

# Xtext-based Tool Support for ABS

**Prof. Dr. Thomas Baar**

thomas.baar@htw-berlin.de



**Hochschule für Technik  
und Wirtschaft Berlin**

*University of Applied Sciences*

ABS Workshop 2018, Darmstadt, May 29<sup>th</sup> 2018

# Outline

- History of this Talk
- Xtext-Technology
- Wrestling with ABS
  - Language Description
- Examples/Demo
  - Focus on Visualization

# Outline

- History of this Talk
- Xtext-Technology
- Wrestling with ABS
  - Language Description
- Examples/Demo
  - Focus on Visualization

# History of this Talk

- Oct. 2017
  - two students and me take part at KeY-workshop, Rastatt
    - They present Xtext-Technology
- Winter term 2017/18
  - The two students work on Xtext-support for ABS as a semester project
    - Results were rather disappointing
- Summer term 2018
  - I work on Xtext support for ABS
  - **Result: Prototypical tool for a FRAGMENT of ABS**

# Outline

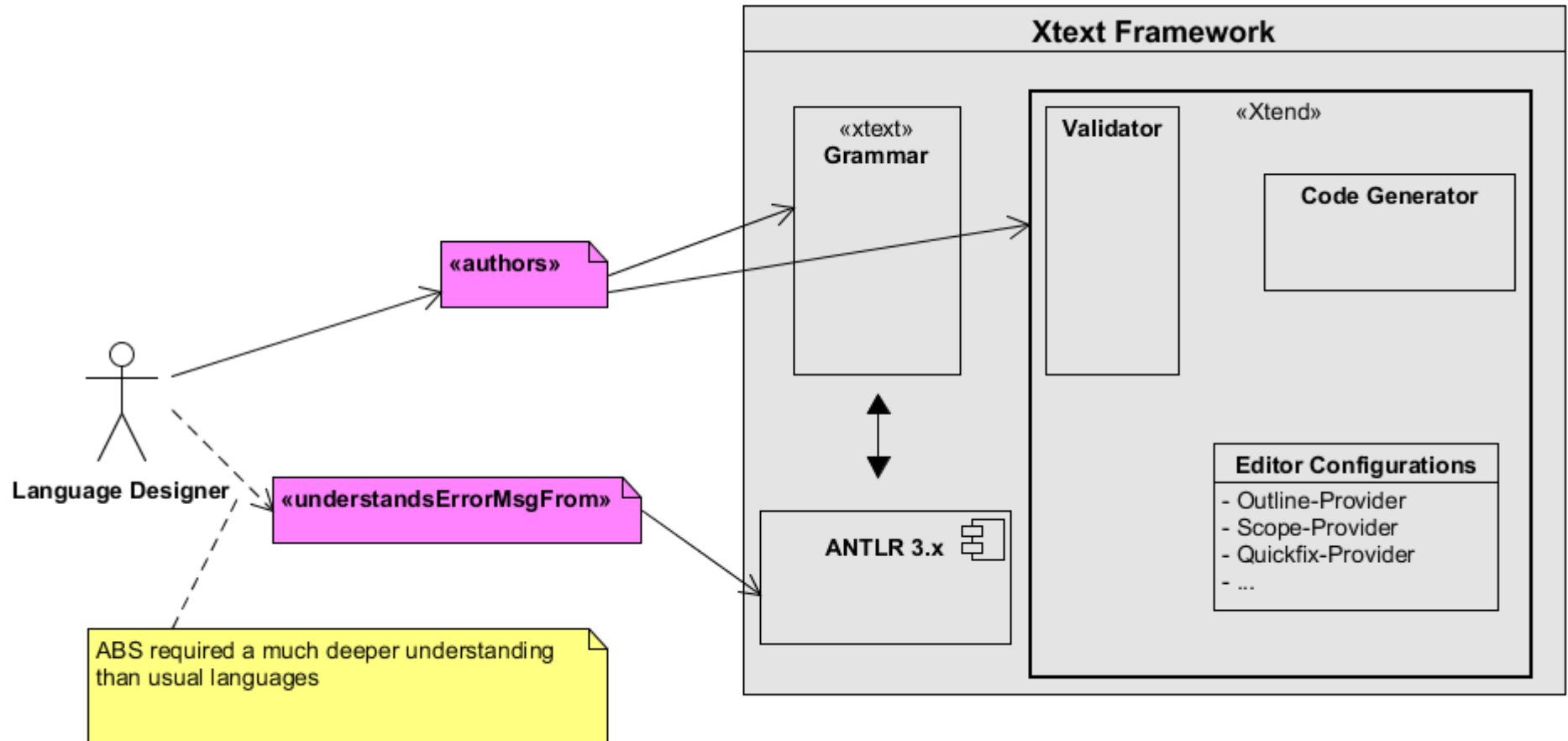
- History of this Talk
- Xtext-Technology
- Wrestling with ABS
  - Language Description
- Examples/Demo
  - Focus on Visualization

# Xtext

- Language Engineering Framework
  - Home: <https://www.eclipse.org/Xtext/>
  - Not only for Eclipse, but also Web-Browsers, LSP\* - Editors
- Focus of Textual Languages
  - Syntax is defined by a grammar + validators
  - Easy access to *Abstract Syntax Tree (AST)*
    - AST can be programatically traversed and analyzed
    - From the AST, other artefacts can be generated

LSP\* – Language Server Protocol

# Xtext Overview



# Defining the grammar Xtext in Action

The screenshot shows the Eclipse IDE interface with the Xtext grammar file `SC1.xtext` open. The Package Explorer on the left shows the project structure, and the Outline view shows the generated grammar elements. The main editor displays the Xtext grammar code:

```
1 grammar exa.sc1.SC1 with org.eclipse.xtext.common.Terminals
2
3 generate sC1 "http://www.sc1.exa/SC1"
4
5 // TODO: split this language in two formally different languages:
6 // The first one without inv-support, the second with inv-support.
7 //
8 StateDiag:
9     vd=VarDecl
10    sd=StateDecl
11    ed=EventDecl
12    td=TransDecl
13    id=InvDecl;
14
15 VarDecl:
16     'vars' ':' vars+=Var*;
17
18 // first state is considered to be start state
19 StateDecl:
```

A red arrow points from the text "Defining the grammar" to the `generate` line in the code.



# Xtext in Action

Rich editor for my DSL

Outline

Resource - PSc1/faulty.sc1 - Eclipse Platform

File Edit Navigate Search Project Run Window Help

Quick Access Resource

Project Expl... v.output v1.sc1 v.myexp faulty.sc1 fixed.sc1

```
vars : collected bill

states: start idle waitingForMoney paid

events: cardInserted coinInserted

transitions :

start => idle / collected = 0 bill = 3

idle => waitingForMoney cardInserted

waitingForMoney => waitingForMoney coinInserted [c

waitingForMoney => paid coinInserted [collected ==
```

Outline

- faulty
  - <unnamed>
    - collected
    - bill
  - <unnamed>
    - start
    - idle
    - waitingForMor
    - paid

Tasks Xtext Syntax Graph Generated Code Problems

0 items

Description	Resource	Path

# Xtext in a Web-Browser

- Technology provided by **dslforge** Demo URL

The screenshot displays the DSL Forge Workbench v0.9.1 running in a web browser. The browser's address bar shows the URL `https://beeone.plugbee.com/workbench`. The interface is divided into several panels:

- Project Explorer:** Shows a tree view of project files, including folders like ANTLR, Arithmetics, Devices, DomainModel, DonaYa, Greetings, MáquinaEstadosTexto, PetriNets, PirateRobot, StateMachine, Statemachines, StaticWeb, and TESTE. Under the PetriNets folder, the file `Examples.petrinet` is selected.
- Code Editor:** Displays the content of `Examples.petrinet`. The code is as follows:

```
1 ptnet "examples" {  
2     place p1  
3     place p2  
4     place p3  
5     transition t1 "Label"  
6     arc a1 p1 -> t1 inhibitor  
7     place p55  
8     arc a2 p55 -> t1 inhibitor  
9     arc name  
10 }  
11
```

A syntax error is highlighted in yellow at line 10, with the message: `10:mismatched input '}' expecting RULE_ID Mismatched input '}' expecting ID`.
- Properties Panel:** Located at the bottom, it shows a table with the following content:

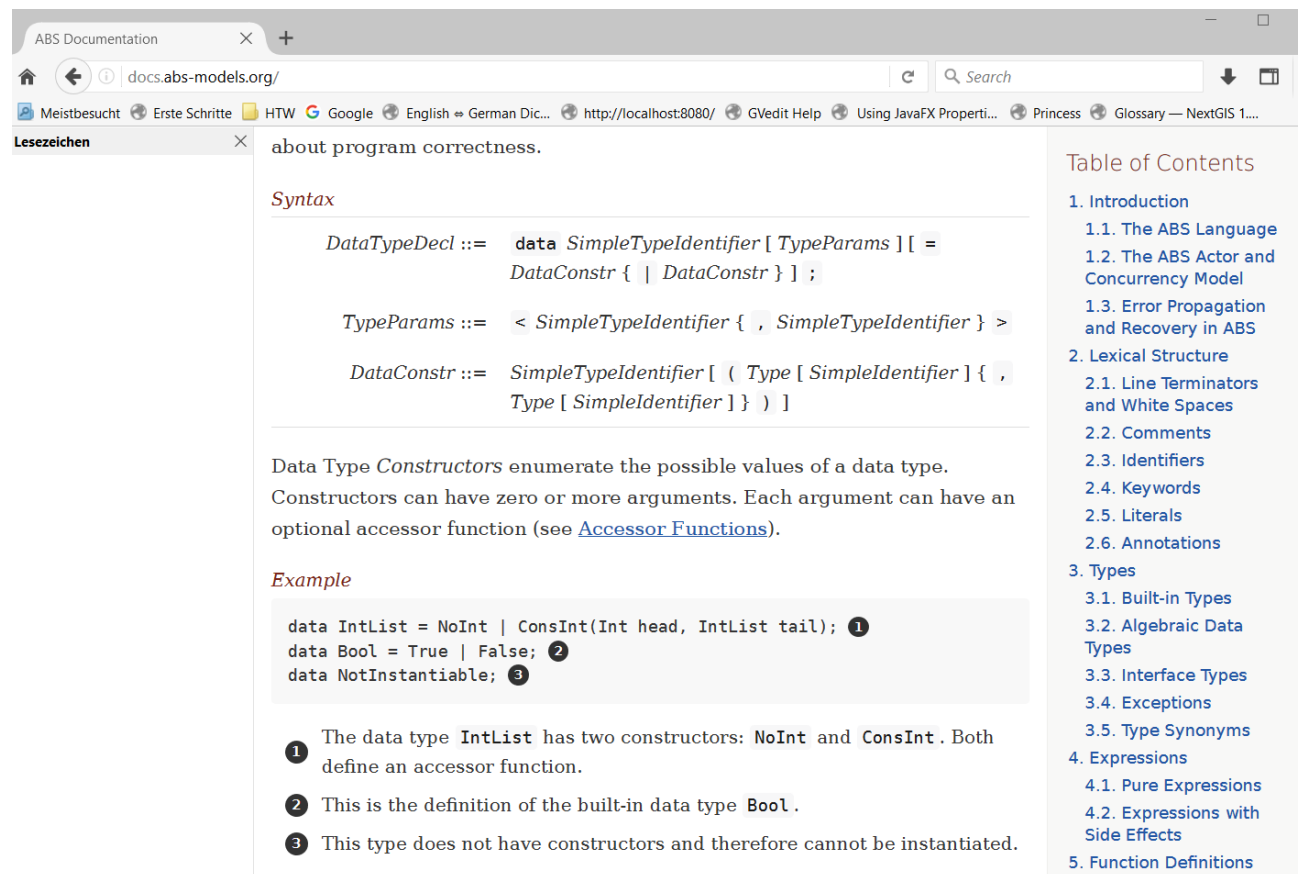
Property	Value
Editable	true

# Outline

- History of this Talk
- Xtext-Technology
- Wrestling with ABS
  - Language Description
- Examples/Demo
  - Focus on Visualization

# ABS Language Description

- My sources:
  - docs.abs-models.org
  - Some input-files



The screenshot shows a web browser window with the URL docs.abs-models.org. The page content includes a search bar, a table of contents on the right, and the main text area. The main text area is titled 'Syntax' and contains the following definitions:

```
DataTypeDecl ::= data SimpleTypeIdentifier [ TypeParams ] [ =  
DataConstr { | DataConstr } ] ;  
  
TypeParams ::= < SimpleTypeIdentifier { , SimpleTypeIdentifier } >  
  
DataConstr ::= SimpleTypeIdentifier [ ( Type [ SimpleIdentifier ] { ,  
Type [ SimpleIdentifier ] } ) ]
```

Below the definitions, there is a paragraph explaining that Data Type Constructors enumerate the possible values of a data type. Constructors can have zero or more arguments. Each argument can have an optional accessor function (see [Accessor Functions](#)).

An example is provided with the following code:

```
data IntList = NoInt | ConsInt(Int head, IntList tail); ❶  
data Bool = True | False; ❷  
data NotInstantiable; ❸
```

Footnotes explain the example:

- ❶ The data type `IntList` has two constructors: `NoInt` and `ConsInt`. Both define an accessor function.
- ❷ This is the definition of the built-in data type `Bool`.
- ❸ This type does not have constructors and therefore cannot be instantiated.

The right sidebar contains a 'Table of Contents' with the following items:

- 1. Introduction
  - 1.1. The ABS Language
  - 1.2. The ABS Actor and Concurrency Model
  - 1.3. Error Propagation and Recovery in ABS
- 2. Lexical Structure
  - 2.1. Line Terminators and White Spaces
  - 2.2. Comments
  - 2.3. Identifiers
  - 2.4. Keywords
  - 2.5. Literals
  - 2.6. Annotations
- 3. Types
  - 3.1. Built-in Types
  - 3.2. Algebraic Data Types
  - 3.3. Interface Types
  - 3.4. Exceptions
  - 3.5. Type Synonyms
- 4. Expressions
  - 4.1. Pure Expressions
  - 4.2. Expressions with Side Effects
- 5. Function Definitions

# Problems when Encoding ABS with Xtext

- Documentation has some (minor) inconsistencies
  - e.g. Same rule under different names
- ABS is not always Java-like
  - e.g. Import clauses

```
module Bar;  
import Drinks.Drink; ①  
import pourMilk from Drinks; ②
```

```
module Bar;  
import * from Drinks; ①
```

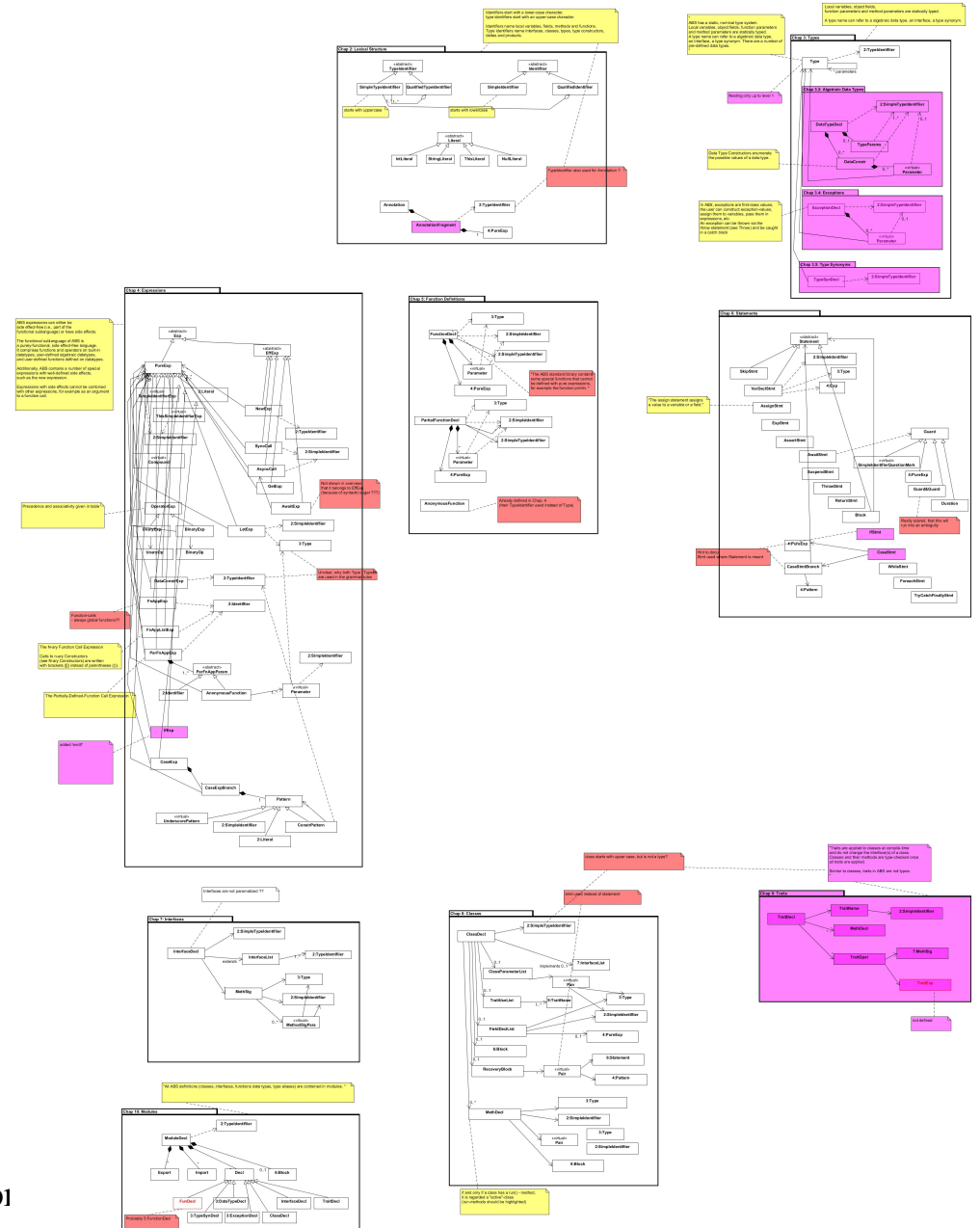
Is it also possible to write

```
import Drinks.*;
```

????

# Problems when Encoding ABS with Xtext

- Sheer size of language definition



# Problems when Encoding ABS with Xtext

- Documented Grammar Rules allow Ambiguous Parsing
  - Requires a lot of effort for Left-Factoring

---

*Exp ::= PureExp | EffExp*

---

*PureExp ::= SimpleIdentifier*  
| **this** . *SimpleIdentifier*  
| **this**  
| **null**  
| *Literal*  
| *LetExp*  
| *DataConstrExp*  
| *FnAppExp*  
| *FnAppListExp*  
| *ParFnAppExp*  
| **IfExp**  
| *CaseExp*  
| *OperatorExp*  
| ( *PureExp* )

---

*IfExp ::= if PureExp then PureExp else PureExp*

---

---

*Statement ::= SkipStmt*  
| *VarDeclStmt*  
| *AssignStmt*  
| **ExpStmt**  
| *AssertStmt*  
| *AwaitStmt*  
| *SuspendStmt*  
| *ThrowStmt*  
| *ReturnStmt*  
| *Block*  
| **IfStmt**  
| *CaseStmt*  
| *WhileStmt*  
| *ForeachStmt*  
| *TryCatchFinallyStmt*

---

---

*IfStmt ::= if ( PureExp ) Stmt [ else*

---

How to parse when Statement is expected???

... if (

*ExpStmt ::= Exp ;*



# Problems when Encoding ABS with Xtext

- Grammar rule reveal only coarsely, what can be referenced
  - Distinction only between Type-/NonType-Identifier
  - Uniqueness-/Scope-rules for identifiers not found

---

```
AssignStmt ::= [ this . ] SimpleIdentifier = Exp ;
```

---

Also Method-Arg allowed ?

```
PureExp ::= SimpleIdentifier  
| this . SimpleIdentifier  
| this  
| null  
| Literal
```

Rather access to Field?

# Outline

- History of this Talk
- Xtext-Technology
- Wrestling with ABS
- Language Description
- Examples/Demo
  - Focus on Visualization

# Demo

## Focus on Visualization

- My Code-Generator generates `.dot-Files` (input for *Graphviz*)
- Visualization at 3 Levels:
  - Abstract Syntax Tree (AST)
  - Program Structure (e.g. Class Diagram)
  - Domain-Specific Visualization
    - Example of meta-programming
    - Works without graphics-library in ABS