

Domänenspezifische Sprachen

Anwendung und die Zukunft, wo stehen wir?

09.02.2023
Informatik Workshop (IWS)
Hochschule Mannheim, Wintersemester 2022

Das Team



Baris Kacmaz



Joshua Krcmar



Jörg Vierling



Niklas Büchner



Shayan Mohajerani

Agenda

- Kontext
- Motivation
- Kleine Pause [15 min]
- Interne DSLs
- Große Pause [45 min] (12.00 bis 12:45)
- Externe DSLs
- Kleine Pause [15 min]
- Abgrenzung von DSLs zu anderen Lösungen
- Wrap up
- Ausblick auf die Zukunft der DSLs
- Diskussionsrunde

Kontext

- Problemszenario
- Praxis 1: JS

Problemszenario

- Wir sind das Logistikunternehmen “Parcel R Us”
 - Liefern primär Pakete regional aus
 - Wir agieren aber auch international
- Rasant steigendes Wachstums der Firma fordert stetige Änderungen der Software
 - Die Verteilung der Lieferungen basiert auf in Software definierten Regeln
 - Regeln müssen mehrere Ebenen der Entscheidung durchlaufen
 - Neue Produkte bzw. Regionen müssen im Ablauf angepasst werden
 - Anpassungen an den Regeln darf nicht zu Fehlern führen
 - Verfügbarkeit 24/7

Praxis 1

- **Aufgaben 1 bis 3**
- Sie sind für die Umsetzung verantwortlich
- Das Unternehmen setzt auf JavaScript zur Visualisierung der Verteilung
- Fangen Sie an!
- GitHub-Repository: <https://github.com/niklasbuechner/iws>

Gesamtdauer: ca. 30 min

Motivation

- Wie waren die Erfahrungen?
- Was sind Domänen?
- Domänenspezifische Sprachen (DSL)
- Unterscheidung von DSLs
- Wie kann eine DSL uns weiterhelfen?

Was sind Domänen?

- Domänen = Wissensgebiete
 - Gebiet menschlichen Wissens, auf dem wissenschaftliche Kenntnisse vorliegen
- Themen und Begriffe eines Wissensgebietes
 - Begriffe spiegeln sich oft in Objekten wider
 - Beziehungen bestimmen die Funktionalität zwischen Objekten
- Einschlägige Fachsprache
 - spezifischen Anforderungen eines Wissensgebietes
- Experten definieren die Anforderungen

Domänenspezifische Sprachen (DSL)

- “Domain-specific language (noun): a computer programming language of limited expressiveness focused on a particular domain”
 - ~ Martin Fowler, 2010
- Spezielle (Programmier-)Sprache
 - Definiert für spezifische Domäne
 - Arbeit innerhalb der Domäne vereinfachen
 - Implementierung konzentriert sich auch auf ein Problem der Domäne
 - Auf einen Anwendungsfall zugeschnitten
 - z. B. Beschreibung von Regeln innerhalb des Logistikunternehmens
- Langzeiteffekt
 - Produktivitätssteigerung der domänentypischen Arbeit

Unterscheidung von DSLs

- Abhängig vom Anwendungsbereich
- Abstrahierende DSLs
 - Vereinfachung der Entwicklung durch Entkopplung von technischen Details (z. B. der Plattform)
 - Abstrahierung der Entwicklung
 - Oft für Prototyping / Plattformunabhängigen Entwicklung verwendet
 - Verwender: Domänenspezifische Experte
- Technische DSLs
 - Zur Vereinfachten Handhabung technische Aspekte
 - Sowie weitere Aufgaben wie Code-Dokumentierung, Testing, Datenaustausch etc
 - Verwender: Entwickler

Abstrahierende DSLs

- Zielgruppe: Domänen-Experten
- Bsp: Person aus Marketing benötigt Informationen aus der Datenbank
- Ohne DSL
 - keine SQL-Kenntnisse
 - Benötigt jemanden, der das für diese Person macht
 - Kann aufgrund von Interpretation der natürlichen Sprache falsch verstanden werden
 - Kann dauern bis man gefunden hat was man sucht
- Mit DSL
 - Middle Man mit SQL-Kenntnissen fällt weg
 - Marketing-Person kann genau das eingeben, was gesucht war

Technische DSLs

- Zielgruppe: Entwickler
- Beziehen sich auf einen technischen Aspekt statt auf Domänen-Anwendungsfall
- Beispiele:
 - HTML
 - CSS
 - SQL
 - XML
 - RegEx

Wie kann eine DSL uns weiterhelfen?

- Speziell bei der Entwicklung von wissenschaftlicher Software
 - Programmlösung auf verschiedenen interdisziplinären Stufen
 - Jede DSL erfüllt einen Zweck
- Im Fall einer fachlichen DSL:
 - Schnittstelle zwischen Domänen-Experten und Software
 - Beispiel Matlab
- Im Fall einer technischen DSL:
 - Definierte Sprache zur Konstruktion einer Software
 - Beispiel SQL

Interne DSLs

- Was ist eine interne DSL?
- Beispiele
- Vor- und Nachteile von internen DSLs
- Praxis 2: Verwendung einer internen DSL
- Wie implementiert man interne DSLs?
- Beispiele
- Praxis 3: Implementieren einer internen DSL

Was ist eine interne DSL?

- Programmiersprache dient als Host
- Auf die Syntax des Hosts eingeschränkt
- Die Host-Sprache muss oft “verbogen” werden oder es müssen Tricks anwenden, um eine flüssige Sprache zu erhalten
 - Meta-Programming
- Benötigen keinen Compiler an sich, jedoch ist eine Parsing-Layer trotzdem notwendig
- “embedded” DSLs
- fluent interfaces

Beispiele



```
1 String sql = "select id, name " +
              "from customers c, order o " +
              "where " +
              "c.since >= sysdate - 30 and " +
              "sum(o.total) > " + significantTotal + " and " +
              "c.id = o.customer_id and " +
              "nvl(c.status, 'DROPPED') != 'DROPPED'";
```

Beispiele



```
1 Table c = CUSTOMER.alias();
2 Table o = ORDER.alias();
3 Clause recent = c.SINCE.laterThan(daysEarlier(30));
4 Clause hasSignificantOrders = o.TOTAT.sum().isAbove(significantTotal);
5 Clause ordersMatch = c.ID.matches(o.CUSTOMER_ID);
6 Clause activeCustomer = c.STATUS.isNotNullOr("DROPPED");
7 String sql = CUSTOMERS.where(recent.and(hasSignificantOrders)
8                         .and(ordersMatch)
9                         .and(activeCustomer)
10                        .select(c.ID, c.NAME)
11                        .sql());
```

Beispiele



```
1 weather at: @time.iso8601 do
2   description @description
3   temperature "#{@temp} C"
4   wind do
5     velocity "#{@wind_vel} kts"
6     direction @wind_direction
7   end
8 end
```

Beispiele



XML

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <weather at="2016-11-29T22:54:15+11:00">
3   <description>Bright & sunny.</description>
4   <temperature>18.3 C</temperature>
5   <wind>
6     <velocity>14 kts</velocity>
7     <direction>SSE</direction>
8   </wind>
9 </weather>
```

Beispiele



```
1 <xs:simpleType name="monatInt">
2   <xs:restriction base="xs:integer">
3     <xs:minInclusive value="1"/>
4     <xs:maxInclusive value="12"/>
5   </xs:restriction>
6 </xs:simpleType>
7 <xs:simpleType name="monate">
8   <xs:list itemType="monatInt"/>
9 </xs:simpleType>
10
11 <monate>
12   1 2 3 4 5 6 7 8 9 10 11 12
13 </monate>
```

Beispiele



```
1 add_product do
2   name 'Charger'
3   description 'Life saving'
4   price 19.99
5 end
6
7 add_product do
8   name 'Lockpick'
9   description 'Door shall not close'
10  price 7.50
11 end
```

Vor- und Nachteile von internen DSLs

Vorteile

- Verwendung der Host-Sprache
- Einfache Integration in bestehende Systeme
- Geringerer Entwicklungsaufwand
- Kein expliziter Parser notwendig

Nachteile

- IDE-Support
- Syntax-Limitierungen
 - Je nach Sprache
- Limitierung der Ausdrucksfähigkeit
- Performance-Overhead

Praxis 2: Verwendung einer internen DSL

- Aufgaben 4 bis 6
 - Zusatzaufgaben 7 bis 9
- Sie sind für die Umsetzung verantwortlich
- Das Unternehmen setzt auf eine eigene interne DSL
- Die interne DSL ist im Cheat Sheet beschrieben
- Fangen Sie an!

Gesamtdauer: ca. 40 min

Wie implementiert man interne DSLs?

- Unterschiedlich in jeder Programmiersprache
- Man muss ein Sprachkonstrukt finden, dass sich dafür eignet

```
Object.defineProperty(window, 'bulky', {  
    get: () => currentRule.addBulkyCondition(true),  
    set: (isBulky) => {  
        if (!isBulky) {  
            return  
        }  
  
        currentRule.addBulkyCondition(true)  
    },  
});
```

Beispiele

```
import com.example.html.* // see declarations below

fun result() =
    html {
        head {
            title {+"XML encoding with Kotlin"}
        }
        body {
            h1 {+"XML encoding with Kotlin"}
            p  {+"this format can be used as an alternative markup to XML'"

                // an element with attributes and text content
                a(href = "https://kotlinlang.org") {+"Kotlin"}
            }
        }
    }
}
```

```
SomeClass.configure do
    foo "#{bar}_baz"
    bar "hello"
end
```

```
SomeClass.config.foo
=> "_baz"
```

Beispiele

```
cal = new Calendar();
cal.add("DSL tutorial")
    .on(2009, 11, 8)
    .from("09:00")
    .to("16:00")
    .at ("Aarhus Music Hall")
;

cal.add("Making use of Patterns")
    .on(2009, 10, 5)
    .from("14:15")
    .to("15:45")
    .at("Aarhus Music Hall")
;
```

```
computer();
processor();
cores(2);
speed(2500);
i386();
disk();
size(150);
disk();
size(75);
speed(7200);
sata();
```

Praxis 3: Implementieren einer internen DSL

```
function createInternalDsl() {  
    rule(1);  
    letter();  
    country("Deutschland");  
    rule(4);  
    parcel();  
    country("Deutschland");  
    rule(20);  
    catchAll();  
}
```

- Erstellt die Funktionen für die interne DSL.
- Die Funktionen sollen das globale Array *rules* befüllen.
- Die *Rule*-Klasse beinhaltet Factory-Methoden, um die Konditionen hinzuzufügen.

Praxis 3: Implementieren einer internen DSL

- **Aufgabe 10**
- Erstellen Sie Ihre eigene interne DSL.
- Das Interface der internen DSL ist in der Funktion `createInternalDsl` schon vorgegeben.
- Implementieren Sie die darin genutzten Funktionen.
 - Ziel: Das Array `rules` soll durch die interne DSL gefüllt werden.
- Implementieren Sie die Evaluierungsfunktion `getAccountFromRules`.

Gesamtdauer: ca. 40 min

Externe DSLs

- Was ist eine externe DSL?
- Beispiele
- Vor- und Nachteile von externen DSLs
- Praxis 4: Verwendung einer externen DSL
- Wie implementiert man externe DSLs?
- Exkurs: Parsing einer Programmiersprache
- Lexen einer externen DSL
- Parsen einer externen DSL
- Praxis 5: Implementieren einer externen DSL

Was ist eine externe DSL?

- Unabhängig von Hostsprachen
- Besitzt eigene Syntax
- Externe Datei mit beliebigem Dateiformat
- Benötigt Parser zur Übersetzung

Was ist eine externe DSL?



Erhöhte Lernkurve & Komplexere Implementierung

- Lernen der Syntax
- Lernen von Parsern, Grammatik und Parser-Generatoren

Beispiele

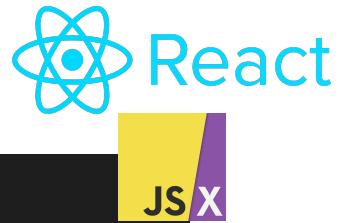


```
1  <!DOCTYPE html>
2  <html>
3    <head>
4      <title>Titel der Seite</title>
5      <link rel="stylesheet" href="css-example.css" />
6    </head>
7    <body>
8      <h1 class="header">Dies ist eine Überschrift</h1>
9      <p class="paragraph">Dies ist ein Paragraph</p>
10     <ul>
11       <li>Kaffee</li>
12       <li>Tee</li>
13       <li>Milch</li>
14     </ul>
15   </body>
16 </html>
17
```



```
1  body {
2    background-color: aqua;
3  }
4
5  .header {
6    margin: 10px;
7    font-size: 30px;
8    font-weight: bold;
9    color: red;
10 }
11
12 .paragraph {
13   padding: 20px;
14   font-size: 16px;
15   color: grey;
16 }
```

Beispiele



```
1 const JsxExample = () => {
2   const myVar = "World";
3
4   return (
5     <div key="div1" onClick={() => console.log("div1 clicked")}>
6       <h1>Hello {myVar}!</h1>
7       <h2>React ist cool!</h2>
8     </div>
9   );
10 };
11
```

BABEL



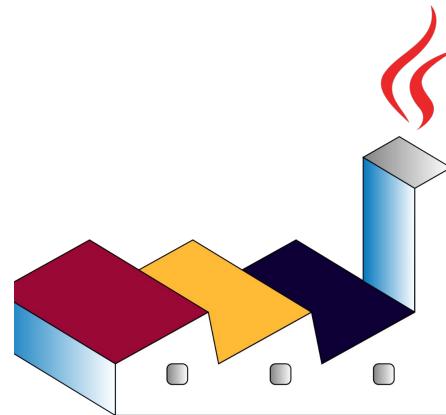
```
1 const JsxExample = () => {
2   const myVar = "World";
3
4   return React.createElement(
5     "div",
6     {
7       key: "div1",
8       onClick: () => console.log("div1 clicked"),
9     },
10    React.createElement("h1", null, "Hello ", myVar, "!"),
11    React.createElement("h2", null, "React ist cool!")
12  );
13};
14
```

Beispiele



```
1 CREATE TABLE Personen (
2     id int,
3     nachname varchar(255),
4     vorname varchar(255),
5     eMail varchar(255),
6     familienstand varchar(255)
7 );
```

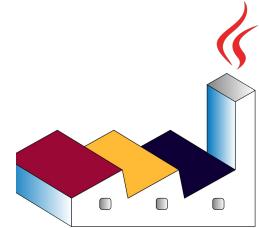
Beispiele



[PlantUML](#) Hands-on

Beispiele

Auch via VSCode & Extension problemlos möglich:



```

1 @startuml PlantUML-Example
2 class Spedition {
3   +handelsregisternummer: String
4   name: String
5 }
6
7
8 class Mitarbeiter{
9   mitarbeiternummer: Integer
10  vorname: String
11  nachname: String
12  geburtsdatum: Date
13 }
14 note left: Das ist ein Mitarbeiter
15
16
17 class Lagerist{
18   warenAnnehmen()
19   warenAusgeben()
20 }
21
22 class Adresse {
23   strasse: String
24   hausnummer: String
25   plz: String
26   ort: String
27   land: String
28 }

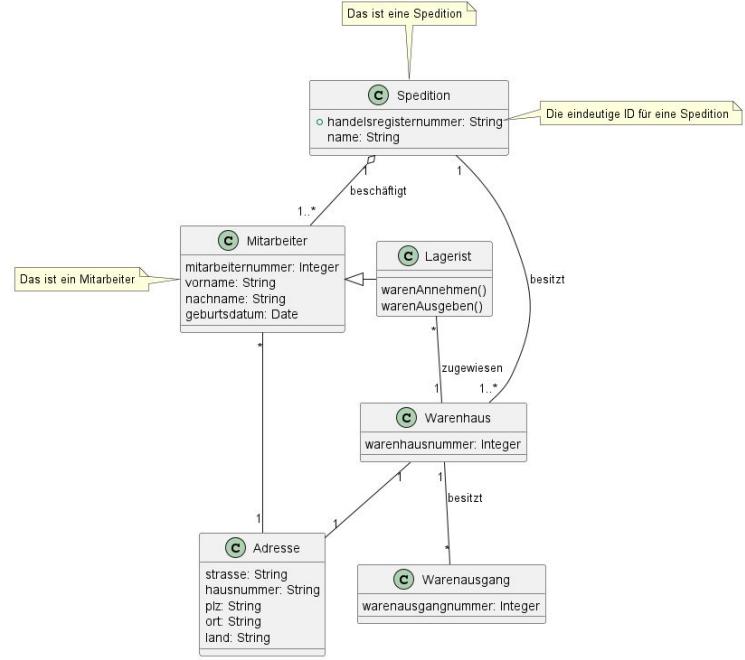
29
30 class Warenhaus{
31   warenhausnummer: Integer
32 }
33
34
35 class Warenausgang{
36   warenausgangnummer: Integer
37 }
38
39
40 Spedition "1" -- "1..*" Warenhaus : besitzt
41
42 Spedition "1" o-- "1..*" Mitarbeiter : beschäftigt
43
44 Warenhaus "1" -- "*" Warenausgang : besitzt
45
46 Mitarbeiter "*" -- "1" Adresse
47
48 Warenhaus "1" -- "1" Adresse
49
50 Mitarbeiter <|- Lagerist
51
52 Lagerist "*" -- "1" Warenhaus : zugewiesen
53
54 note top of Spedition: Das ist eine Spedition
55
56 note right of Spedition::handelsregisternummer
57 Die eindeutige ID für eine Spedition
58 end note
59 @enduml

```

```

29
30 class Warenhaus{
31   warenhausnummer: Integer
32 }
33
34
35 class Warenausgang{
36   warenausgangnummer: Integer
37 }
38
39
40 Spedition "1" -- "1..*" Warenhaus : besitzt
41
42 Spedition "1" o-- "1..*" Mitarbeiter : beschäftigt
43
44 Warenhaus "1" -- "*" Warenausgang : besitzt
45
46 Mitarbeiter "*" -- "1" Adresse
47
48 Warenhaus "1" -- "1" Adresse
49
50 Mitarbeiter <|- Lagerist
51
52 Lagerist "*" -- "1" Warenhaus : zugewiesen
53
54 note top of Spedition: Das ist eine Spedition
55
56 note right of Spedition::handelsregisternummer
57 Die eindeutige ID für eine Spedition
58 end note
59 @enduml

```



Beispiele

LATEX



```
1 \documentclass[11pt,conference,compsoc,final,a4paper]{IEEEtran}
2
3 \usepackage{siunitx}
4 \usepackage{svg}
5
6 \newcommand{\dokumententitel}{Demonstrationsexperiment}
7
8 \input{preamble} % Weitere Einstellungen aus einer anderen Datei lesen
9
10 \begin{document}
11
12 \title{\dokumententitel}
13
14 \maketitle
15
16 \begin{abstract}
17 Dies ist ein Abstract...
18 \end{abstract}
19
20 \section{Einleitung}
21 Dies ist eine Einleitung...
22
23 \subsection{Unterkapitel}
24 Dies ist ein Unterkapitel mit Aufzählung:
25
26 \begin{itemize}
27   \item 1. Punkt
28   \item 2. Punkt
29   \item 3. Punkt
30 \end{itemize}
31
32 \end{document}
```



Demonstrationsexperiment

Zusammenfassung—Dies ist ein Abstract...

1. Einleitung

Dies ist eine Einleitung...

1.1. Unterkapitel

Dies ist ein Unterkapitel mit Aufzählung:

- 1. Punkt
- 2. Punkt
- 3. Punkt

Beispiele



```
 1 # IWS: Ein Markdown-Beispiel
 2
 3 Dies ist ein Markdown-Beispiel, das für **IWS** erstellt wurde.
 4
 5 ## Was ist Markdown?
 6
 7 'Markdown' ist eine einfache Auszeichnungssprache, die es Benutzern ermöglicht,
 8 Texte in einer leicht lesbaren Formatierung zu schreiben,
 9 die im Anschluss daran in _HTML_ umgewandelt werden können.
10
```



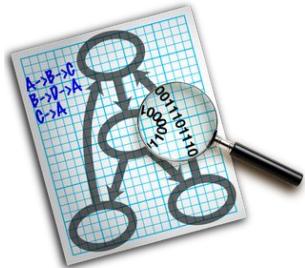
IWS: Ein Markdown-Beispiel

Dies ist ein Markdown-Beispiel, das für IWS erstellt wurde.

Was ist Markdown?

Markdown ist eine einfache Auszeichnungssprache, die es Benutzern ermöglicht, Texte in einer leicht lesbaren Formatierung zu schreiben, die im Anschluss daran in HTML umgewandelt werden können.

Beispiele



Graphviz (DOT)



(.*)

YACC (Yet Another Compiler Compiler)

uvm.!

Vor- und Nachteile von externen DSLs

Vorteile

- Bessere Lesbarkeit und Ausdrucksstärke
- Erhöhte Produktivität
- Bessere Wartbarkeit
- Mehr Flexibilität im Vgl. zu internen DSLs
- Volle Kontrolle über die Syntax und Struktur
- Mögliche Integration in bestehende Tools und Technologien

Nachteile

- Erhöhte Lernkurve
- Erhöhter Entwicklungsaufwand
- Begrenzte Werkzeuge

Praxis 4: Verwendung einer externen DSL

- **Aufgabe 11**
- Sie sind für die Umsetzung verantwortlich
- Das Unternehmen setzt nun auf eine eigene externe DSL
- Die externe DSL ist im Cheat Sheet beschrieben
- Fangen Sie an!

Gesamtdauer: ca. 20 min

Wie implementiert man externe DSLs?

1. Lexer verarbeitet die Eingabe
2. Parser baut daraus einen Abstract Syntax Tree
3. Die DSL wird ausgeführt
 - a. Ausführen in einer VM (Bsp.: SQL)
 - b. Evaluieren in einer Rule Engine (Bsp.: Matlab)
 - c. Konfigurieren eines Prozesses (Bsp.: Gradle)

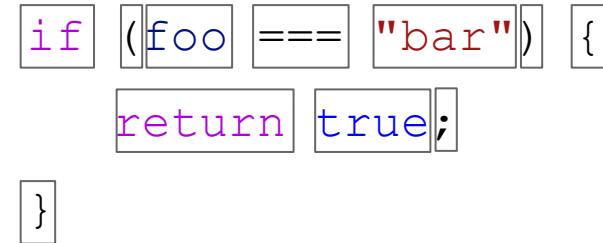
Exkurs: Parsing einer Programmiersprache



Exkurs: Parsing einer Programmiersprache

```
if (foo === "bar") {  
    return true;  
}
```

Lexer
→



If Token
Open Parenthesis Token
Label Token
Triple Equals Token
String Token
Closing Parenthesis Token
...

Exkurs: Parsing einer Programmiersprache

If Token
Open Parenthesis Token
Label Token
Triple Equals Token
String Token
Closing Parenthesis Token
Opening Bracket Token
Return Token
True Token
Closing Bracket Token



Abstract Syntax Tree

```
IF {  
    condition: {  
        VARIABLE "foo" EQUALS "bar"  
    }  
    successPath: {  
        RETURN {  
            VALUE {  
                true  
            }  
        }  
    }  
    failurePath: {}  
}
```

Lexen einer externen DSL

Österreich -> 20

Brief -> 1

Paket -> 4

.split("\n")



Österreich -> 20

Brief -> 1

Paket -> 4

.split("->")



Österreich 20

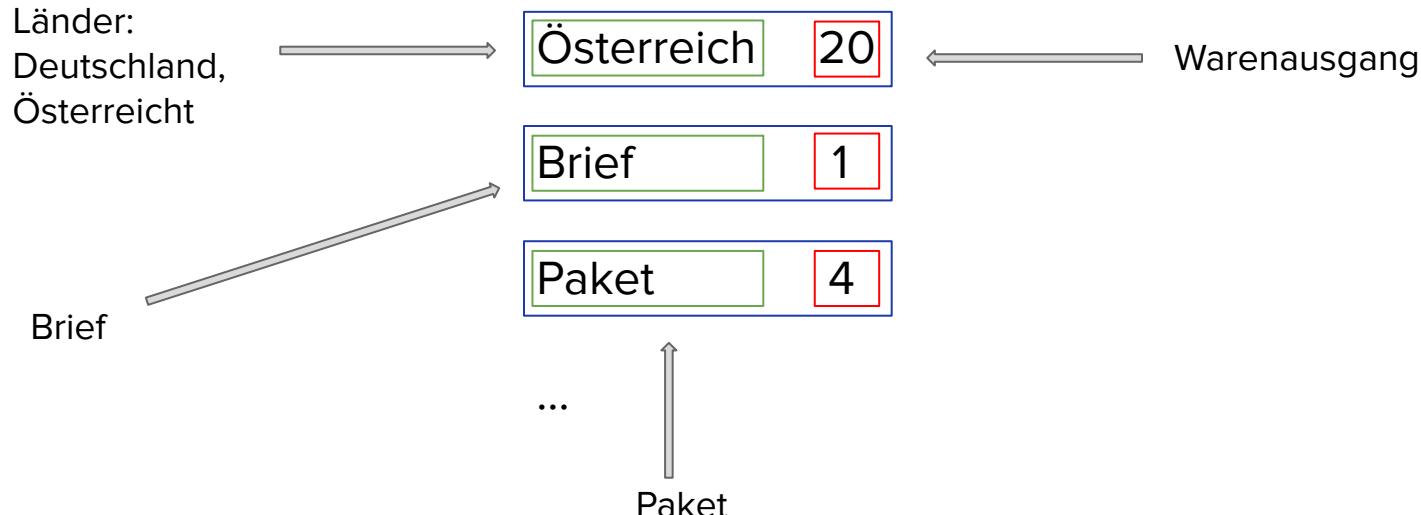
Brief 1

Paket 4

...

...

Parse einer externen DSL



Parse einer externen DSL

```
[  
  [ "Österreich", "20" ],  
  [ "Brief", "11" ],  
  [ "Paket", "12" ],  
]
```

Praxis 5: Implementieren einer externen DSL

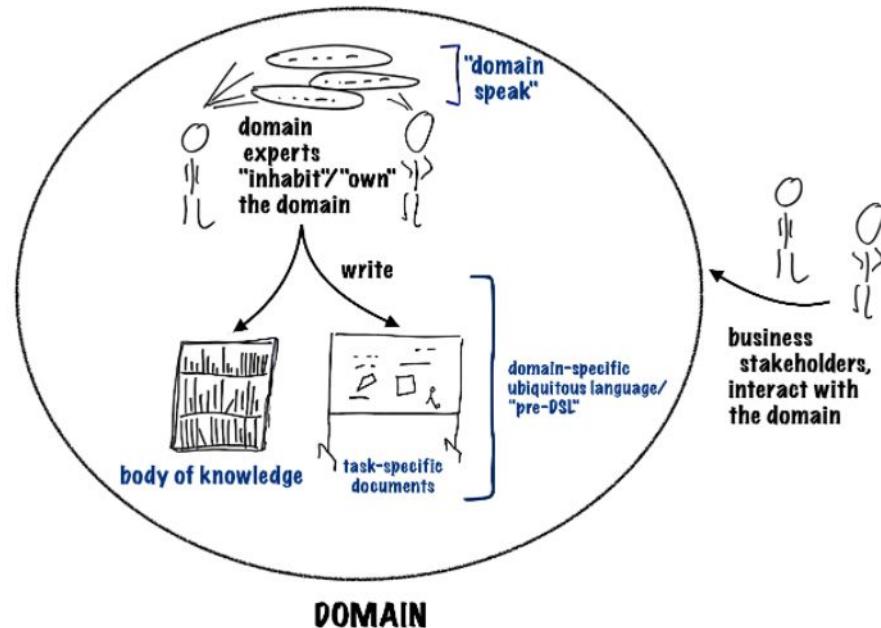
- **Aufgabe 12**
- Lexen
 - Nach Zeilen splitten
 - Zeilen nach Pfeil splitten => Token
 - Token Brief
 - Token Paket
 - Token Österreich
 - Numerische Token => Warenausgang
- Parser
 - Rule Klasse in “js/rule.js”
 - Warenausgang übernehmen
 - Einzelne Konditionen erkennen und übernehmen
- Evaluieren
 - Rule Klasse besitzt eine “test” Methode, die zurückgibt, ob die Regel hier greift.

Gesamtdauer: ca. 40 min

Abgrenzung von DSLs zu anderen Lösungen

DSLs

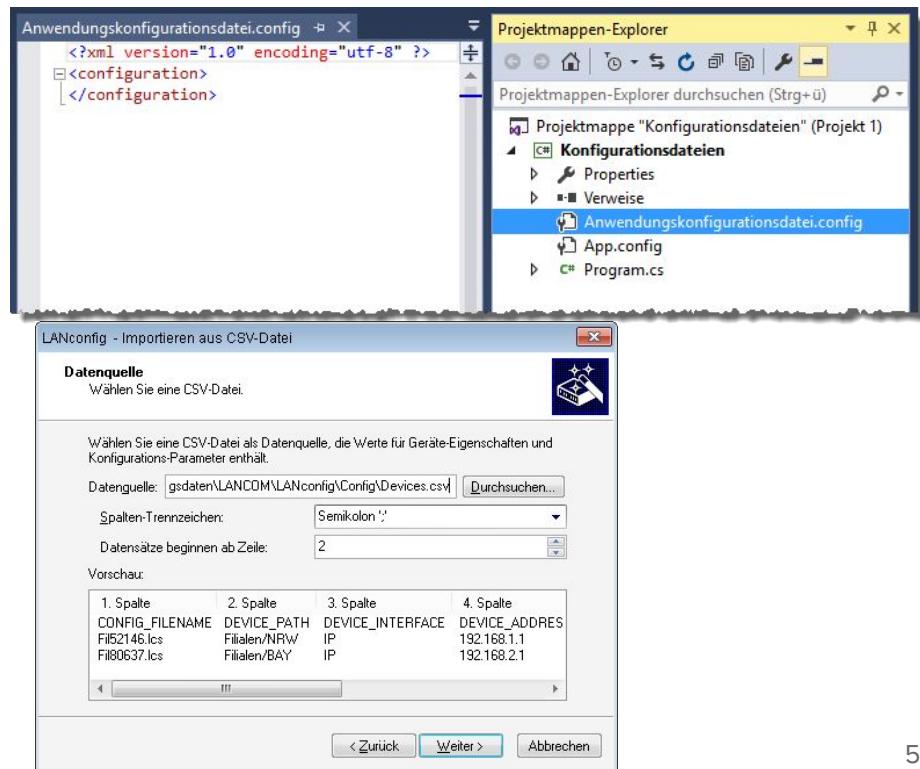
- Spezialisierte Sprache
 - Externe DSL: Eigene Syntax und Semantik
- Entwickelt für bestimmte Domäne
- Höhere Abstraktionsstufe und leistungsstärker
- Idee → eine spezifische Domäne und deren Probleme abdecken



Abgrenzung zu Konfigurationsdateien

Konfigurationsdatei

- Dienen zur Festlegung von Einstellungen/Optionen z. B.
 - Funktionsweise von Anwendungen
- Beschreiben Einstellungen einfach und verständlich
 - Als einfache Texte oder Key-Value-Paare
- Enthalten normalerweise Text im bestimmten Format
 - Z. B. XML, JSON
 - Können auch durch Eingabemasken konfiguriert werden
- Idee → Anpassung von Umgebungsvariablen in Anwendungen



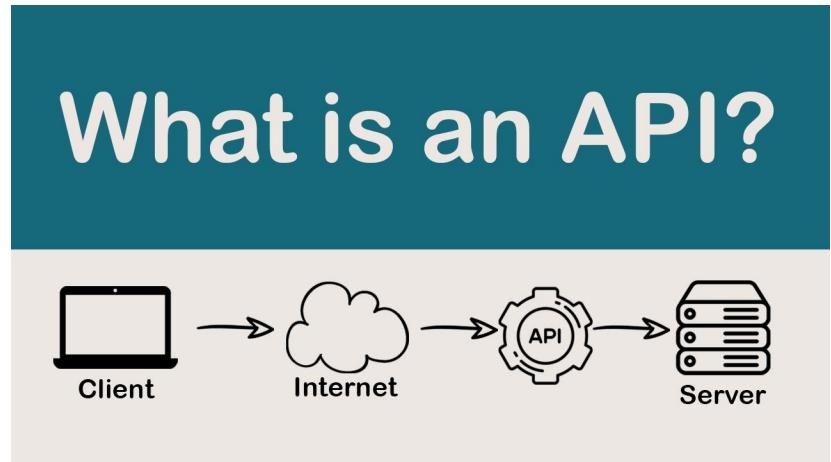
Abgrenzung zu Frameworks

- Vorgefertigte Struktur für die Entwicklung von Anwendungen
 - Sammlung von Klassen, Bibliotheken und Tools
- Wiederkehrende Aufgaben können schneller abgewickelt werden
- Idee → Stellen eine kohärente Struktur für spezifische Anforderungen bereit



Abgrenzung zu APIs

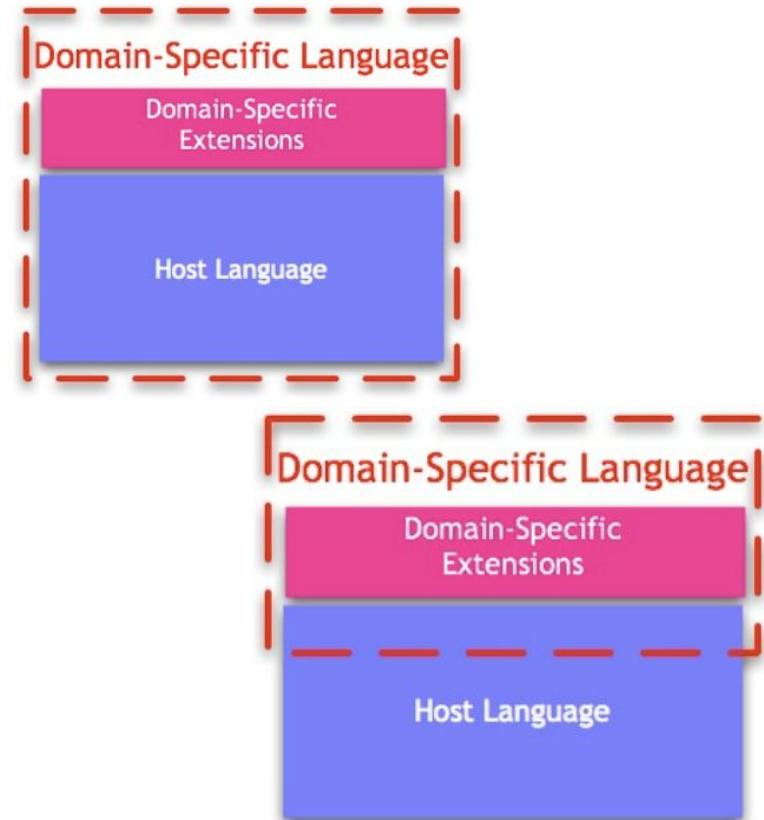
- Sind Schnittstellen zum programmgesteuerten Zugriff auf Funktionalitäten von Anwendungen
- Stellen eine Reihe von definierten Funktionen und Methoden zur Systeminteraktion bereit
- Idee → Ein Befehl entgegenzunehmen, an ein System zu überbringen und anschließend die Antwort zurückzubringen



Wrap up

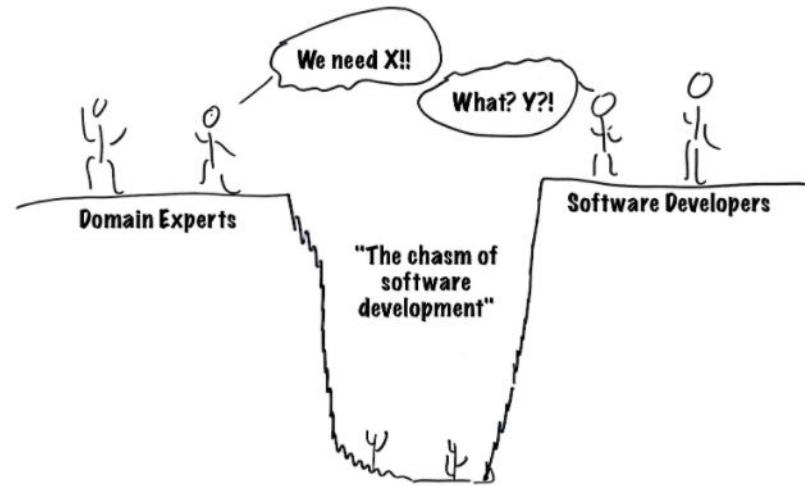
Wrap up

- Interne DSLs werden innerhalb einer allgemeinen Programmiersprache definiert
 - Einfach zu integrieren
 - Können einfach mit einer allgemeinen Programmiersprache verwendet werden
 - Performance der GPL kann genutzt werden
 - Geringere Lernkurve
- Externe DSLs werden als separate Sprachen definiert
 - Erfordert eigene Laufzeitumgebung
 - Benötigen meistens eigenen Compiler/Interpreter
 - Möglicherweise geringere Performance
 - Höhere Flexibilität



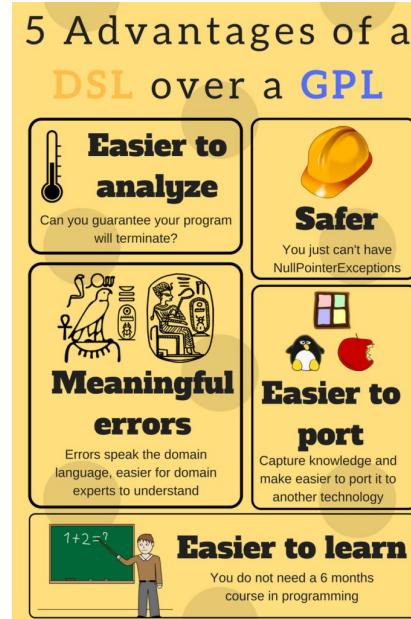
Wrap up

- DSLs sind auf eine spezifische Domäne oder einen bestimmten Zweck abgestimmt
 - Allgemeine Programmiersprachen für eine breite Palette von Anwendungen und Zwecken
- DSLs sind oft einfacher und intuitiver zu verwenden als allgemeine Programmiersprachen



Wrap up

- DSLs können als separate Sprachen oder als Teil einer allgemeinen Programmiersprache implementiert werden
 - Allgemeine Programmiersprachen sind in der Regel als eigenständige Sprachen implementiert
- Das Error-Handling kann auf die Domain angestimmt werden
 - Dadurch sicherer zur Nutzung
- Können einfacher erlernt werden



e_124 [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_71.jk

```
main" java.lang.NullPointerException
at homework.AVLTree_INC.leftRotate(Homework_five_124.java:175)
at homework.AVLTree_INC.insert(Homework_five_124.java:179)
at homework.AVLTree_INC.insert(Homework_five_124.java:163)
at homework.AVLTree_INC.insert(Homework_five_124.java:163)
at homework.AVLTree_INC.insert(Homework_five_124.java:161)
at homework.AVLTree_INC.insert(Homework_five_124.java:161)
at homework.AVLTree_INC.insert(Homework_five_124.java:163)
at homework.AVLTree_INC.insert(Homework_five_124.java:163)
at homework.AVLTree_INC.insert(Homework_five_124.java:161)
at homework.HomeworkFive_124.main(Homework_five_124.java:517)
```

Ausblick auf die Zukunft der DSLs

Ausblick auf die Zukunft der DSLs

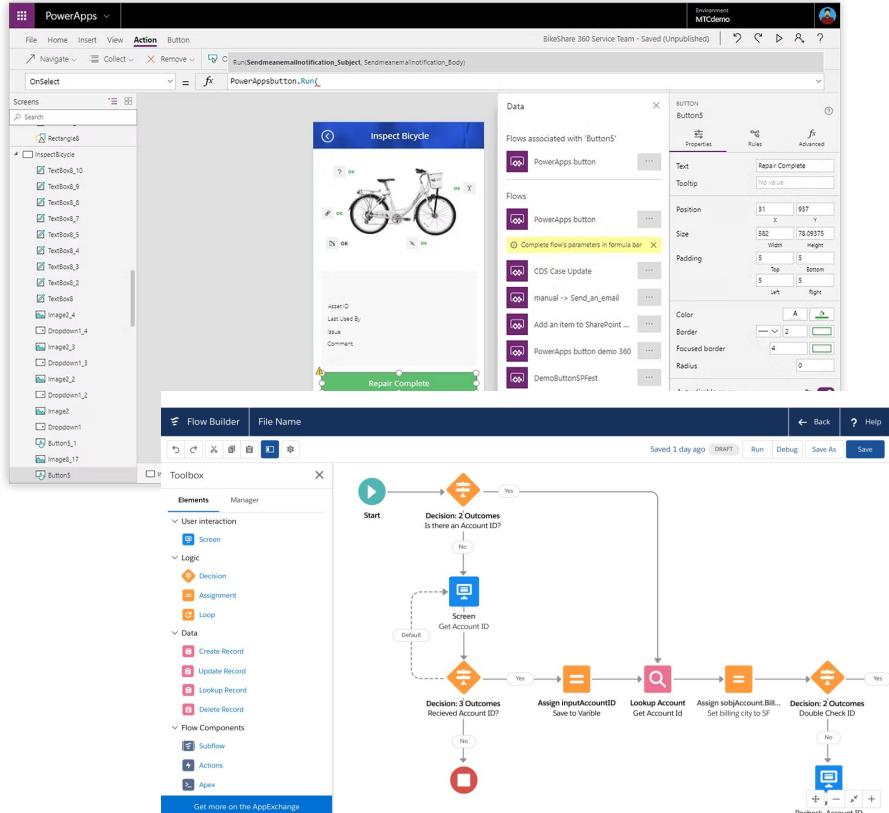
- Benötigen weiterhin eine effiziente und intuitiv verständliche Art der Sprache
 - Erhöhte Produktivität der Benutzer
 - Komplexe Aufgaben können simpler ausgedrückt werden
- Zunahme an Domains
 - Größere Terminologie einer bestimmten Domäne
 - Erleichtert Arbeit für Fachexperten
- Verstärkte Zusammenarbeit
 - Zwischen Entwicklern und Fachexperten

```
bmi := 38;  
  
if bmi is less than 30 then  
    write "Normalgewicht";  
elseif bmi is less than 35 then  
    write "Adipositas I";  
elseif bmi is less than 40 then  
    write "Adipositas II";  
else  
    write "Adipositas III";  
endif;
```

potassium IS WITHIN 3.5 TO 5 |
potassium >= 3.5 AND potassium <= 5

Ausblick auf die Zukunft der DSLs

- Zunahme an Programmiersprachen
 - Mehr Werkzeuge und Technologien, die miteinander integriert werden
- Zuwachs an Low-Code-Plattform Technologien
 - Anwendungen können ohne viel technisches Wissen entwickelt werden
 - Z. B. PowerApps, Salesforce



Diskussionsrunde

Diskussionsrunde



Diskussionsrunde

1. Wie sehen Sie die Zukunft der DSLs in den nächsten 5-10 Jahren?
2. Wie können DSLs dazu beitragen, den Übergang zu einer vernetzten und automatisierten Welt zu unterstützen?
3. Welche neuen Technologien werden die Entwicklung von DSLs beeinflussen?
4. Wie können DSLs dazu beitragen, die Barrieren bei der Verwendung von technischen Systemen zu reduzieren?

**Vielen Dank für die
Aufmerksamkeit!**
