Gerd Doeben-Henisch, member PES

The Planet Earth Simulator Project – Description of Formal Systems Language (FSL) Vers.1.0

Abstract—**This is the general description of the FORMAL SYSTEMS LANGUAGE (FSL) of the PLANET EARTH SIMULATOR (PES) Project. A more detailed document is the FSL-grammar itself.**

Index Terms—**Computational Science, Computational Semiotics, Syntax, FSL-Grammar**

INTRODUCTION

The PLANET EARTH SIMULATOR (PES)-Project [1] is an open source project which has been started January 2003 by Gerd Doeben-Henisch and Jens Heise at the Institute for New Media [2] in collaboration with the University of Applied Sciences [3] in Frankfurt am Main (Germany). The intention of this project is to offer everybody who has a webbrowser at hisdisposal to generate knowledge or to use knowledge which others have already delivered to the system.

To get a rough idea of the project we will have a short look to it from the point of usage.

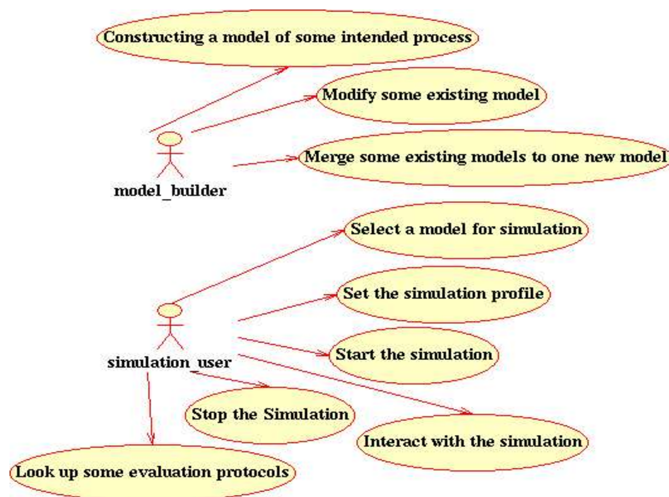

Fig. 1: The Use Case of the Planet Earth Simulator. The user can act either as a model builder or as a simulation user.

From the point of USAGE (cf. Fig. 1) you can construct arbitrary intended models of parts of the world. For this you have a VISUAL LANGUAGE *VisualFSL* at your disposal to built VISUAL ARTIFACTS --diagrams, graphs-- as representations of your intended parts of the world. A complete description is called an LMODEL as short for *Logical* Model. And, as far as other pre-built LModels are already available, you can SELECT LMODELS for simulation and then you can RUN THE LMODELS for to simulate some intended processes. During such a simulation you can also INTERACT with the simulated processes, individually or in groups. SYNCHRONOUSLY or ASYNCHRONOUSLY you can also activate some EVALUATION PROTOCOL for to collect and to show selected data of the process activities.

The PES-Project is supported by the Institute for New Media e.V. in Frankfurt am Main (Germany)

THE FORMAL SYSTEMS LANGUAGE FSL OF THE PES-PROJECT

In the PES-Project we have until now decided to take the well established terminology of systems and systems based programming languages as our computer readable language. We call this language FORMAL SYSTEMS LANGUAGE (FSL).

As direct interface to a 'normal' user as model builder we will provide a graphical user interface with a VISUAL LANGUAGE *VisualFSL* based on SYSTEMS as smallest possible units (cf. Fig.2).

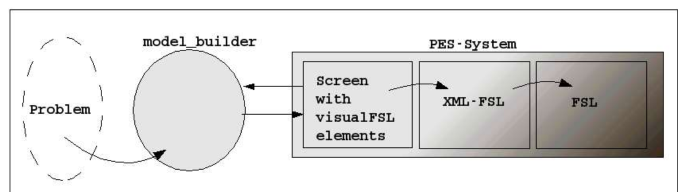

Fig.2: A user as model builder tries to translate a real world problem into a visual representation with the visual Formal Systems Language FSL. This will in turn be converted into FSL

The representation in *visualFSL* will automatically be converted into an internal XML-Format of FSL. This format then can be further converted into a 'pure' FSL-text.

GENERAL DESCRIPTION OF FSL

We will give now a general overview of the Formal

Systems Language V1.0 (cf. Fig.3).

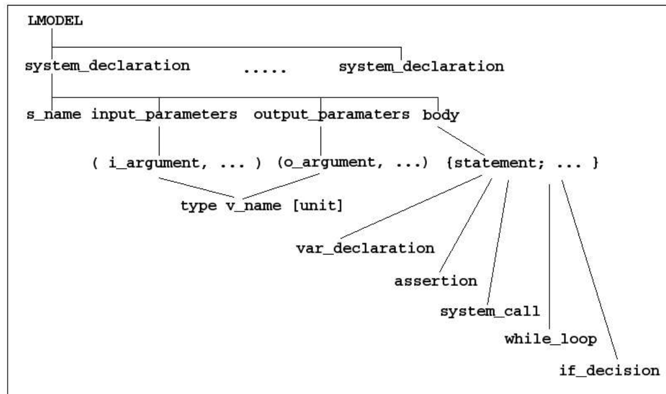

Fig.3: General overview of the FSL version 1.0

The main units described by a wellformed FSL-document is an LMODEL (short for LogicalModel).

An LMODEL consists of at least one *system_declaration*. If a system_declaration contains a *system_call* of a function which is *not built-in*, then there must exist another system_declaration which describes the system of the system_call.

A *system_declaration* consists of a system_name called *s_name,* some input parameters (which can be empty) called *input_parameters*, some output parameters (which can be empty) called *output_parameters*, and a *body* which contains the description of that what the system 'will do'.

The *s_name* is an identifier which is used to identify the system. But there can be the same identifier as long as the input and output parameters differ by type and number.

The *input_parameters* as well as the *output_parameters* consist of arbitrarily many arguments, each of them has the structure *type v_name [unit]*. This means every argument has the name of a variable introduced by a type and –optionally-- a unit of measurement.

The *body* consists of statements embedded in curly braces {}. The *statements* are sequences of a single statement each followed by a semicolon ' ;' .

There are several types of statements:

- A declaration of a variable *var_declaration* introduces a local variable with type and name.
- An *assertion* combines a variable with some value. The value can be either another variable or a constant of type num or bool or a string or a system call. However, to use a system call in an assertion is somehow redundant because the output variable of the system call can be the same variable as the variable for which the values is asserted. Thus it is more economic to use systems calls as 'stand alone assertions' .
- In a *system_call* one uses the name of a system to call this system to work; additionally one can give the calling system some arguments as input and variables for the storage of output values. A system_call can be recursiv because the value of an input argument can again be a system_call if the output value of the system call is in agreement with the input argument.
- A *while_loop* can be used to repeat some statements depending on some condition.
- With an *if_decision* one can introduce decisions between alternative statements.

BUILT IN FUNCTIONS

Here is the actual list of built in functions taken from the bison grammar file "fsl-1.0.y" (for an explanation of (f)lex and bison see [5]):

/*-----------ATTACHEMENT: PRESUPPOSED BUILT-IN SYSTEMS/FUNCTIONS for FSL-1.0---------------*/

/* num := int | float */
add(num X, num Y)[num Z] Addition, Z = X + Y
sub(num X, num Y)[num Z] Subtraction, Z = X - Y
mul(num X, num Y)[num Z] Multiplication, Z = X Y
div(num X, num Y)[num Z] Divisionion, Z = X / Y
pow(num X, num Y)[num Z] power, Z = X^Y
sqr(num X, num Y)[num Z] square root, Z = Y-the root of X

----

less(num X,num Y)[bool Z] Z = X < Y
equal(num X,num Y)[bool Z] Z = X == Y
lequal(num X,num Y)[bool Z] Z = X <= Y
greater(num X,num Y)[bool Z] Z = X > Y
gequal(num X,num Y)[bool Z] Z = X >= Y

In every String-Operation has the system to guarantee the correct operation in memory

strcpy(str X)[str Y] Y = X
strcat(str X,str Y)[str Z] Z = x + Y
fsubstr(str X,str S)[int P] P is the position of S in X; if no occurence, then P isnegativ
rsubstr(str X,str S,int P)[bool Z] Replace S in X at P; if no occurence, then Z is 0

----

show(str A)[int SCREEN] prints content of var A on the screen at actual position
show(str A,int ROW,int COL)[int SCREEN] prints content of var A on the screen beginning at position (ROW,COL)
get(int SCREEN)[str X] Reads Keyboard and stores content in X (str)

aton(str X)[int Y] converts string X to number Y
aton(str X)[float Y] converts string X to number Y
ntoa(int X)[str Y] converts number X to string Y
ntoa(float X)[str Y] converts number X to string Y
open(str FILENAME,str MODE)[bool X] opens a file FILENAME in MODE ("r","a","w") gets a result X
write(str/int/float/bool X,str FILENAME)[bool Y] write content of X into file FILENAME, gets result Y
read(str FILENAME)[int/float/str/bool X,bool Y] read content of file FILENAME into X, gets result Y
close(str FILENAME)[bool X] closes a file FILENAME with result X
exit(str MESSAGE)[bool FBCK] Exits an lmodel with message and Feedback
----

FSL-1.0 GRAMMAR ACCORDING TO THE BISON-GRAMMAR FILE FSL-1.0.Y

/*---------BISON DECLARATIONS ----------------------*/

%union{ /*--- Defining data types ---------*/

char string[255]; /* Achtung: Textinput bzgl. Laenge kontrollieren ! */
}

/*--------- BISON TOKENS ------------------------------
*
* Every token kann have a special data type <...> if diferent from int
*
---/

%token <string>FNAME LB RB <string>TYPE <string>VAR <string>UNIT SEP LSB RSB LCB RCB EQU <string>NUM <string>STRING COLON <string>BOOL CMPOP LOP1 LOP2 WHILE IF ELSE

%% /*------- Grammar rules and actions ------------------ */

/* An lmodel consists of a heading system declaration followed by
* as many other system declarations as there are system calls which
* need further exlications by a sytem declaration
--/

lmodel: /* empty */
| lmodel system_declaration
;

/*---
* A system declaration consist of input, output and a body
---/

system_declaration: FNAME input output body
;

/*---
* input and output contain a variable list of arguments
* The body contains a variable list of statements
---/

input: LB iarguments RB
;
output: LSB oarguments RSB
;

body: LCB statements RCB
;

/*---
* input arguments are either empty or
* consist of 2 default elements and 1 optional element:
* default is the type of the argument and the name of the variable
* optional is a unit of measurement
---/

iarguments: /* empty */
| itype ivar
| itype ivar iunit
| iarguments SEP itype ivar
| iarguments SEP itype ivar iunit
;

itype: TYPE
;

ivar: VAR
;

iunit: UNIT
;
/*---
* output arguments are either empty or
* consist of 2 default elements and 1 optional element:
* default is the type of the argument and the name of
* the variable
* optional is a unit of measurement
---/

oarguments: /* empty */
| otype ovar
| otype ovar ounit
| oarguments SEP otype ovar
| oarguments SEP otype ovar ounit
;

otype: TYPE
;

```
ovar:  VAR
;

ounit: UNIT
;

/*-------------------------------------------------------
 * statemets are sequences of single statements
 * There are several kinds of statements
 * We will start with the following  types:
 * definitions, assertions, system calls, if calls,  and
* while calls
 *-----------------------------------------------------------------*/

statements: /*empty*/
        | definition COLON
        | assertion COLON
        | system_call COLON
        | while_call
        | if_call
        | statements definition COLON
        | statements assertion COLON
        | statements system_call COLON
        | statements while_call
        | statements if_call
  ;

/*------------------------------------------------------------------
 * In a definition one introduces a new variable
 *-------------------------------------------------------------------*/

definition: TYPE VAR
;

/*------------------------------------------------------------------
 * In an assertion a certain value will be assigned
* to a variable
 * The Compiler/ Interpreter has to control that the type
* and the kind
 * of asserted value are in agreement!
 *------------------------------------------------------------------*/

assertion: VAR EQU term
;

term : VAR
    | const
;

const: NUM
    | BOOL
    | STRING
    | system_call
;

/*------------------------------------------------------------------
 * In a system call one calls a system only by
 * giving its name, its input arguments, and its
 * output arguments
 *
 * System calls presuppose either built-in systems
 * or one needs an additional system declaration with
 * the same name an the same type of arguments
 *-----------------------------------------------------------------*/

system_call: FNAME cinput coutput
;

cinput: LB ciarguments RB
;

ciarguments: ciargument
        | ciarguments SEP ciargument
;

ciargument: term
;

coutput: LSB coarguments RSB
;

coarguments: coargument
        | coarguments SEP coargument
;

coargument: VAR
;

/*-------------------------------------------------------
 *
 * WHILE CALL
 *
 * As long as a condition is true the body will be executed
 *
 * There is a difference betwen comparing
* operations CMPOP
 * and logical operations LOGOP
 * While CMPOP can also compare NUMs and
* STRINGs can
 * LOGOP only oerate on BOOLean values
 *---------------------------------------------------------------*/

while_call: WHILE condition body
;

condition: LB cond RB
;

cond: VAR
    | BOOL
   | CMPOP LB term2 SEP term2 RB LSB VAR RSB
   | LOP1 LB bool RB LSB VAR RSB
   | LOP2 LB bool SEP bool RB LSB VAR RSB
   | system_call
```

```
;

term2: NUM
    | STRING
    | bool
;

bool: VAR
   | BOOL
   | system_call
   | CMPOP LB term2 SEP term2 RB LSB VAR RSB
   | LOP1 LB bool RB LSB VAR RSB
   | LOP2 LB bool SEP bool RB LSB VAR RSB
;

/*------------------------------------------------------------
 *
 * IF CALL
 *
 * As long as a condition is true the body will be executed
 *
 *----------------------------------------------------------------*/

if_call: IF condition body
      |  IF condition body ELSE body
;



%%
```

THE FSL-1.0 SCANNER ACCORDING THE FLEX-RULE FILE FSL-1.0.L

```
/* Some flex-Makros */

FNAME      [a-z]+
ALPH       [a-z A-Z.,;!?\-+0123456789]+
ZIFF      [0-9]+

%%  /* flex rules   */

[ \t]       ;

"("         { printf( "LB: %s\n", yytext ); return(LB); }
")"         { printf( "RB: %s\n", yytext ); return(RB); }
"["         { printf( "LSB: %s\n", yytext ); return(LSB); }
"]"         { printf( "RSB: %s\n", yytext ); return(RSB); }
"{"         { printf( "LCB: %s\n", yytext ); return(LCB); }
"}"         { printf( "RCB: %s\n", yytext ); return(RCB); }
","         { printf( "SEP: %s\n", yytext ); return(SEP); }
";"    { printf( "COLON: %s\n", yytext ); return(COLON); }
"="         { printf( "EQU: %s\n", yytext ); return(EQU); }

int             {printf(    "TYPE:    %s    \n",    yytext);    strcpy
(yylval.string,yytext); return(TYPE); }
   float            {printf(   "TYPE:   %s   \n",   yytext);   strcpy
(yylval.string,yytext); return(TYPE); }
   str             {printf(   "TYPE:   %s   \n",   yytext);   strcpy
(yylval.string,yytext); return(TYPE); }
   bool            {printf(   "TYPE:   %s   \n",   yytext);   strcpy
(yylval.string,yytext); return(TYPE); }


   year            {printf(   "UNIT:   %s   \n",   yytext);   strcpy
(yylval.string,yytext);  return(UNIT); }
   month            {printf(   "UNIT:   %s   \n",   yytext);   strcpy
(yylval.string,yytext);  return(UNIT); }
   week            {printf(   "UNIT:   %s   \n",   yytext);   strcpy
(yylval.string,yytext);  return(UNIT); }
   day            {printf(   "UNIT:   %s   \n",   yytext);   strcpy
(yylval.string,yytext);  return(UNIT); }
   hour            {printf(   "UNIT:   %s   \n",   yytext);   strcpy
(yylval.string,yytext);  return(UNIT); }
   min            {printf(   "UNIT:   %s   \n",   yytext);   strcpy
(yylval.string,yytext);  return(UNIT); }
   sec            {printf(   "UNIT:   %s   \n",   yytext);   strcpy
(yylval.string,yytext);  return(UNIT); }

   true            {printf(   "BOOL:   %s   \n",   yytext);   strcpy
(yylval.string,yytext);  return(BOOL); }
   false            {printf(   "BOOL:   %s   \n",   yytext);   strcpy
(yylval.string,yytext);  return(BOOL); }


   while            {printf(   "WHILE:  %s  \n",   yytext);   strcpy
(yylval.string,yytext);  return(WHILE); }
   if            {printf(   "IF:   %s   \n",   yytext);   strcpy
(yylval.string,yytext);  return(IF); }
   else            {printf(   "IF:   %s   \n",   yytext);   strcpy
(yylval.string,yytext);  return(ELSE); }

   less            {printf(   "LESS:   %s   \n",   yytext);   strcpy
(yylval.string,yytext);  return(CMPOP); }
   greater        {printf(  "GREATER:  %s  \n",  yytext);  strcpy
(yylval.string,yytext);  return(CMPOP); }
   equal            {printf(   "EQUAL:   %s   \n",   yytext);   strcpy
(yylval.string,yytext);  return(CMPOP); }

   not            {printf(   "NOT:   %s   \n",   yytext);   strcpy
(yylval.string,yytext);  return(LOP1); }
   and            {printf(   "AND:   %s   \n",   yytext);   strcpy
(yylval.string,yytext);  return(LOP2); }
   or            {printf(   "OR:   %s   \n",   yytext);   strcpy
(yylval.string,yytext);  return(LOP2); }

   [\-]*{ZIFF}        {printf(  "VAL:  %s  \n",  yytext);   strcpy
(yylval.string,yytext); return(NUM); }
   [\-]*{ZIFF}[,.]{ZIFF}        {printf(  "VAL:  %s  \n",  yytext);
strcpy(yylval.string,yytext); return(NUM); }

   [V][V][a-z:A-Z 0-9\(\)\[\]\{\}\"\+\-\*\^V,.=]*[V][V]     {printf(
"COMMENT: %s \n", yytext);   }
```

```
["]{ALPH}["]    {printf( "STRING: %s \n", yytext);  strcpy
(yylval.string,yytext); return(STRING); }

([A-Z]+{ZIFF}*)([_][A-Z]+{ZIFF}*)*    {printf( "VAR: %
s \n", yytext);  strcpy(yylval.string,yytext); return(VAR); }

{FNAME}{ZIFF}*      {printf( "FNAME: %s \n", yytext);
strcpy(yylval.string,yytext); return(FNAME); }

.          {return(yytext[0]); }

%%
```

SOME EXAMPLE LMODELs

```
//----------------------------------------//
// test8.fsl-1.0                  //
//   //
//-----------------------------------------//

pop1(int DURATION)[float BIRTHS year]{

float POPULATION;
float MIO;
float BIRTHRATE;
float BIRTHS;
int COUNT;
bool B;

COUNT = 1;

if(less(DURATION,1)[B]){
// Checking the test condition //
exit("null")[B]; }

else {

mul(5.7,pow(10,6)[MIO])[POPULATION];
BIRTHRATE = 0.025;

while(less(COUNT,DURATION)[B]){

mul(POPULATION, BIRTH_RATE)[BIRTHS];

show(BIRTHS)[SCREEN];

add(COUNT,1)[COUNT];
}
}//End of else //
}
```

```
//----------------------------------------//
// test9.fsl-1.0                  //
//   //
//-----------------------------------------//
//  //
// The values will interactively asked from the user //
// The intermediate results will be shown on screen //
// and will be stored in a file             //
//  //
//------------------------------------------//

pop1(int DURATION)[int VOID]{

float POPULATION;
float MIO;
float BIRTHRATE;
float BIRTHS;
int COUNT;
bool B;
bool FBCK;

COUNT = 1;

show("Please enter number of population at starttime!")
[SCREEN];
get(SCREEN)[POPULATION1];

atof(POPULATION1)[ POPULATION];

show("Please enter number of birthrate at starttime!")
[SCREEN];
get(SCREEN)[BIRTHRATE1];

atof(BIRTHRATE1)[ BIRTHRATE];

if(less(DURATION,1)[B]){
// Checking the test condition //
exit("null")[B]; }

else {

open("results","a")[FBCK];

while(less(COUNT,DURATION)[B]){

mul(POPULATION, BIRTH_RATE)[BIRTHS];

show(BIRTHS)[SCREEN];

write(BIRTHS,"results")[FBCK];

add(COUNT,1)[COUNT];

}//End of while //

close("results")[FBCK];
```

```
}//End of else //

}
```

References

[1] Planet Earth Simulator (PES) Project: http://www.planetearthsimulator.org (This is the project website)
[2] Institute for New Media e.V.: http://www.inm.de (This is the web site of the sponsoring institute)
[3] University of Applied Sciences - Department of Computer Science and Engineering, http://www.fh-frankfurt.de/2_studium/introseiten/index_2fb2.html (The main site of the faculty)
[4 George J.Klir , *Facets of Systems Science*, New York - London: Plenum Press, 1991
[5] Helmut HEROLD, "lex und yacc. Lexikalische und syntaktische Analyse", Addison-Wesley Publishing Company, Bonn - München - Paris et al , 1992