

**Decision Support System**

# Table of Contents

Table of Contents 2

Problem Statement 3

Contributions 4

Software / Platform 5

Appendix A 6

GRAMMAR 6

# Problem Statement

OpenMRS – Clinical Decision Support System

It is one thing to have information readily available, it is another to understand and make the best use of that information. The OpenMRS system provides a repository of data on patients but it is still up to the physician to decide the course of treatment, tests that need to be run, and medications to be administered. The Decision Support System (DSS) is designed to assist the physician by providing alerts based on correlation of data and programmatic rules. Because of the vast amounts of data on patients, the number of medications and tests available and the variability of patient behavior, the DSS is designed to provide rules to avoid mistakes, speed patient care and optimize resources.

DSS allows doctors to create rules to alert them if they prescribe a medication that may interact with other medications that the patient is taking or may be allergic to. Or, it can suggest running tests that may be due or alert to the fact that prior tests need to be redone. Decision support is about correlating the data is a manner useful to the physician.

To create a DSS, a method must exist to specify these rules, and should be of a nature to allow non-technical individuals to create them. Once created a system to store and interpret those rules needs to be devised and finally the results of the rules need to be displayed back to the physician in an intuitive and meaningful way.

The task here is given the presented language grammar (see appendix A), create the interpreter that can store and process rules and display the results on the patient summary and dashboards in OpenMRS.

# Contributions

Robert Bierman

Kay Choi

Steven Gimeno

Jason Lum

Ying Kit Ng

Bianca Uy

Victor Woeltjen

# Software / Platform

Most of the programming was done in Java.

client pages in Javascript JSP and HTML, and XML

# Appendix A

## GRAMMAR

PROGRAM -> ‘program’ D\* BLOCK ==> program

BLOCK -> ‘{‘ S\* ‘}’ ==> block

D -> 'function' NAME FUNHEAD BLOCK ==> functionDecl

FUNHEAD -> '(' (NAME list ',')? ')' ==> formals

S -> ‘if’ EE ‘then’ BLOCK ('else' BLOCK)? ==> if

-> ‘if’ EE ‘then’ BLOCK Elif ==> if

-> ‘while’ EE BLOCK ==> while

-> 'for' NAME in NLIST BLOCK ==> FOR

-> ‘return’ EE ==> return

-> BLOCK

-> IdMod’:=‘ EE ==> assign

-> NAME '(' (EE list ',')? ')' ==> call

Elif -> ‘elsif’ EE 'then' BLOCK Elif ==> elsif

-> ‘elsif’ EE 'then' BLOCK ('else' BLOCK)? ==> elsif

EE -> E

-> EE '||' E

E -> SE

-> SE ‘==‘ SE ==> =

-> SE ‘!=‘ SE ==> !=

-> SE ‘<‘ SE ==> <

-> SE ‘<=‘ SE ==> <=

SE -> T

-> SE ‘+’ T ==> +

-> SE ‘-’ T ==> -

-> SE ‘|’ T ==> or

T -> TT

-> T ‘\*‘ F ==> \*

-> T ‘/’ F ==> /

-> T ‘&’ F ==> and

TT -> F

-> TT \*\* F ==> \*\*

F -> ‘(‘ EE ‘)’

-> IdMod

-> <literal>

-> NAME '(' (EE list ',')? ')' ==> call

-> Object '(' (NAME list ',')? ')' ==> ObjectDecl

-> new NAME ==> Object

-> LIST

IdMod -> NAME

-> NAME '.' NAME ==> fieldRef

NLIST -> NAME

-> LIST

LIST -> '{' (E list ',')? '}' ==> list

NAME -> <id>