Bootstrapping CoCoViLa

Enn Tõugu & Pavel Grigorenko
Institute of Cybernetics of TUT

Lohusalu, October 2013

Background

- It is natural to use MBSD in development and application of domain-specific languages, and there are numerous tools, mainly simulation tools.
- Our goal is to expand the application domain of CoCoViLa for general-purpose software development.
- This requires a different software technology. In particular, no restrictions can be accepted on using a programming language in implementation of components.
- When software components are developed for one project, one cannot expect that these components will be reused many times, hence the effort for developing components should be minimal.
- As a test case for the technology we have chosen the development of a successor of CoCoViLa that we call CCVL here.

About the experiment

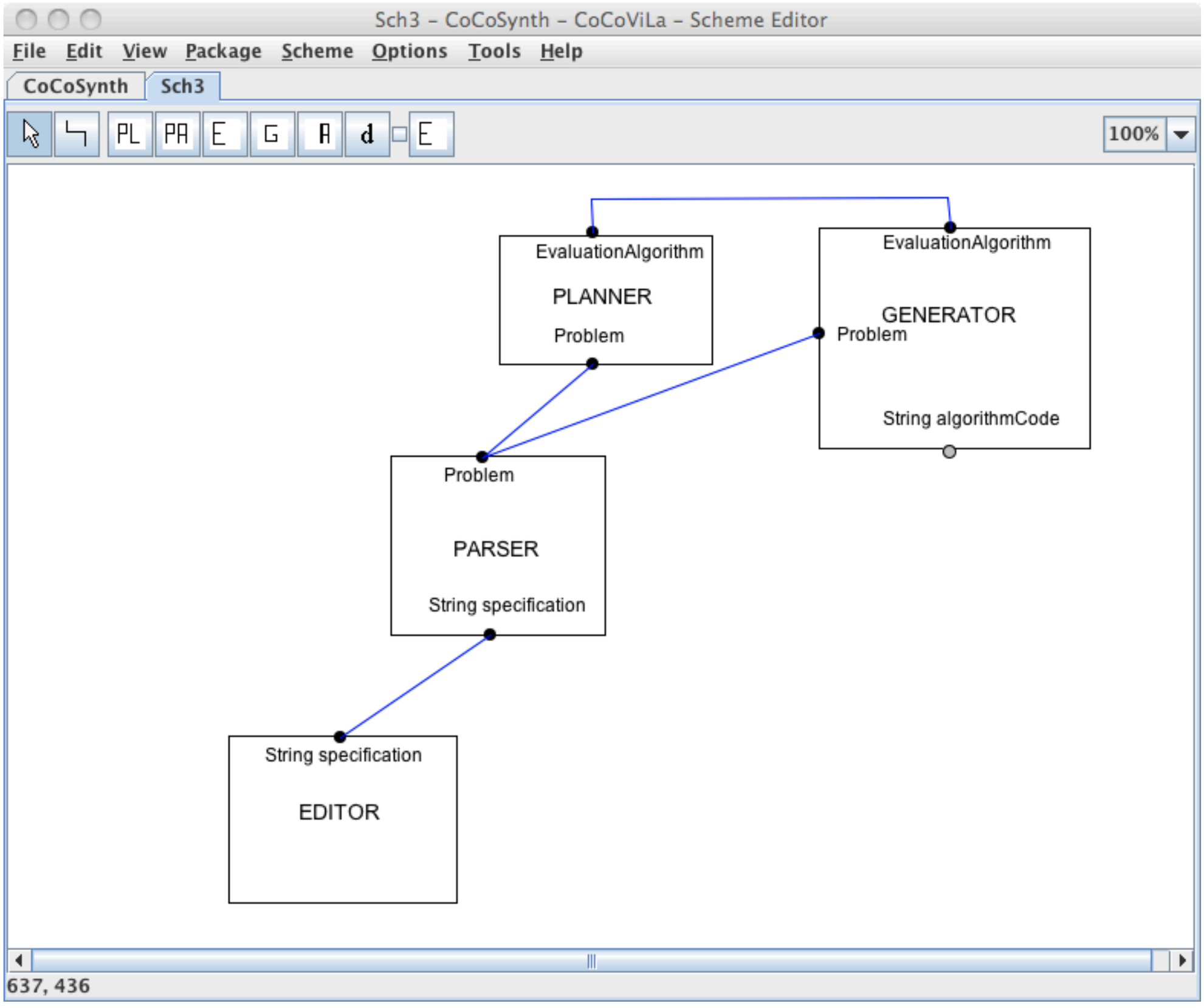
- When beginning this project, we had already source code of CoCoViLa, that consisted of 240 Java classes, totally 30K lines of code. These classes had been developed without any restrictions on programming in Java.

- Our intention is to use these classes as much as possible for CCVL, and to modify thereafter the software in continuous integration mode on Eclipse platform.

- At present, we have no intention of changing the functionality of the system. So we have an essential part of the requirements specification in the form of documentation of the existing software -- *http://www.cs.ioc.ee/cocovila/docs.php*. The first design step of the project is developing a model of the system.

Questions to be answered

- Can one use Java software developed without restrictions in this technology?
- How much the existing legacy classes have to be changed, and how?
- Can one use a metaclass for implementing GUI, and how?
- Can one develop software gradually, in a continuous integration mode?

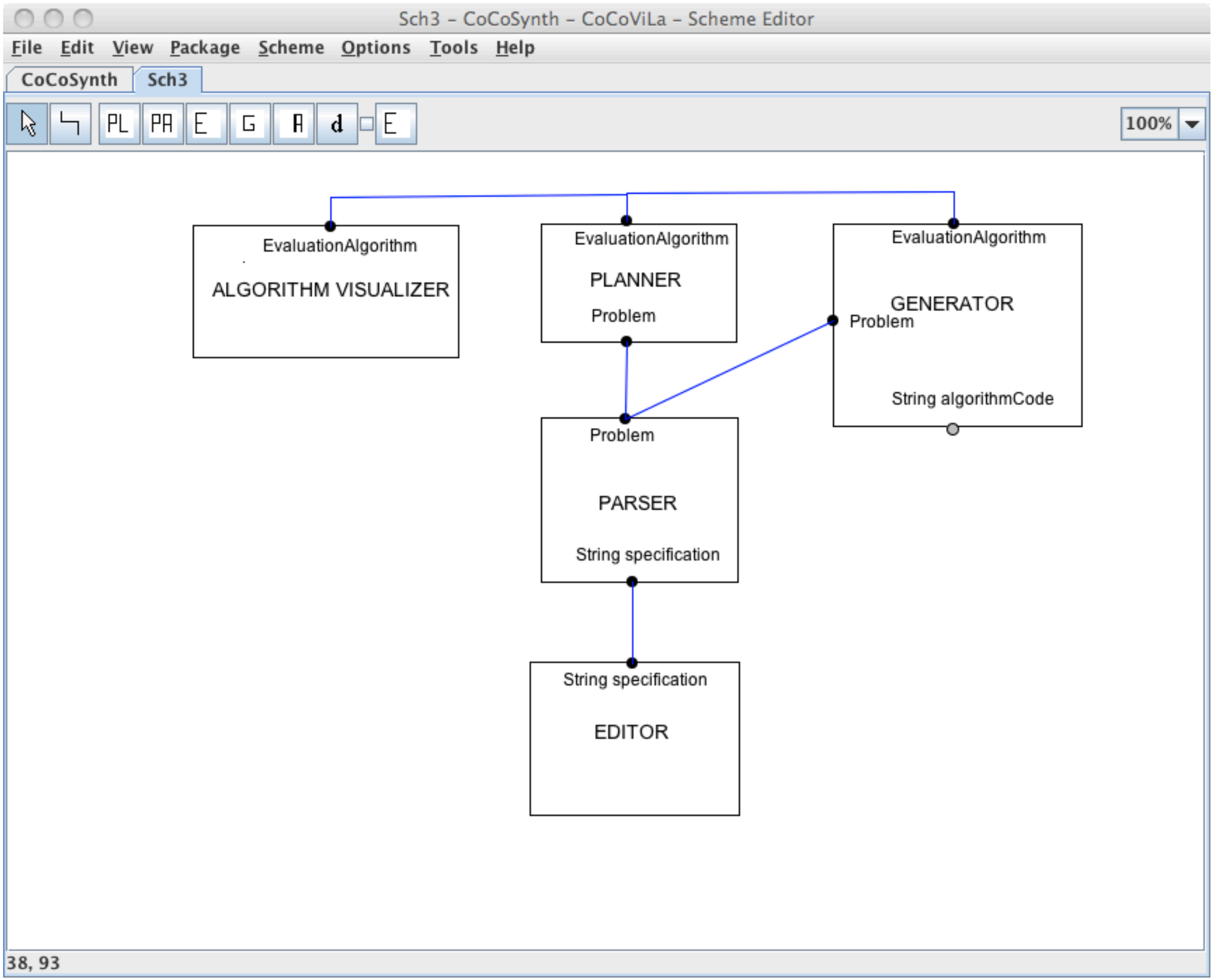

Sch3 – CoCoSynth – CoCoViLa – Scheme Editor
File Edit View Package Scheme Options Tools Help
CoCoSynth
Sch3
100%
EvaluationAlgorithm
PLANNER
Problem
EvaluationAlgorithm
GENERATOR
Problem
String algorithmCode
Problem
PARSER
String specification
String specification
EDITOR
637, 436

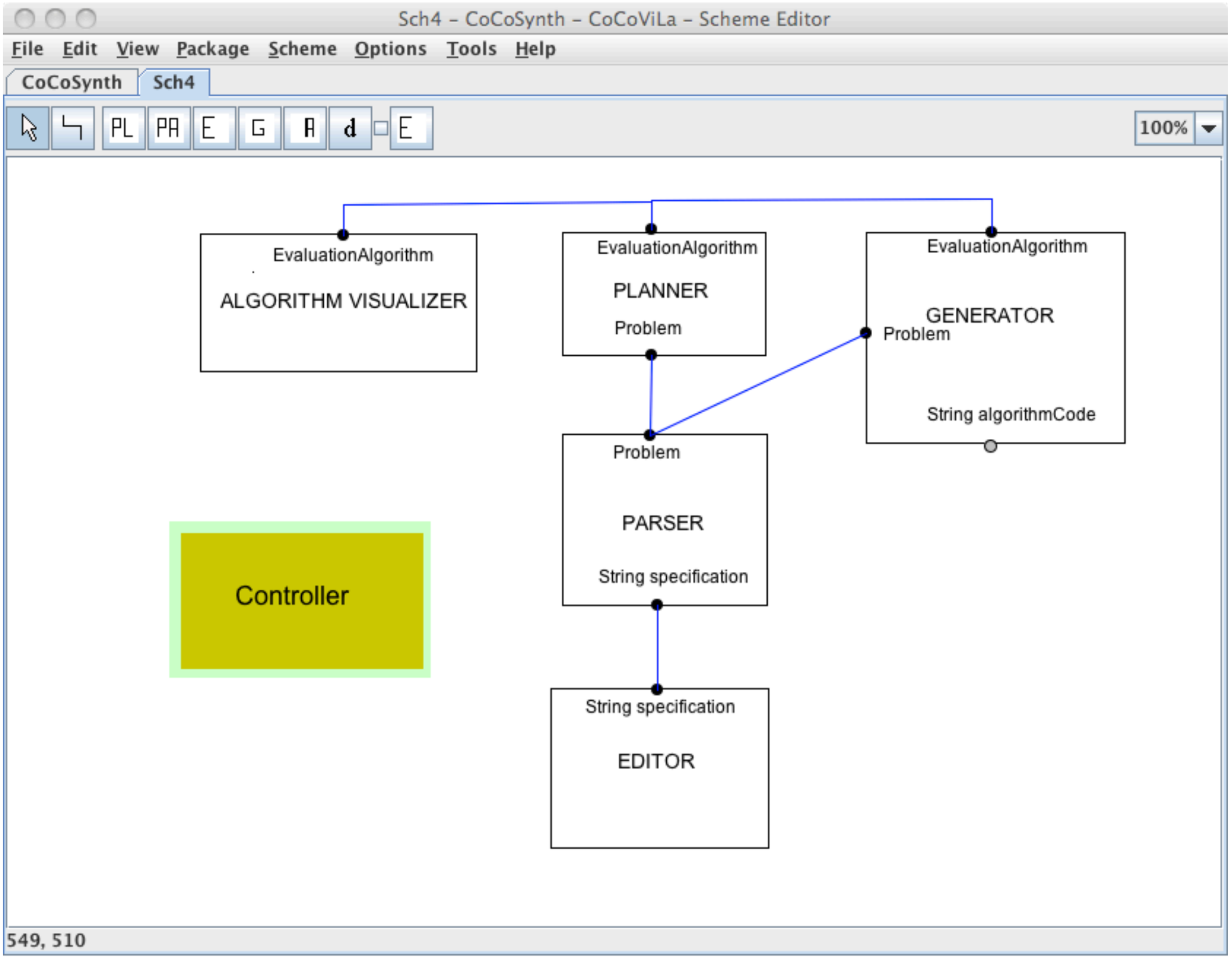
Metainterface of Parser

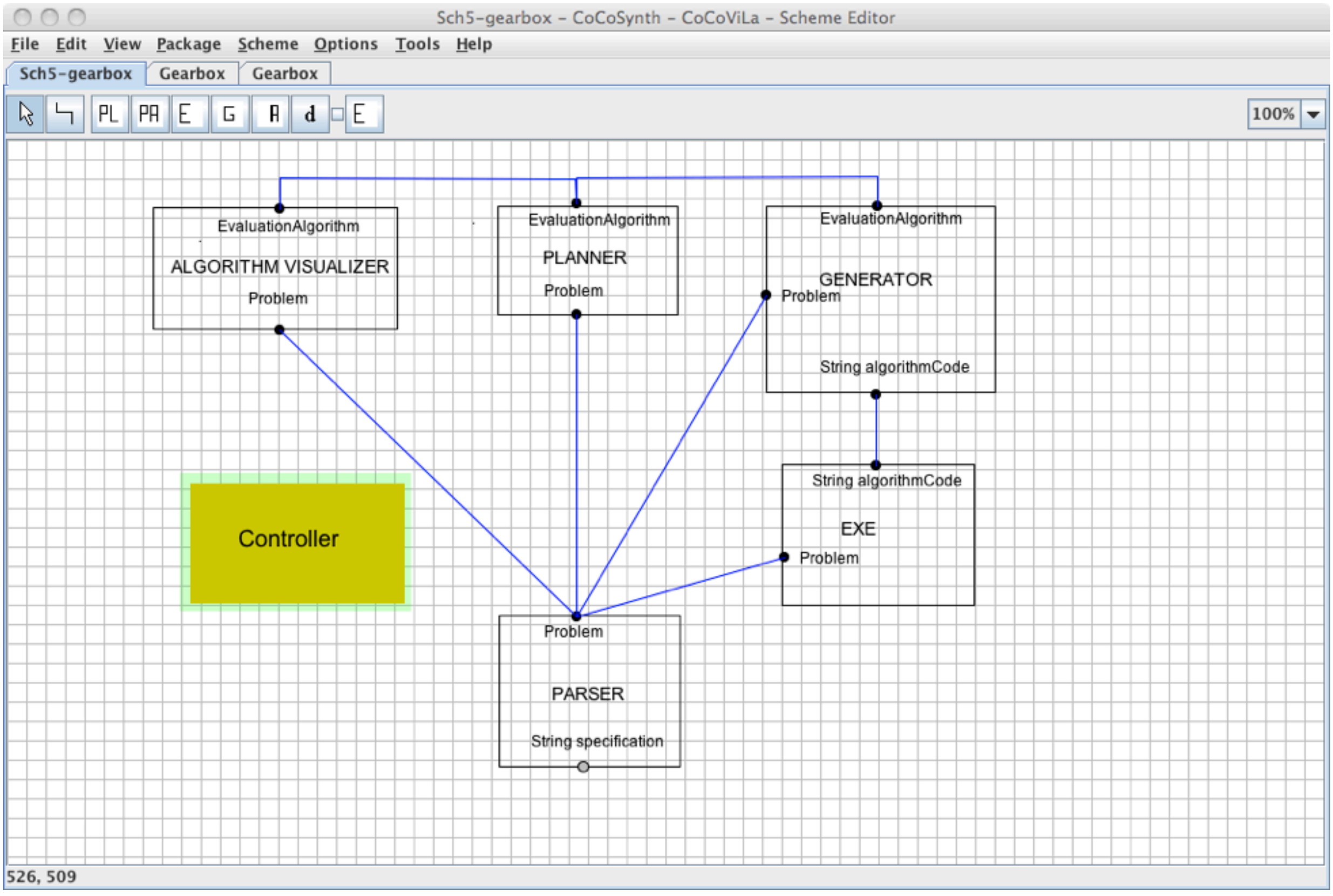
*/Users/enntougu/workspace/cocovila-packages/CoCoSynth/Par...

File Edit

```
1  import ee.ioc.cs.vsle.synthesize.ClassList;
2  import ee.ioc.cs.vsle.synthesize.ProblemCreator;
3  import ee.ioc.cs.vsle.synthesize.SpecParser;
4
5  public class Parser {
6          /*@ specification Parser {
7                  String packagePath;
8                  ClassList classList;
9                  String spec;
10                 ee.ioc.cs.vsle.synthesize.Problem
   problem;
11                 String mainClassName;
12                 spec -> mainClassName {getMainClassName};
13                 spec, mainClassName, packagePath ->
   classList {makeClasslist};
14                 classList, mainClassName -> problem
   {makeProblem};
15           }@*/
16         public String getMainClassName(String spec) {
17                 return SpecParser.getClassName( spec );
18         }
```


Sch3 – CoCoSynth – CoCoViLa – Scheme Editor
File Edit View Package Scheme Options Tools Help
CoCoSynth
Sch3
100%
EvaluationAlgorithm
ALGORITHM VISUALIZER
EvaluationAlgorithm
PLANNER
Problem
EvaluationAlgorithm
GENERATOR
Problem
String algorithmCode
Problem
PARSER
String specification
String specification
EDITOR
38, 93


Sch4 – CoCoSynth – CoCoViLa – Scheme Editor
File Edit View Package Scheme Options Tools Help
CoCoSynth
Sch4
100%
EvaluationAlgorithm
ALGORITHM VISUALIZER
EvaluationAlgorithm
PLANNER
Problem
EvaluationAlgorithm
GENERATOR
Problem
String algorithmCode
Problem
PARSER
String specification
Controller
String specification
EDITOR
549, 510



reede, 6, detsember 2013

Controller

- Controller is a large class that implements the user interface: drawing of diagrams and executing menu commands.
- Controller interacts with objects of a diagram by means of aliases and subtasks.
- The first implementation of Controller is essentially the existing CoCoViLa class `Editor`, where *action listeners of the menu commands are changed so that they call subtasks*. The subtasks perform required computations on the CCVL model.

Logic

- A specification is translated in a set of formulas of the form

  $S_1, \ldots S_k, x_1, \ldots x_m \rightarrow y \{F\}, \qquad (*)$

  where $x_1, \ldots x_m$ and y are names of variables of the specification, and $S_1, \ldots S_k$ are themselves formulas of the form $u_1, \ldots u_n \rightarrow v_1, \ldots v_r$ and are called **subtasks**.
- In constructive logic, the formula (*) is an implication, and its meaning is that realization of y can be computed, if realizations of $S_1, \ldots S_k, x_1, \ldots x_m$ are known.
- Subtasks represent goals for computations on the model ("compute y from $x_1, \ldots x_m$").
- F is a name of method that implements the formula (*), i.e. it is a realization of this formula.
- These formulas describe computability on the model (what can be computed from what).

  If the model includes n varibles, then it is possible to write 2^{2n} different subtasks. (A few of them are solvable.)

Logic of `Controller`

- Having a variable denoting the goal of synthesis, e.g. a variable *code* denoting the synthesized code, one can write a formula

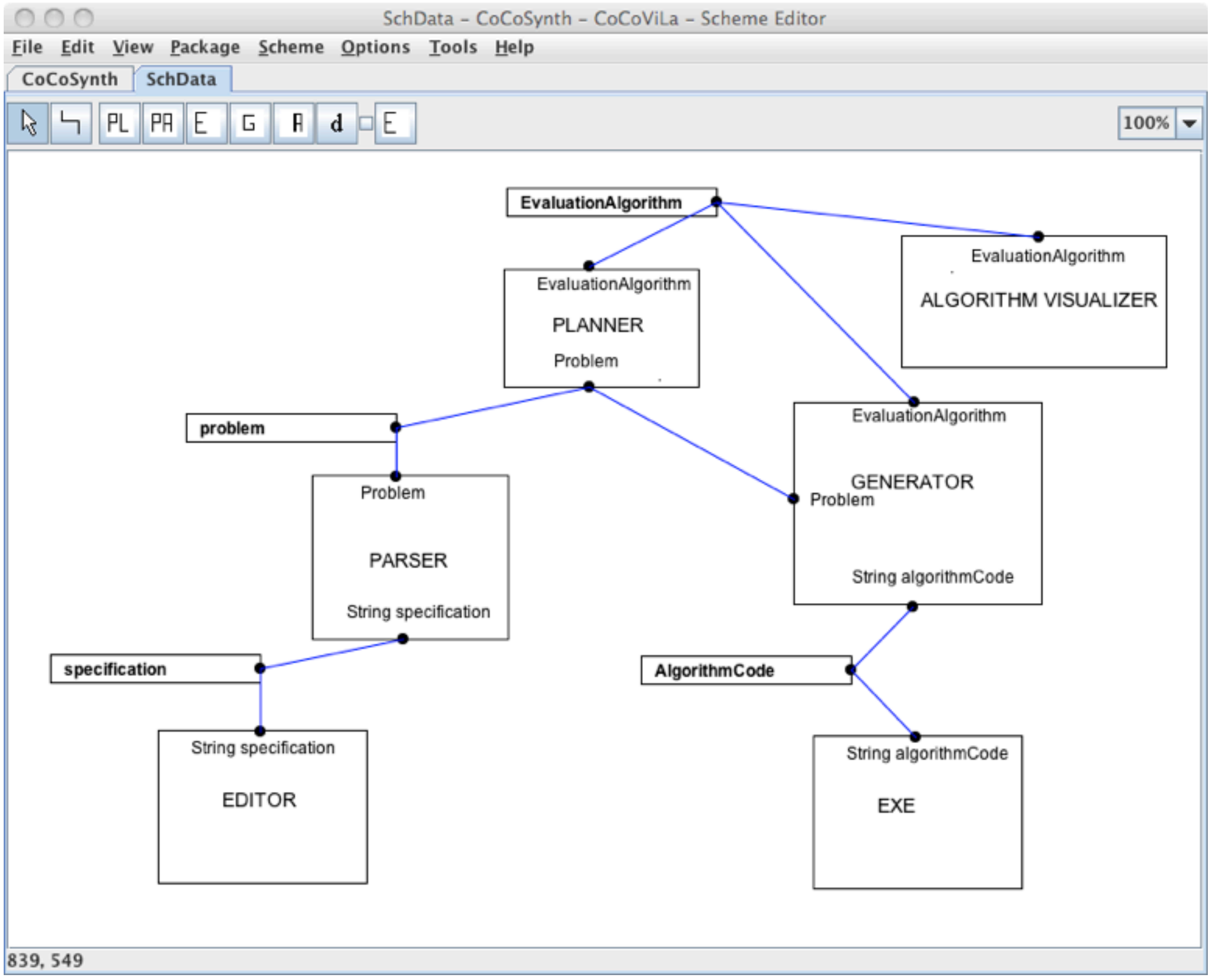
  $S_1, \ldots S_k, x_1, \ldots x_m$ -> *code* {*contollerMethod*},

  where $S_1, \ldots S_k$ denote the goals for computations performed by action listeners of widgets of user interface (of menu buttons etc.)

- A model is complete for a set of goals, if these goals are solvable on the model..

- Having a complete model for $S_1, \ldots S_k$, a user interface can be developed as a program of an automaton with actions on transitions given by $S_1, \ldots S_k$. This is convenient, for instance, if a user interface is specified by a statechart or any other state transition model.

Another representation of model



SchData – CoCoSynth – CoCoViLa – Scheme Editor

File Edit View Package Scheme Options Tools Help

CoCoSynth SchData

100%

Controller

EvaluationAlgorithm

EvaluationAlgorithm

PLANNER

Problem

EvaluationAlgorithm

ALGORITHM VISUALIZER

problem

Problem

PARSER

String specification

EvaluationAlgorithm

GENERATOR

Problem

String algorithmCode

specification

String specification

EDITOR

AlgorithmCode

String algorithmCode

EXE

11, 399

reede, 6, detsember 2013

Controller class

/Users/enntougu/workspace/cocovila-packages/CoCoSynth/Controller.java

File Edit

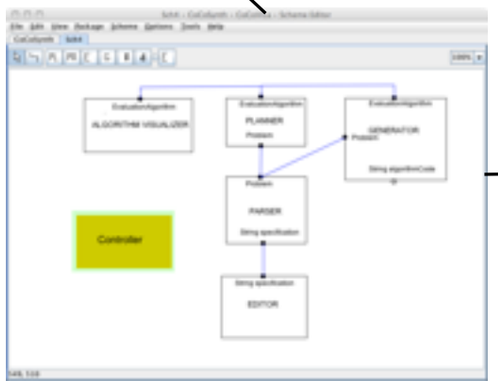
```
1  import java.util.concurrent.Callable;
2  import java.util.concurrent.CountDownLatch;
3
4  import javax.swing.SwingUtilities;
5
6  import ee.ioc.cs.vsle.api.Subtask;
7  import ee.ioc.cs.vsle.editor.Editor;
8
9  public class Controller {
10             /*@ specification Controller {
11                         void ready;
12                         String classPath;
13                         alias subtasksReady = (*.ready);
14                         alias spec = (*.spec);
15                         [spec -> subtasksReady], classPath -> ready {initGUI};
16             }
17             @*/
18
19             public void initGUI(final Subtask subtask, String cp) {
20                         final CountDownLatch latch = new CountDownLatch(1);
21
22                         Runnable runEditor = new Runnable() {
23                                     public void run() {
24                                                 Editor ed = Editor.createEditor();
25                                                 Runnable callback = new Runnable() {
26                                                             public void run() {
27                                                                         latch.countDown();
28                                                             }
29                                                 };
30                                                 ed.setSubtask(subtask, callback);
31                                                 ed.setVisible(true);
32                                     }
33                         };
```

Creating and using CCVL

Synthesied CCVL code

scheme>specification>compute all

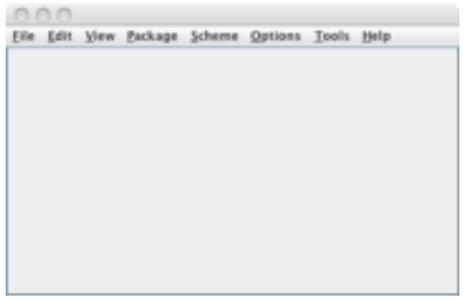
CoCoViLa



specify CCVL

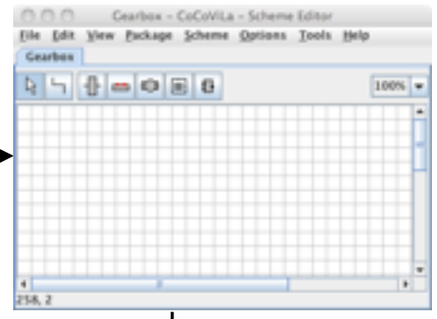
run

CCVL



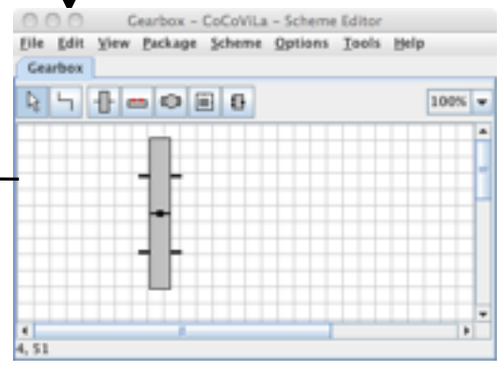
load Gearbox

CCVL



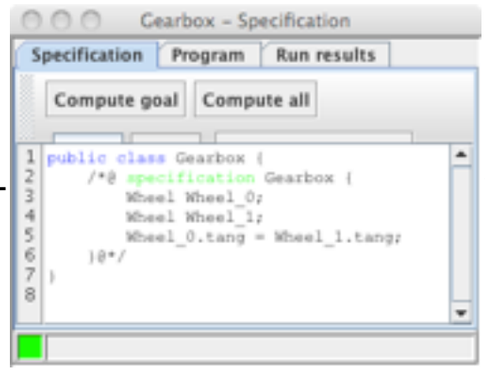
draw a diagram

CCVL



scheme>specification

CCVL



scheme>compute all

CCVL