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 29th 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

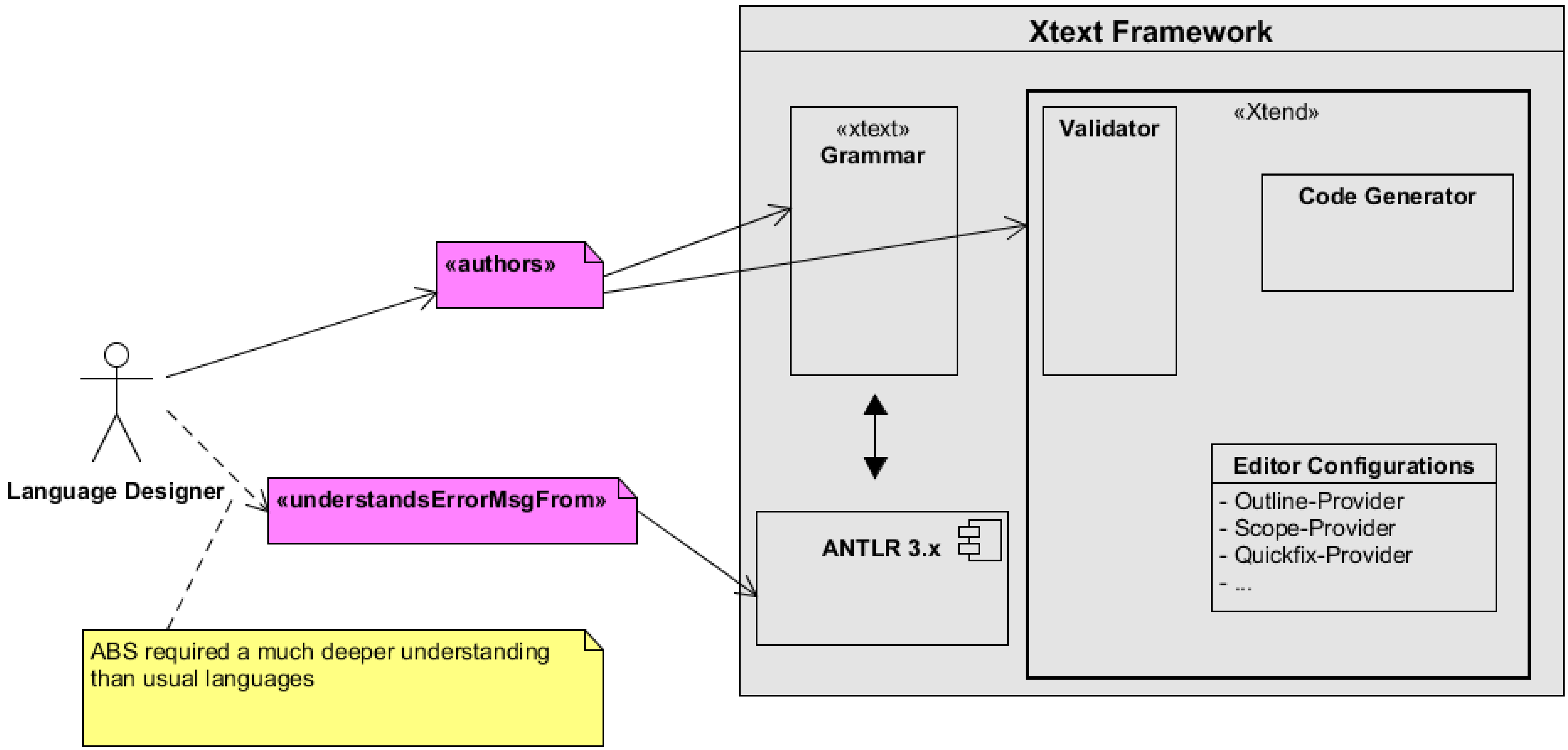
- History of this Talk
- Xtext-Technology
- Wrestling with ABS
  - Language Description
- Examples/Demo
  - Focus on Visualization

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

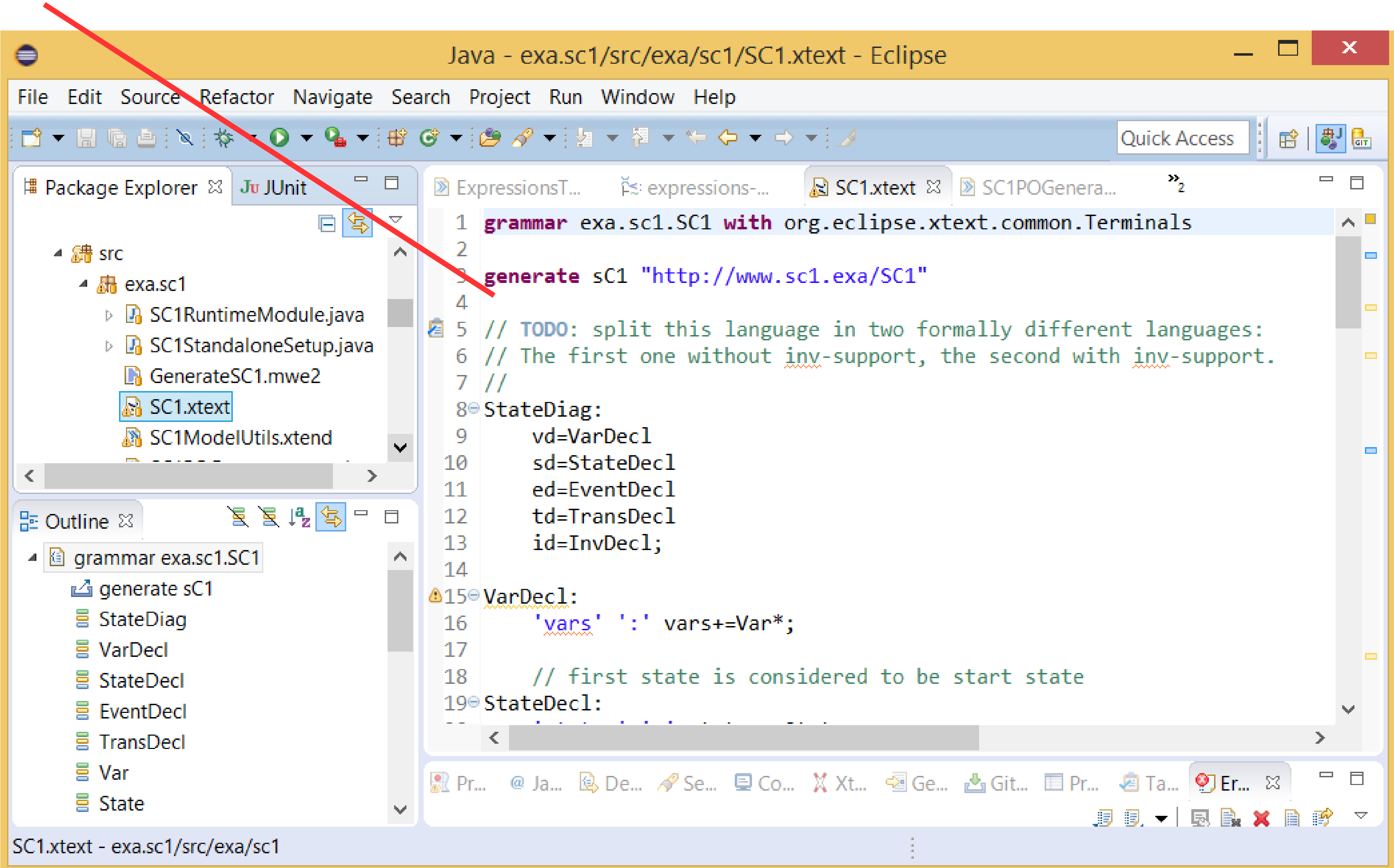
Xtext Overview



ABS required a much deeper understanding than usual languages

Xtext in Action

Defining the grammar

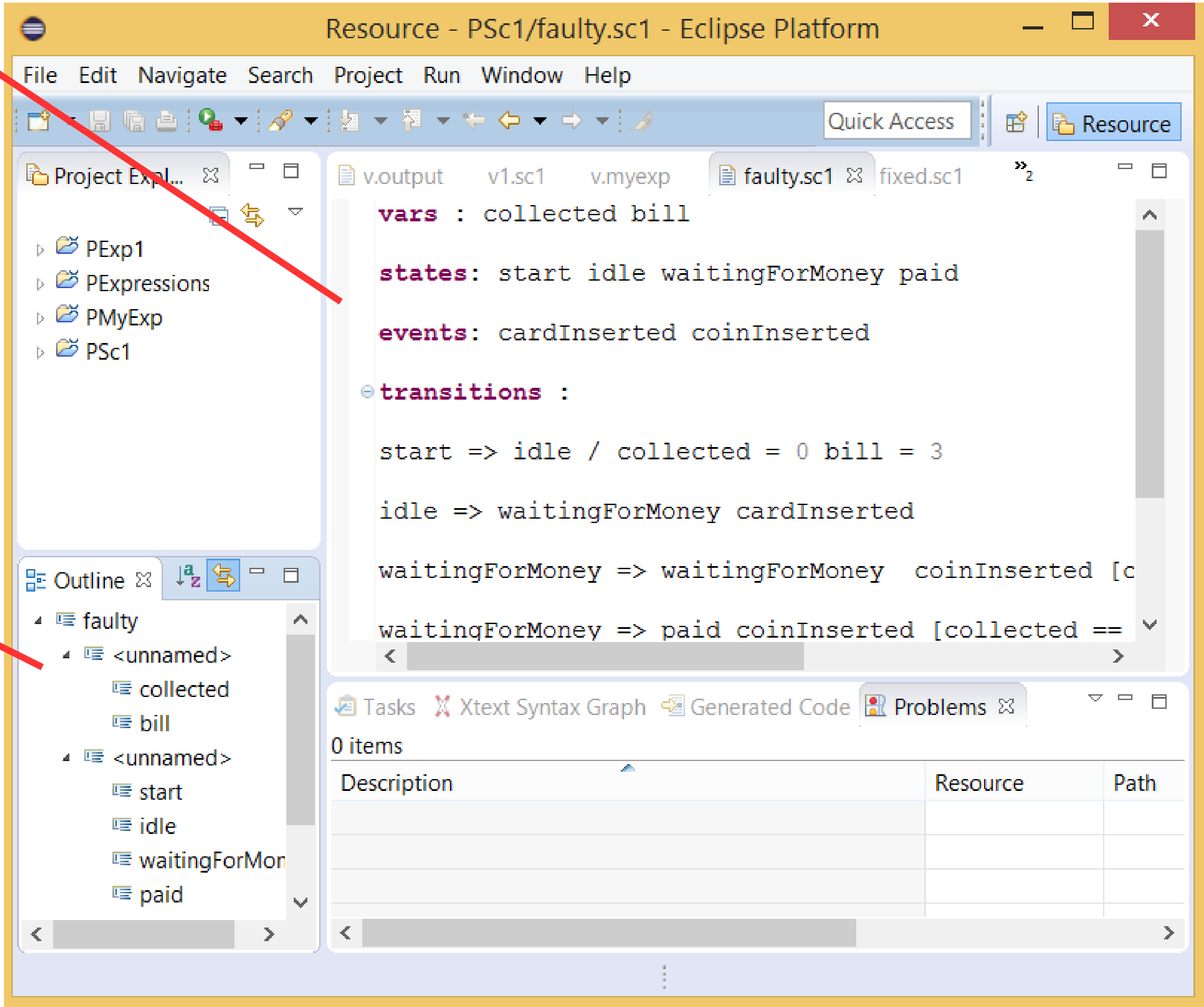


```
grammar exa.sc1.SC1 with org.eclipse.xtext.common.Terminals

generate sC1 "http://www.sc1.exa/SC1"

// TODO: split this language in two formally different languages:
// The first one without inv-support, the second with inv-support.
//
StateDiag:
    vd=VarDecl
    sd=StateDecl
    ed=EventDecl
    td=TransDecl
    id=InvDecl;

VarDecl:
    'vars' ':' vars+=Var*;

    // first state is considered to be start state
StateDecl:
```

Xtext in Action

Rich editor for my DSL

Outline

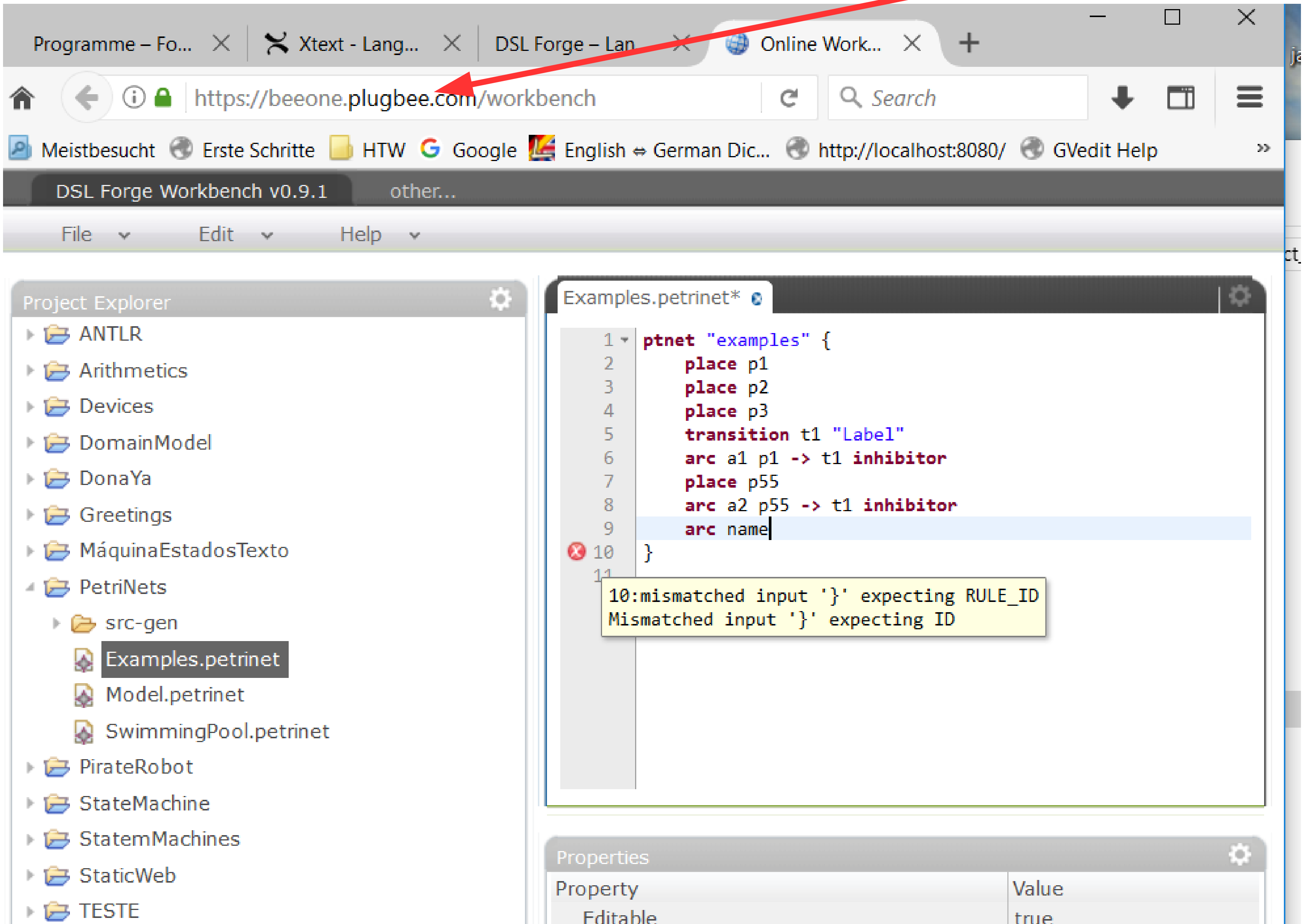


```
vars : collected bill

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 ==
```

Xtext in a Web-Browser

- Technology provided by dslforge

Demo URL



```
ptnet "examples" {
    place p1
    place p2
    place p3
    transition t1 "Label"
    arc a1 p1 -> t1 inhibitor
    place p55
    arc a2 p55 -> t1 inhibitor
    arc name
}
```

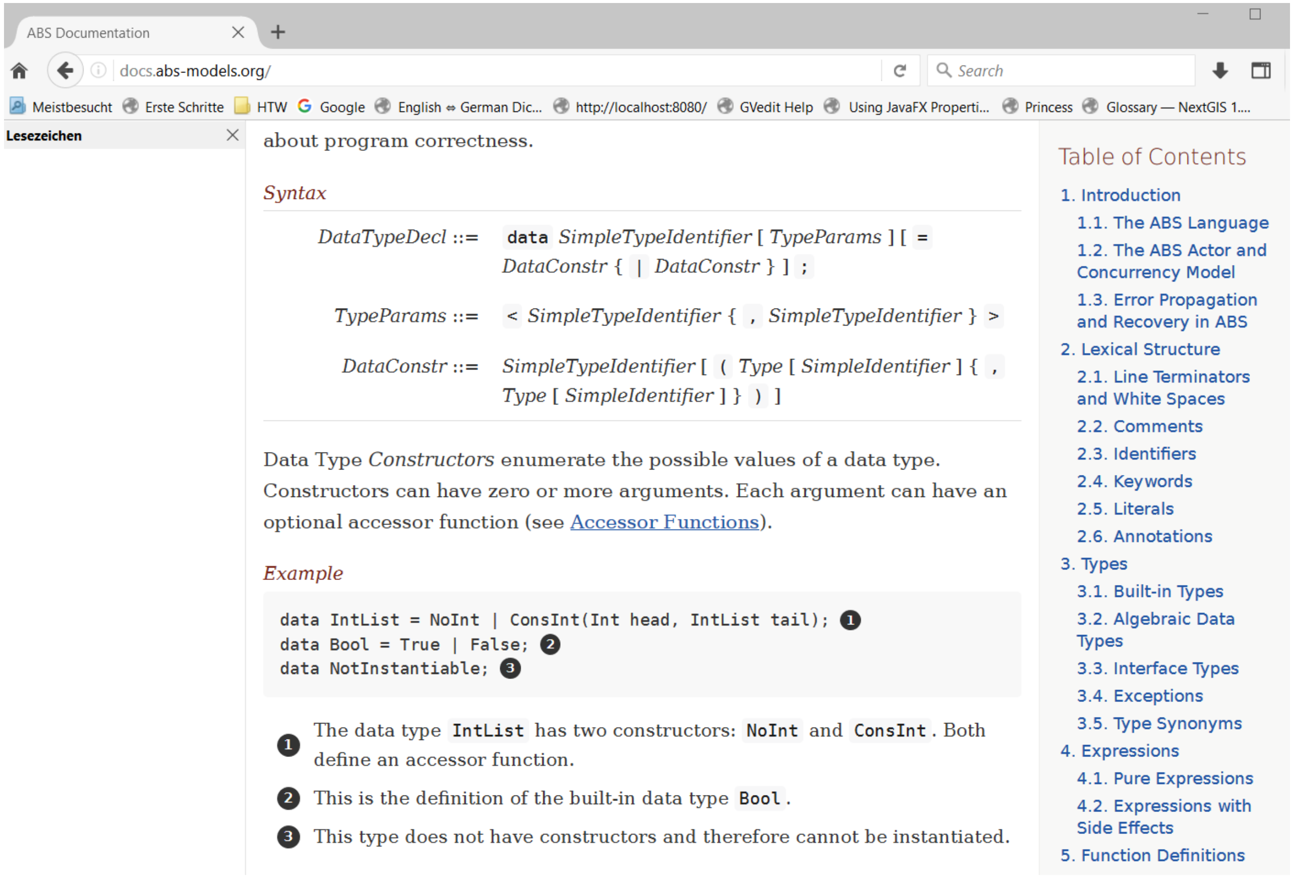
```
10:mismatched input '}' expecting RULE_ID
Mismatched input '}' expecting ID
```

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



about program correctness.

Syntax

DataTypeDecl ::= `data` *SimpleTypeIdentifier* [*TypeParams*] [`=` *DataConstr* { `|` *DataConstr* }] `;`

TypeParams ::= `<` *SimpleTypeIdentifier* { `,` *SimpleTypeIdentifier* } `>`

DataConstr ::= *SimpleTypeIdentifier* [`(` *Type* [*SimpleIdentifier*] { `,` *Type* [*SimpleIdentifier*] } `)`]

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).

Example

```
data IntList = NoInt | ConsInt(Int head, IntList tail); 1
data Bool = True | False; 2
data NotInstantiable; 3
```

1 The data type `IntList` has two constructors: `NoInt` and `ConsInt`. Both define an accessor function.

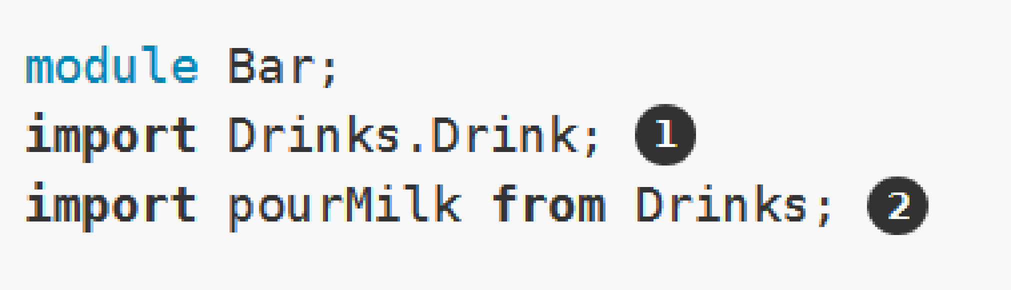
2 This is the definition of the built-in data type `Bool`.

3 This type does not have constructors and therefore cannot be instantiated.

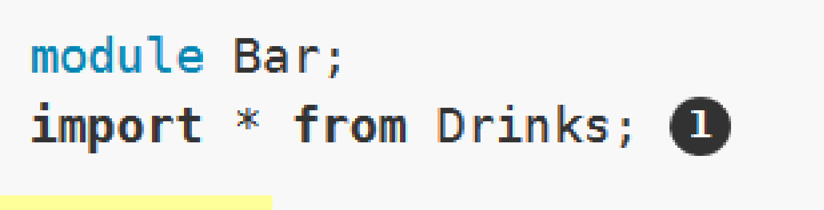
Table of Contents

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; ❶
import pourMilk from Drinks; ❷
```



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

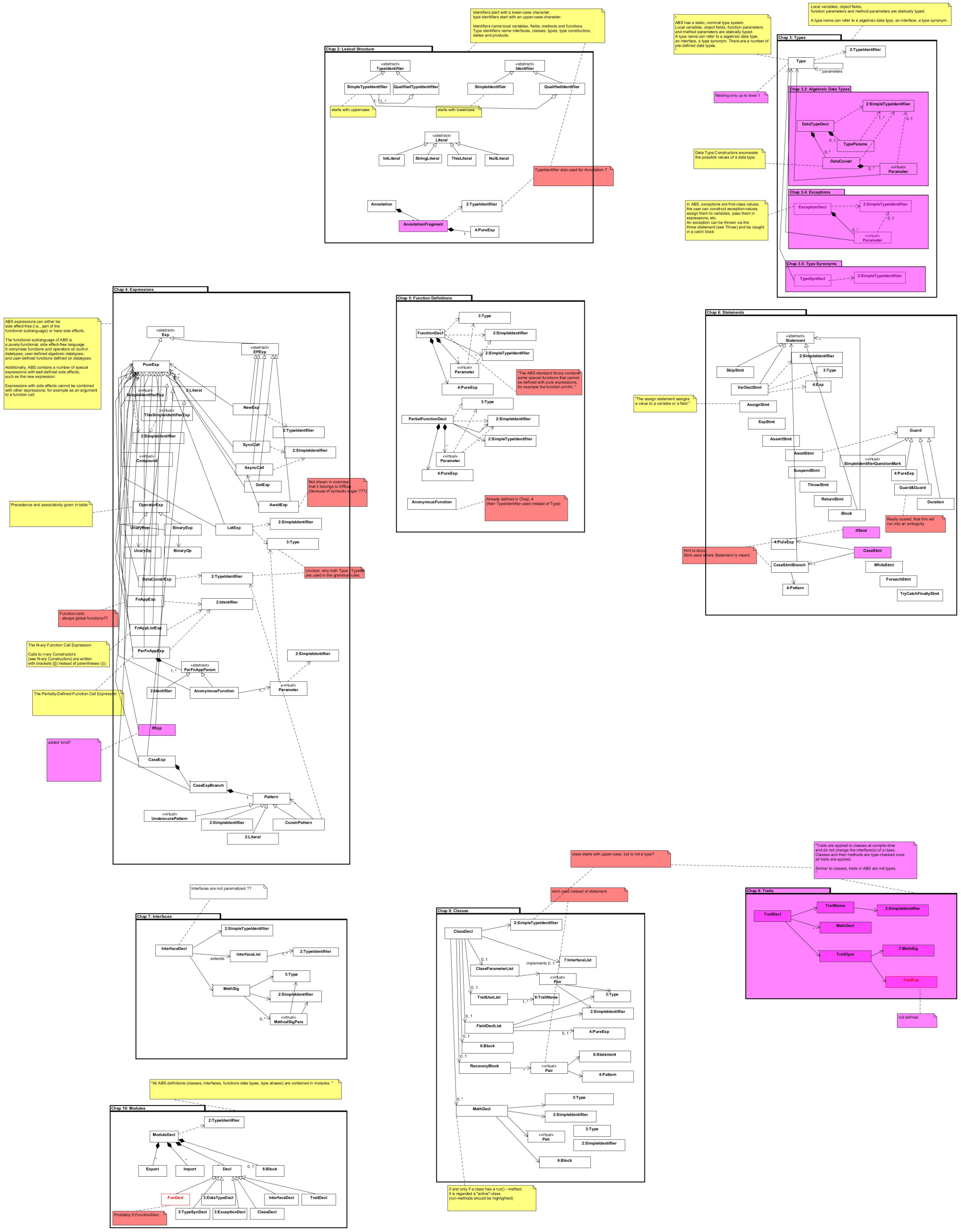
Is it also possible to write

```
    import Drinks.*;
```

????

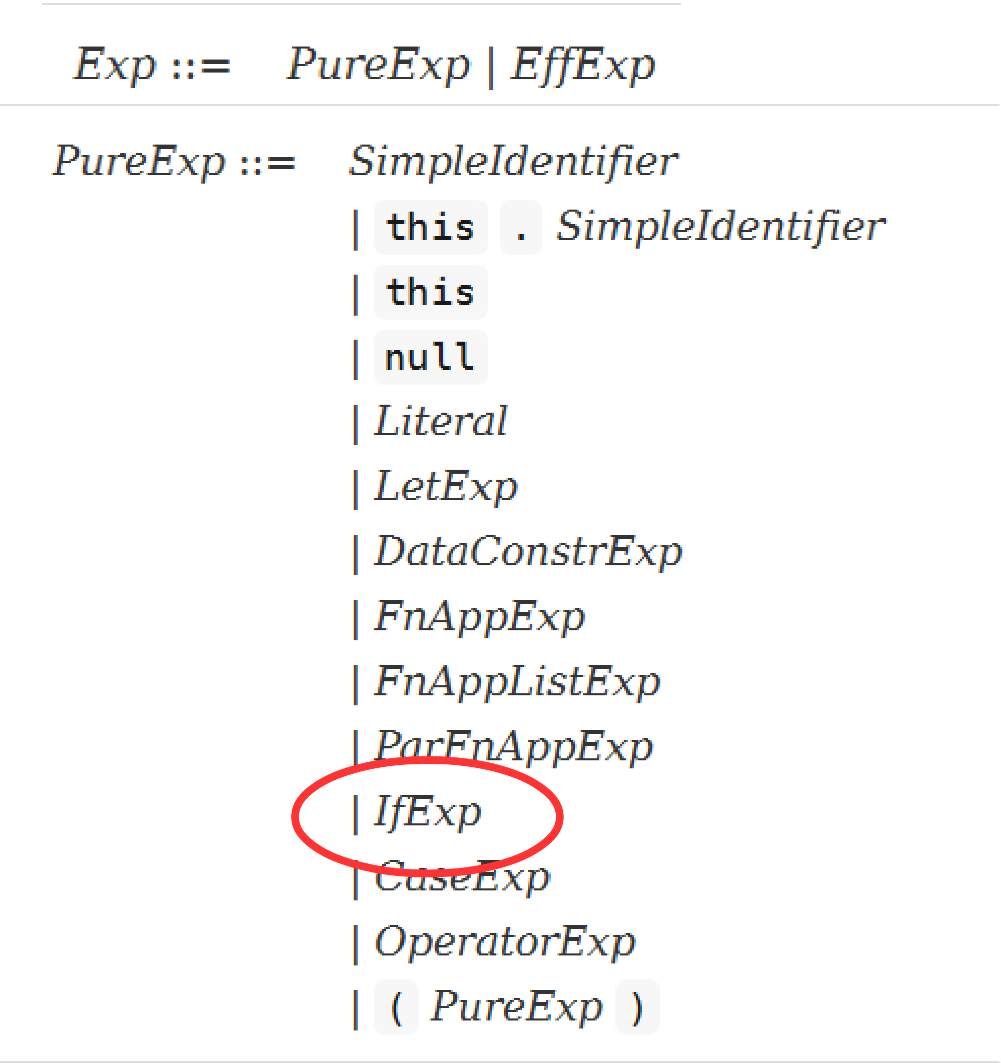
Problems when Encoding ABS with Xtext

- Sheer size of language definition



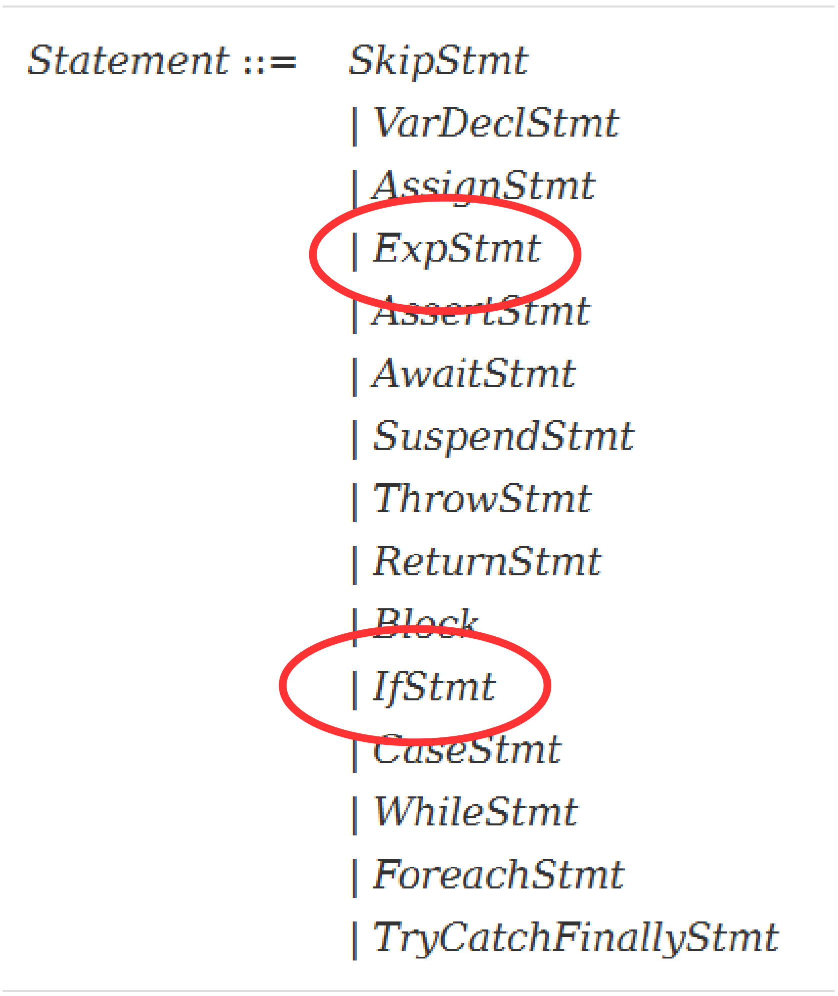
Problems when Encoding ABS with Xtext

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

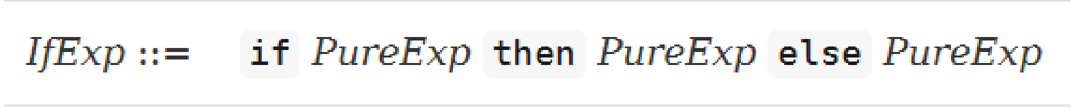


Exp ::= *PureExp* | *EffExp*

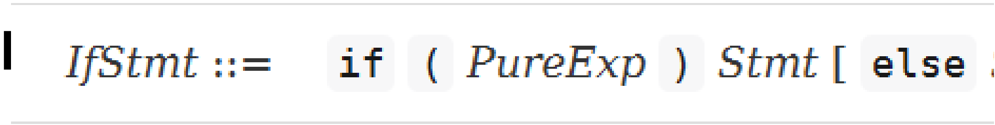
PureExp ::= *SimpleIdentifier*
| `this` `.` *SimpleIdentifier*
| `this`
| `null`
| *Literal*
| *LetExp*
| *DataConstrExp*
| *FnAppExp*
| *FnAppListExp*
| *ParFnAppExp*
| *IfExp*
| *CaseExp*
| *OperatorExp*
| `(` *PureExp* `)`



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



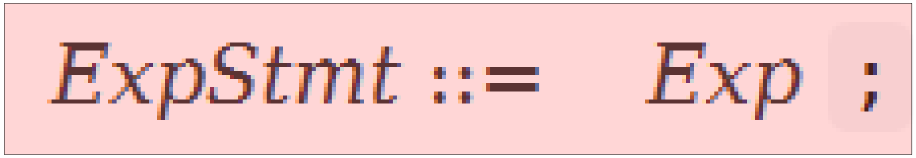
IfExp ::= `if` *PureExp* `then` *PureExp* `else` *PureExp*



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

How to parse when `Statement` is expected???

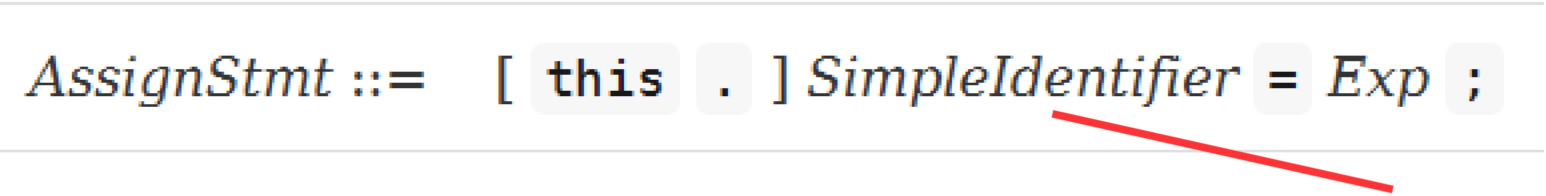
```
... if (
```



ExpStmt ::= *Exp* `;`

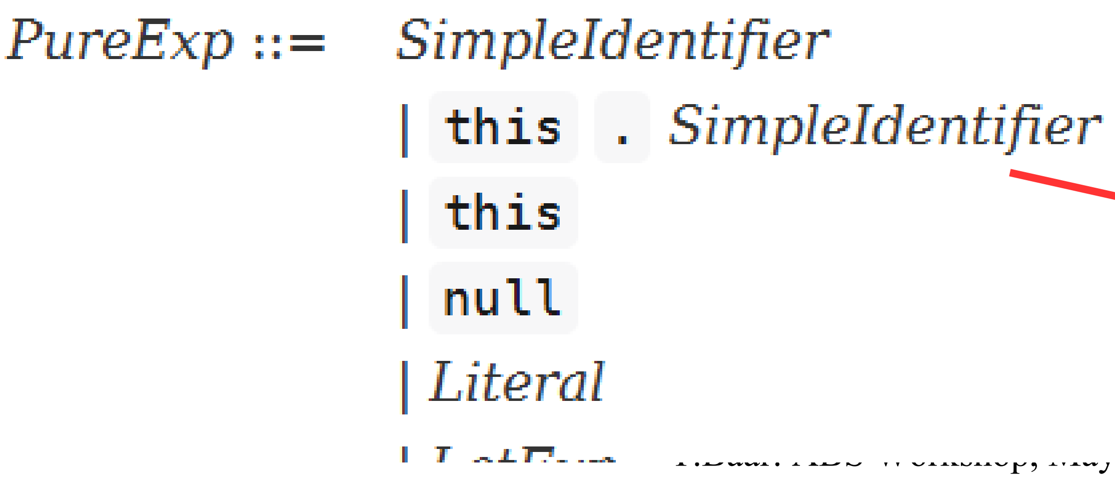
Problems when Encoding ABS with Xtext

- 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
            | LetExp
```

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