Freie Universität Berlin

Erster Meilenstein
SWP Übersetzerbau Sommer 2010

Institut für Informatik
FU Berlin

Agenda

- Aufgaben des Projekts
- Vorbereitung
  - Anforderungsanalyse / -definition
- Planung
  - Aufgabenteilung
  - Modularisierung
- Implementierung
  - Dokumentation
  - Tests
- Aktueller Stand

Agenda

- **Aufgaben des Projekts**
- Vorbereitung
  - Anforderungsanalyse / -definition
- Planung
  - Aufgabenteilung
  - Modularisierung
- Implementierung
  - Dokumentation
  - Tests
- Aktueller Stand

Aufgaben des Projekts

- Praktische Anwendung der Softwaretechnik-Theorie
- Kennenlernen der Entwicklungsphasen eines Softwareprojekts
- Konkretes Ziel: Übersetzer-Frontend
- Notwendige Zwischenschritte:
  - Formelle Definition der Quellsprache
  - Definition der Zielarchitektur
  - Formelle Definition der Zielsprache

Agenda

- Aufgaben des Projekts
- **Vorbereitung**
  - **Anforderungsanalyse / -definition**
- Planung
  - Modularisierung
  - Aufgabenteilung
- Implementierung
  - Dokumentation
  - Tests
- Aktueller Stand

Anforderungsbeschreibung

- Übersetzer-Frontend
  - Source-to-source Übersetzer
  - XML-basierte Quellsprache
  - Java Quellcode als Zielsprache

- Quellsprache:
  - Wohlgeformtes und valides XML
  - Korrektheitsprüfung mit XML-Schema und Validator
  - Sprachstrukturen repräsentieren eine einfache objektorientierte imperative Programmiersprache

- Zielsprache:
  - Java Sourcecode
  - Korrektheitsprüfung mit dem Java Compiler

- Formelle Grammatik
  - Wird noch ausformuliert

- Beispiel
  - Operatoren
    - Inner_Op ::= Inner_Op [+|-] Inner_Op
      | Inner_Op [*|/|%] Inner_Op
      | Inner_Op

Agenda

- Aufgaben des Projekts
- Vorbereitung
  - Anforderungsanalyse / -definition
- **Planung**
  - **Aufgabenteilung**
  - **Modularisierung**
- Implementierung
  - Dokumentation
  - Tests
- Aktueller Stand

Projektplanung

- ca. 2 Wochen
  - Vorbereitung, formale Anforderungen
  - Rollenverteilung, Modularisierung
- ca. 3 Wochen
  - Implementierung des Grundsystems
  - Modultests
- ca. 3 Wochen
  - Implementierung der optionalen Features
  - Optimierung
  - Integrationstests
- ca. 2 Wochen
  - Vervollständigung der Dokumentation
  - Abschlusspräsentation

Modularisierung

- Aufgrund der Besonderheiten der Quellsprache (XML) boten sich 4 Teilprojekte an:
  - Lexer → (René)
  - Document Object Model (DOM) → (Stefan)
  - Syntaxbaum → (Alex, Markus)
  - Codegenerierung → (alle gemeinsam)

- Die Schnittstellen sollen zwischen den beteiligten Projekten kollaborativ festgelegt werden.

- Implementierung soll parallel durchgeführt werden, über den Fortschritt soll wöchentlich berichtet werden.

Agenda

- Aufgaben des Projekts
- Vorbereitung
  - Anforderungsanalyse / -definition
- Planung
  - Aufgabenteilung
  - Modularisierung
- **Implementierung**
  - **Dokumentation**
  - **Tests**
- Aktueller Stand

Implementierung

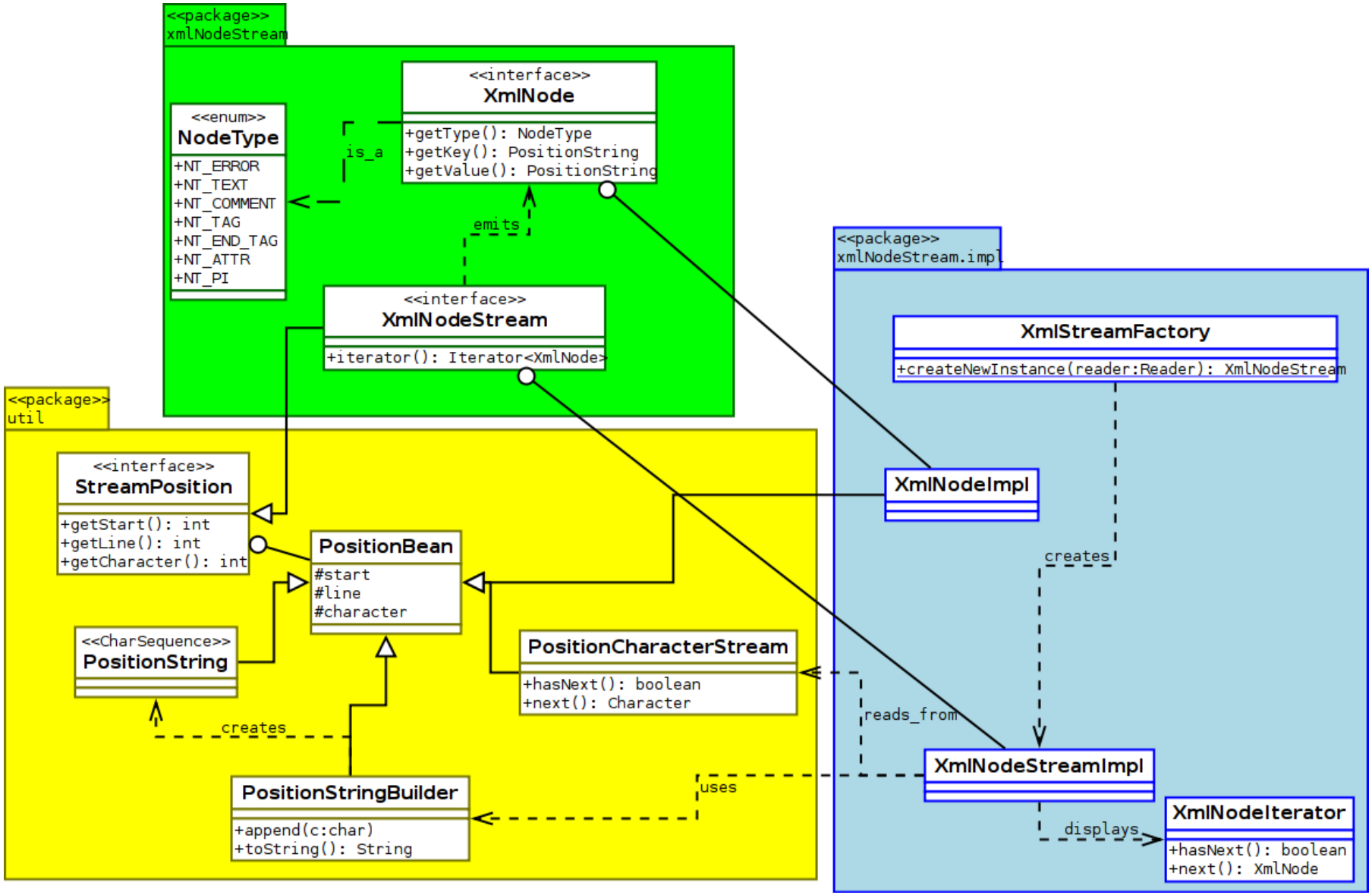
- Teilaufgabe 1: Lexer
  - Aufgabenstellung / Ziel: XML-Source in Token zerlegen
  - Eingabe: Quellcode als XML-Datei
  - Ausgabe: Token-Strom
  - Klassen:
    - xmlNodeStream – Tokenizer für XML
    - statementLexer – Tokenizer für die inneren Ausdrücke
    - Gemeinsame Komponenten für die Weitergabe der Position in der Eingabe zwecks Fehlerbehandlung

Implementierung

- Lexer, Schnittstellendefinition:
  - xmlNodeStream
    - Wird instanziiert über die Factory: XmlNodeStreamFactory.createNewInstance(Reader)
    - Implementiert das Interface Iterator<XmlNode>
    - Gibt zurück:
  - statementLexer
    - Wird instanziiert über die Factory-Methode: StatementLexer.tokenize(PositionCharacterStream)
    - Gibt zurück: Iterator<StatementNode>.
  - PositionBean - Kapselt Position in der Eingabe
  - PositionCharacterStream - Ein Zeichen-Iterator mit Pushback und Stream-Position
  - PositionString - Ein String, der seine Position in der Eingabe „kennt“
  - PositionStringBuilder - Ein StringBuilder für PositionString

Implementierung

- Lexer, UML-Diagramm:



Implementierung

- Teilaufgabe 2: DOM
  - Aufgabenstellung / Ziele:
    Aus XML-Token eine DOM-Baumstruktur generieren
    Baumknoten attributieren
    XML-Code validieren
  - Eingabe:
    Token-Strom
  - Ausgabe:
    DOM-Baum
  - Klassen:
    - DomNode – repräsentiert den Knoten des DOM-Baums
    - DomAttribute – Attribut eines Knotens
    - DomCreator – statische Klasse (Singleton), die den Baum generiert und zurückgibt

Implementierung

- DOM, Schnittstellendefinition:
  - DomNode:
    - getter und setter, um die benachbarten Elemente abzurufen
    - getAttributes – liefert die Attributliste des Knotens
  - DomAttribute
    - getName – liefert den Namen des Attributs
    - getValue – liefert den Wert des Attributs
  - DomCreator
    - Instanziierung über: DomCreator.init(Reader file)
    - Gibt zurück: DOM (mit der Methode createDOM())

Implementierung

- DOM, UML-Diagramm:

Implementierung

- Teilaufgabe 3: Parser / Syntaxbaum
  - Aufgabenstellung / Ziel:
    Aus DOM-Baum einen Syntaxbaum erzeugen
  - Eingabe:
    DOM-Baum
  - Ausgabe:
    Syntaxbaum
  - Klassen:
    - AbstractSyntaxTree
    - Klassen, die einzelne Sprachelemente repräsentieren

Implementierung

- Parser, Schnittstellendefinition:
  - AbstractSyntaxTree:
    - Instanziierung über den Konstruktor mit dem DOM-Baum als einzigen Parameter:

      public AbstractSyntaxTree(DomNode node)
    - Gibt zurück: Syntaxbaum über die getRoot()-Methode
  - Klassen der Sprachelemente:
    - Instanziierung über den Konstruktor mit dem jeweiligen Knoten als Parameter (z.B. public Class(DomNode node))
    - Geben zurück: Komponenten des Sprachelements über die passenden getter-Methoden (z.B. getDeclarations())

Implementierung

- Parser, UML-Diagramm:

Implementierung

- Teilaufgabe 4: Codegenerierung
  - Aufgabenstellung / Ziel:
    Aus dem abstrakten Syntaxbaum die Zielsprache erzeugen
  - Eingabe:
    Syntaxbaum
  - Ausgabe:
    Java-Quellcode
  - Klassen:
    - JavaBuilder – erstellt die Java Klassen
    - Director – generiert aus den Java Klassen die Quelldateien
    - Weitere Klassen noch in Entwicklung

Implementierung

- Codegenerierung, Schnittstellendefinition:
  - JavaBuilder
    - Wird instanziiert über den Konstruktor mit dem abstrakten Syntaxbaum und dem Pfad zu den erstellten Klassen
    - Gibt zurück: eine Liste mit Klassen über getClasses()
  - Director
    - Wird nicht instanziiert (statisch)
    - Hat eine einzige Methode

      build(Builder b, DirectoryWriter w)
    - Ausgabe wird in Form von .java-Dateien direkt auf den gewünschten Ausgabekanal (Bildschirm, Festplatte, …) geschrieben

Implementierung

- Codegenerierung, UML-Diagramm:

Agenda

- Aufgaben des Projekts
- Vorbereitung
  - Anforderungsanalyse / -definition
- Planung
  - Aufgabenteilung
  - Modularisierung
- Implementierung
  - Dokumentation
  - Tests
- **Aktueller Stand**

Aktueller Stand

- Lexer (Rene)
  - xmlNodeStream:
    - Zu 90 % abgearbeitet
    - Es fehlt noch die korrekte Umsetzung von XML-Entities (&, <, u.ä)
  - statementLexer:
    - Zu 75 % abgearbeitet
    - Es fehlen noch Gleitkommazahlen und Strings
  - Gemeinsame Komponenten:
    - Fehlt: Methode, die Identifier auf ihre Validität überprüft

Aktueller Stand

- Lexer (René), Entstandene Probleme:
  - xmlNodeStream:
    - Es werden teilweise nicht valide XML-Ausdrücke akzeptiert
      - Sind eher selten
      - Vorgehensweise wird noch geklärt
  - statementLexer:
    - Kommunikationsprobleme mit Kollegen wegen Interface
      - Inzwischen weitestgehend ausgeräumt
    - Zusätzliche Änderungen an dem Interface für eine bessere Fehlerbehandlung
      - Klasse PositionString soll Abhilfe schaffen (implementiert).

Aktueller Stand

- DOM (Stefan)
  - Implementierung ist zu 100% funktionsfähig
- TODO:
  - Doku
  - mehr/bessere Tests
- Probleme:
  - Kleinere Implementierungsdetails
    - Werden in Absprache gelöst

Aktueller Stand

- Syntaxbaum (Alex)
  - AbstractSyntaxTree:
    - Implementierung fertig
- TODO:
  - Builder vervollständigen
  - Fehlerbehandlung
  - Testen
- Probleme:
  - Sprachdefinition unpräzise
    - In Absprache präzisiert (z.B. return)
  - Ungünstige Designentscheidungen
    - In Absprache korrigiert (z.B. Static-Statement)

Aktueller Stand

- Expressions / Statementparser (Markus)
  - Expression
    - Implementierung fertig
  - Statementparser
    - Implementierung fertig
- TODO:
  - Identifier für Symboltabelle
  - Dokumentation
- Probleme:
  - Kleiner Designentscheidungen
    - In Absprache präzisiert

Aktueller Stand

- Codegenerierung:
  - Implementierung zu etwa 30% fertig
  - Wird aktuell daran gearbeitet

Ausblick

- Kommende Woche:
  - Codegeneration abschliessen
  - Dokumentation
  - Komponententests
- Darauffolgende Woche(n):
  - Optionale Features
  - Integrationstests
- Abschlusspräsentation

Vielen Dank!