A New Generation PEPA Workbench

Mirco Tribastone[*] Stephen Gilmore[†]

Abstract

We present recent developments on the implementation of a new PEPA Workbench, a cross-platform application for editing, analysing, and output management of PEPA models. The workbench is designed on top of the Eclipse API, allowing it to be plugged into the Eclipse IDE, as well as released as a standalone application. The new workbench employs improved static and dynamic analysis features and user-friendly approach to analysis of underlying CTMCs.

1 Introduction

Performance Evaluation Process Algebra (PEPA) [1] is an algebraic process-oriented language for modelling concurrent systems. Performance of PEPA models can be evaluated either by deriving the underlying Continous Time Markov Chain (CTMC) and calculating the long-run probability of the states of the chain or by extracting a set of Ordinary Differential Equations (ODE) from the model [2].

The major advantage of using high-level modelling languages is that the entire process of deriving and solving the underlying system can be automated by software tools. Since 1994 PEPA modelers have been provided with the PEPA Workbench [3], an application for managing PEPA models. The first release written in ML relied on external tools such as Matlab or Maple for obtaining the steady-state probability of the CTMC. The latest official version (Tabasco release, [4]) is an open-source cross-platform application written exclusively in Java incorporating numerical iterative solvers as well as features for performance evaluation such as state finder, utilisation and throughput analysis. We present a new generation PEPA Workbench which adopts the Eclipse API. Eclipse [5] is an open source project, providing a powerful integrated development environment for a large variety of programming and modelling languages such as Java, C/C++, Python, UML, etc. Moreover, Eclipse features a plug-in architecture which makes it an extensible platform for third-party programmers to support new tools, platforms and languages.

In this paper we show the first stages towards plugging PEPA tools into the Eclipse platform. We first present our refactoring of the previous PEPA workbench source code. Then we show the Eclipse extensions we implemented and compare them to the already existing functionalities. The paper is concluded by discussing ongoing work on this project.

[*]LFCS, School of Informatics, Edinburgh University. `Email: mtribast@inf.ed.ac.uk`
[†]LFCS, School of Informatics, Edinburgh University. `Email: stg@inf.ed.ac.uk`

2 The PEPA Workbench

The Tabasco release consists of a single project containing both the business model and the view model. For the sake of better manageability, a design choice was made to divide up the entire project into three different sub-projects. The *Core* project deals with the core logic for PEPA models. Services provided by this project include, for example, parsing, state space deriving, and model solving. A basic command-line user interface is also available within this project. The *PEPA Eclipse Plug-in* project is the Plug-in project containing all the necessary classes to provide the view logic of the workbench. Characteristics of this sub-project are discussed in detail in this paper. Finally, the *PEPA Help* project is concerned with help information to the end-user. Help is provided in three different forms: dynamic help linked to Eclipse windows, HTML-based help linked to the Eclipse main help section, and printer-friendly PDF reference guide.

2.1 The *PEPA Eclipse Plug-in* Project

All the data managed by an Eclipse instance is organized into a *workspace*, i.e. a collection of *projects* simultaneously managed by the Eclipse environment. Projects contain the resources (i.e., files) which can be authored by the user. Those resources can be manipulated using two main classes of tools, *editors* and *views*. The former follow the traditional open-save-close cycle for resource modification. The latter are typically used to navigate resources, modify properties of a resource and provide additional information on the resource being edited.

The Eclipse Project extends Eclipse by defining a new class of projects, the PEPA project, capable of dealing with PEPA models. A PEPA project is responsible for listening to events in resources and run *PEPA builders*, which in turn construct model objects (parse tree, state space) from the model source code.

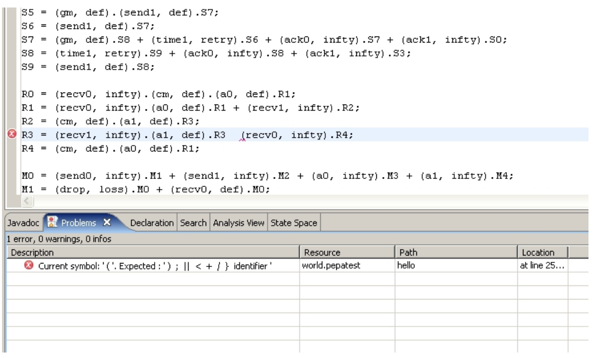
Unlike the Tabasco release of the PEPA Workbench, parse tree building follows the open-save-close model of the editor holding the model. When the editor is first opened, a parse tree is derived. The parse tree is then kept synchronised with the user's save action. The Eclipse *Problems* view is augmented to report syntax errors to the user, and associated error markers are also shown in the editor site to notify the position of the encountered error (see Fig. 1). A *Parse Tree* view is notified of changes in the parse tree. It reacts by updating a tree viewer consisting of the alphabets of the model processes.[1]

[1]The alphabet of a process is the complete set of action the process can perform. For example, if we consider the given model:

$$\begin{aligned} P &\overset{def}{=} (a,r).P1 + (b,s).P2 \\ P1 &\overset{def}{=} (c,t).P \\ P2 &\overset{def}{=} (d,u).P \\ Q &\overset{def}{=} (e,v).Q + (f,z).Q \end{aligned}$$

then the computed alphabets are as follows:

$$Alphabet(P) = Alphabet(P1) = Alphabet(P) = Alphabet(P2) = \{a,b,c,d\}$$
$$Alphabet(Q) = \{e,f\}$$

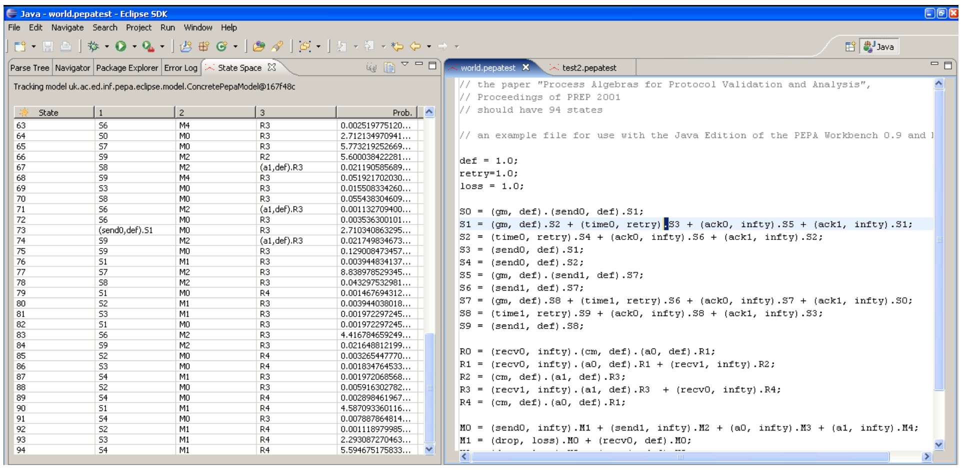


```
S5 = (gm, def).(send1, def).S7;
S6 = (send1, def).S7;
S7 = (gm, def).S8 + (time1, retry).S6 + (ack0, infty).S7 + (ack1, infty).S0;
S8 = (time1, retry).S9 + (ack0, infty).S8 + (ack1, infty).S3;
S9 = (send1, def).S8;

R0 = (recv0, infty).(cm, def).(a0, def).R1;
R1 = (recv0, infty).(a0, def).R1 + (recv1, infty).R2;
R2 = (cm, def).(a1, def).R3;
R3 = (recv1, infty).(a1, def).R3  (recv0, infty).R4;
R4 = (cm, def).(a0, def).R1;

M0 = (send0, infty).M1 + (send1, infty).M2 + (a0, infty).M3 + (a1, infty).M4;
M1 = (drop, loss).M0 + (recv0, def).M0;
```

1 error, 0 warnings, 0 infos

Description	Resource	Path	Location
Current symbol: '('. Expected : ') ; || < + / } identifier '	world.pepatest	hello	at line 25...

Figure 1: The Eclipse Problems view highlights a PEPA syntax error



Tracking model uk.ac.ed.inf.pepa.eclipse.model.ConcretePepaModel@167f48c

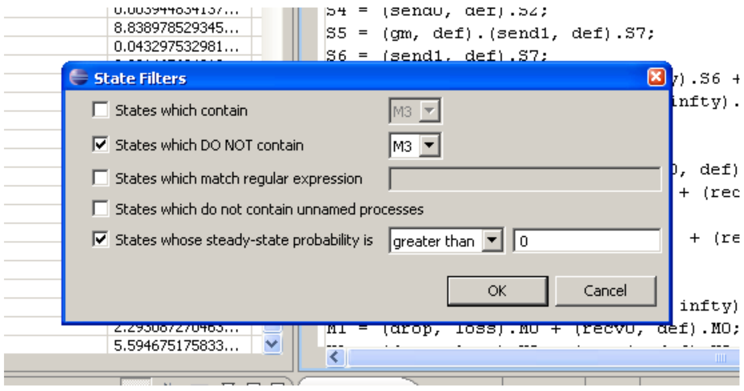
State	1	2	3	Prob.
63	S6	M4	R3	0.002519775120...
64	S0	M0	R3	2.712134970941...
65	S7	M0	R3	5.773219252669...
66	S9	M2	R2	5.600038422281...
67	S8	M2	(a1,def).R3	0.021190585689...
68	S9	M4	R3	0.051921702030...
69	S3	M0	R3	0.015508334260...
70	S8	M0	R3	0.055438304609...
71	S6	M2	(a1,def).R3	0.001132709400...
72	S6	M0	R3	0.003536300101...
73	(send0,def).S1	M0	R3	2.710340863295...
74	S9	M2	(a1,def).R3	0.021749834673...
75	S9	M0	R3	0.129008473457...
76	S1	M1	R3	0.003944834137...
77	S7	M2	R3	8.838978529345...
78	S8	M2	R3	0.043297532981...
79	S1	M0	R4	0.001467694312...
80	S2	M1	R3	0.003944038018...
81	S3	M1	R3	0.001972297245...
82	S1	M0	R3	0.001972297245...
83	S6	M2	R3	4.416784659249...
84	S9	M2	R3	0.021648812199...
85	S2	M0	R4	0.003265447770...
86	S3	M0	R4	0.001834764533...
87	S4	M1	R3	0.001972068568...
88	S2	M0	R3	0.005916302782...
89	S4	M0	R4	0.002898461967...
90	S1	M1	R4	4.587093360116...
91	S4	M0	R3	0.007887864814...
92	S2	M1	R4	0.001118979985...
93	S3	M1	R4	2.293087270463...
94	S4	M1	R4	5.594675175833...

```
// the paper "Process Algebras for Protocol Validation and Analysis",
// Proceedings of PREP 2001
// should have 94 states

// an example file for use with the Java Edition of the PEPA Workbench 0.9 and 

def = 1.0;
retry=1.0;
loss = 1.0;

S0 = (gm, def).(send0, def).S1;
S1 = (gm, def).S2 + (time0, retry).S3 + (ack0, infty).S5 + (ack1, infty).S1;
S2 = (time0, retry).S4 + (ack0, infty).S6 + (ack1, infty).S2;
S3 = (send0, def).S1;
S4 = (send0, def).S2;
S5 = (gm, def).(send1, def).S7;
S6 = (send1, def).S7;
S7 = (gm, def).S8 + (time1, retry).S6 + (ack0, infty).S7 + (ack1, infty).S0;
S8 = (time1, retry).S9 + (ack0, infty).S8 + (ack1, infty).S3;
S9 = (send1, def).S8;

R0 = (recv0, infty).(cm, def).(a0, def).R1;
R1 = (recv0, infty).(a0, def).R1 + (recv1, infty).R2;
R2 = (cm, def).(a1, def).R3;
R3 = (recv1, infty).(a1, def).R3  + (recv0, infty).R4;
R4 = (cm, def).(a0, def).R1;

M0 = (send0, infty).M1 + (send1, infty).M2 + (a0, infty).M3 + (a1, infty).M4;
M1 = (drop, loss).M0 + (recv0, def).M0;
```

Figure 2: The State Space View

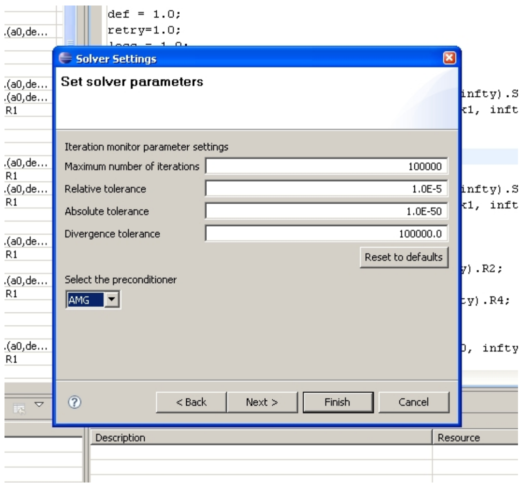
As in the Tabasco release, the state space derivation stage is not synchronised with the state of the editor. This design choice was made in order to cope with large models more efficiently. In fact, state space derivation of such models may be a long-running task which may dramatically reduce the responsiveness of the user interface. A *State Space* view is notified by changes in the state space of the model (see Fig. 2). The view allows the user to navigate, sort and filter the state space. Filters are provided in the form of user-friendly, high-level rules such as *Filter states which do/do not contain reference* or *Filter states which can perform outgoing/incoming activities* (see Fig. 3). The *PEPA Eclipse Plug-in* project provides decorators to export the generated state space into various formats. An Eclipse wizard is already available to generate input data for the MRMC (Markov Reward Model Checker) tool [6].

The *PEPA Eclipse Plug-in* project uses the Matrix for Java Toolkit [7] as the engine for iterative solution of the underlying Markov chain of the model. An Eclipse wizard is provided to guide the user through the process of model solving (see Fig. 4). A monitor is also implemented in order to report to the user information about the current iteration (iteration number, residual and other metrics). The State Space view is also designed to listen to changes in the


State Filters

- [ ] States which contain M3
- [x] States which DO NOT contain M3
- [ ] States which match regular expression
- [ ] States which do not contain unnamed processes
- [x] States whose steady-state probability is greater than 0

Figure 3: Filtering rules available with the State Space View


Solver Settings

Set solver parameters

Iteration monitor parameter settings

Maximum number of iterations	100000
Relative tolerance	1.0E-5
Absolute tolerance	1.0E-50
Divergence tolerance	100000.0

Select the preconditioner: AMG

Figure 4: Page of the solver wizard gathering information about solver parameters and preconditioner

solution of the model, reacting by updating a sortable and filterable *Probability* column which shows the long-running probability of each state.

3 Conclusion and Ongoing Work

In this paper a brief overview of the current development state of a new Eclipse-based version of the PEPA Workbench has been provided. We envisage two main areas regarding the future work on this project. With regards to the *Core* project, we aim at improving the static and dynamic analysis features. The purpose is to help the user discover potential errors in the model as early as possible. We are currently working on issues such as the automatic detection of self-loops, unused or unreachable definitions (*dead code*), and early identification of transient states. As for the *PEPA Eclipse Plug-in* project, our goal is to incorporate tools to automatically map PEPA models to the underlying ODE representation.

References

[1] J. Hillston, *A Compositional Approach to Performance Modelling.* Cambridge University Press, 1996.

[2] M. Calder, S. Gilmore, and J. Hillston, "Automatically deriving ODEs from process algebra models of signalling pathways," in *Proceedings of Computational Methods in Systems Biology (CMSB 2005)* (G. Plotkin, ed.), (Edinburgh, Scotland), pp. 204–215, Apr. 2005.

[3] S. Gilmore and J. Hillston, "The PEPA Workbench: A Tool to Support a Process Algebra-based Approach to Performance Modelling," in *Proceedings of the Seventh International Conference on Modelling Techniques and Tools for Computer Performance Evaluation*, no. 794 in Lecture Notes in Computer Science, (Vienna), pp. 353–368, Springer-Verlag, May 1994.

[4] "PEPA Workbench Tabasco release." `http://www.dcs.ed.ac.uk/pepa/tools/`.

[5] "Eclipse." `http://www.eclipse.org`.

[6] J.-P. Katoen, M. Khattri, and I. S. Zapreev, "A Markov Reward Model Checker," in *Proceedings of the Second International conference Quantitative Evaluation of Systems (QEST)*, pp. 243–244, IEEE CS Press, 2005.

[7] "Matrix Toolkit for Java." `http://rs.cipr.uib.no/mtj/`.