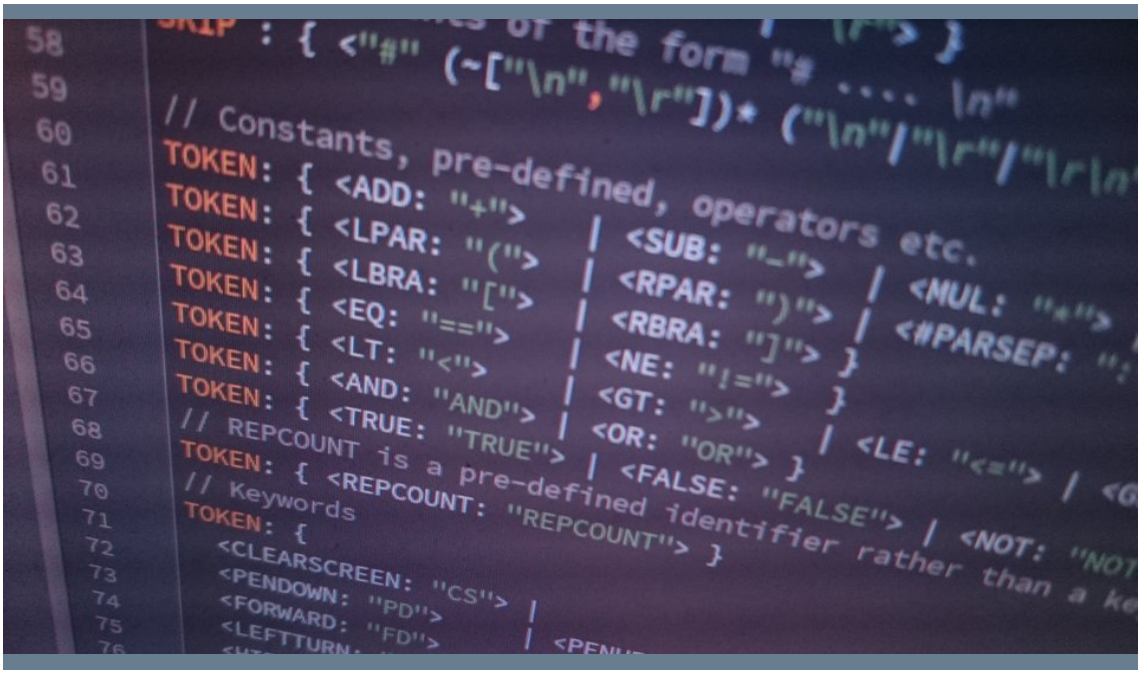

Presentator3000

Report

This document describes the process of what came to be this online presentation tool and what we could have done better in hindsight.

Degree course:	Information technology
Lecture:	Project 2
Authors:	Michael Bolli (michael.bolli.1@students.bfh.ch),
	Robin Schmid (robinsamuel.schmid@students.bfh.ch)
Principal:	Marcel Pfahrer
Date:	2016-10-11

Version history

Version	Date	Status	Misc.
0.1	2016-10-11	Draft	Initial version

1 Introduction

(Intro Text) In order to confirm that the structure of a text or a program is correct, it is necessary to make sure this program is valid in respect to the grammar. This is called parsing. First the lexical analysis groups characters into tokens which can stand for various identifiers, keywords and constants of a language. After that the parser has to ensure that the input text is well structured, and tries to generate a parse tree. Only if a parse tree can be built, is a text or program syntactically correct.

This project is a parser for LOGO in Java via JavaCC.

2 Grammar

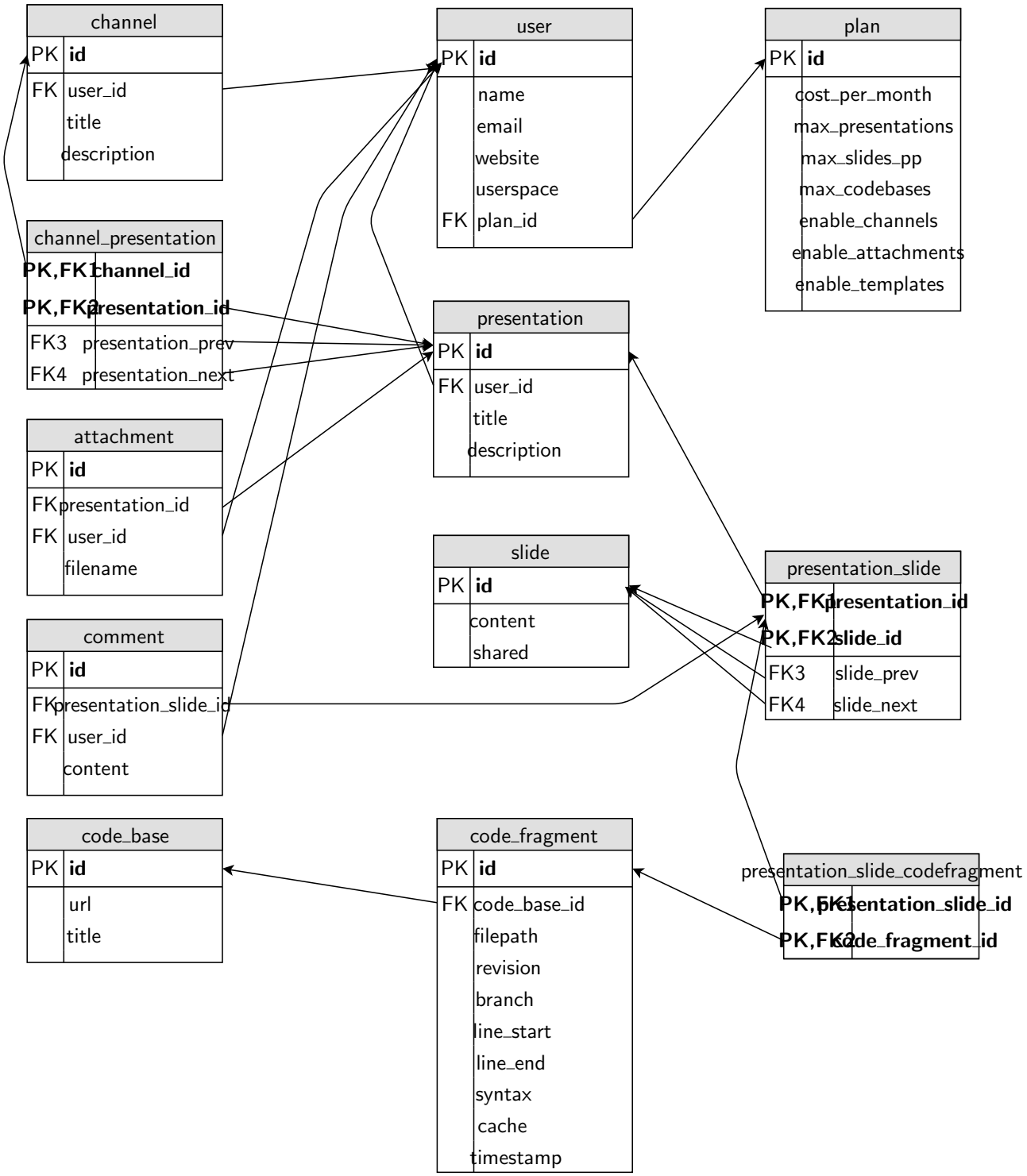

Figure 2.1: Domain Model

3 Use Cases

3.1 Use Case Diagramm

3.2 Use Case Beschreibungen

3.2.1 UC 001: Register

Use Case Name	Register
Primary Actor	
Stakeholders	
Preconditions	
Main Success Scenario	1. Der EnterpriseManager gibt eine valide Emailadresse und zweimal ein sicheres Passwort ein.
Extensions	

3.2.2 UC 002: Login

3.2.3 UC 003: Logout

3.2.4 UC 004: Manage Presentation

3.2.5 UC 005: Manage Slide

3.2.6 UC 006: Share Presentation

3.2.7 UC 007: View Presentation

3.2.8 UC 008: Print Presentation

3.2.9 UC 009: Manage Comment

3.2.10 UC 010: Manage Codebase

3.2.11 UC 011: Manage Code Fragment

3.2.12 UC 012: Update External Resources

3.2.13 UC 013: Manage Channel

3.2.14 UC 014: Subscribe Channel

3.2.15 UC 015: Bookmark Presentation

3.2.16 UC 016: Manage Profile

4 Solution

After some fiddling in the beginning, implementation was quite straightforward. The grammar had to be converted into a tree structure, each node representing methods in the Logo.jj file. Most of these methods are writing strings of Java code into the destination file via PrintWriter. It was of course necessary to pay attention to things like indentation and REPCOUNT in- and decreasing.

The REPCOUNT (or for-loop in Java) isn't directly supported by LOGO. We solved that with a similar approach to indentation: An integer keeps track of REPCOUNT, which is in- and decreased when used in tokens like REPEAT.

5 Test

Apart from the provided LOGO files in the logofiles Folder, we used our own test file test.logo for final testing.

6 Limitations & Considerations

This parser only verifies the correctness of the LOGO program on a syntactical level. The semantic verficiation will be performed during the Java compilation of what is produced by our parser.

We might have been faster to solve this project if our group had met in real life for a day, instead of communicating via e-Mail over weeks.