Executive Summary

Modelling and analysing complex cellular pathways that underly biological activity is essential for improving our understanding of health and disease. Pathway Logic (PL)[1] is a symbolic approach for reasoning about biological entities and processes based on the rewriting logic system Maude [1]. The Pathway Logic Assistant (PLA) is a tool developed by SRI International for browsing and analysing PL models using various symbolic analysis methods. PL has proven useful for modelling a variety of biological processes including signal transduction, metabolic pathways, and protease networks.

In order to effectively exploit the inferential power of PLA, users are currently required to have a good understanding of the underlying rewrite logic. This project produces a graphical rule editor that could be used with integration to PLA to make it more accessible to users with no background in formal logic, allowing them to express rules using a natural graphical representation, which is automatically mapped to the underlying logic.

To achieve a successful implementation, this project has explored and researched the following main areas:

Pathway Logic and Maude: PL and Maude are the core logic that the new graphical editor needs to communicate with. PL and Maude can be considered the conceptual layer of the new system architecture.

Model representation languages and Interoperability tools: IOP is the current interoperability platform used for exchanging models and connecting various PL tools in PLA. The new tool has been implemented with full integration to the current platform to ensure reliability.

Pathway visualisation tools: This project has explored the opportunity to utilize and integrate one of the available powerful pathway visualization tools to aid the analysis process to the utmost. This investigation has resulted in integration unfeasibility which has led to developing a fully tailored graphical editor solution for PLA. User interface represents the presentation layer of the new system.

In this project:

- I have created an intuitive graphical editor that will make PLA more accessible to users with no formal logic background. This will make PLA a unique reasoning facility with the aid of powerful graphical visualisation which other systems lack. It will reduce the time required for modelling biological networks dramatically with models as closest as possible to biologists mental model.
- I explored the opportunity to integrate a number of available pathway visualisation tools which resulted in the production of a new graphical tool.
- I exploited Java Swing and AWT GUI toolkits to represent biological entities with most flexibility. The classes developed for this editor can be generalised to be used in other graphical-based applications.

[1] See pl.csl.sri.com

Acknowledgments

I would like to thank my project supervisor, Dr Oliver Ray, for the valuable information, guidance and support during this project. Thanks and appreciation also goes to my personal tutor, Dr Tim Kovacs, for his encouragement and support. Special thanks goes to Dr Carolyn Talcott, program director of the symbolic systems biology group at SRI International and others in her team for their great cooperation and support to make the integration between the new editor and PLA system a success. My deepest thanks go to my family, for their love, understanding and continuous support.

Table of Contents

1 Introduction

Analysing biological processes like metabolism and signal transduction has become a critical necessity for making inferences and proving experimental hypotheses. Pathway Logic (PL), is a symbolic approach for reasoning about biological entities and processes based on rewriting logic formalism called Maude [1]. PL is used mainly to formalise biologists' mental models and analyse experiments outcomes. Pathway Logic Assistant (PLA) is a tool created by SRI International[1] for working on PL model by applying various symbolic systems analysis methods. PLA has become a handful tool for many biologists as it provides reasoning and inferences power using rewrite theories. With all of its great advantages, both biologists users and SRI International have proposed a number of improvements that may empower PLA to overcome some limitations and hence reduces the overhead. Most obvious requirements are concentrated towards the graphical representation and model editing.

This project is a proposal to address the various issues with PLA to achieve, not only a satisfactory implementation, but in addition, to have innovative ways that simplifies working with PLA to the utmost.

The aims and objectives, deliverables and added values of this project are summarised in the following sections, followed by the scope of this report and how this document is structured.

1.1 Aims and Objectives

The project aims to assist scientists – mainly biologists – to prove and test biological hypotheses and to provide the incentive to propose new hypotheses. This may lead to significant discoveries and innovations in the field of biology in particular, and science in general.

To achieve this target, the project will provide an integrated, intuitive, and easy to use graphical tool to assist biologists in editing logical rules for Pathway Logic networks. This would make the Pathway Logic approach more accessible to users with no background in formal logic and would facilitate the construction and modification of large models.

The tool will be empowered by user-friendly facilities like drag-drop and seamless import and export of models for graphical editing. It will integrate and adapt intuitive graphical representation of rules for more enhanced visualization. The tool will act as a powerful aid for reasoning to better understand complex biological systems and accelerate the design of experiments. The new rule editor will be part of the current PLA and thus it will require to integrate and communicate to PLA environment seamlessly.

The project will evaluate the current process and modelling procedures practiced by biologists and will undergo evaluation process to make sure biologists concerns and demands are all addressed.

[1] SRI International is an independent research institute founded in the US and is specialised in conducting R&D for both government and business agencies, website: www.sri.com.

1.2 Deliverables and Added Value

The project provides a graphical user interface software system that works with integration to the current system (PLA). The software is built on the existent PLA tool with the trend to improve and address it limitations. The software will be delivered with some important documentation like user manual and technical specifications. Training sessions might be produced depending on user demands.

This project will offer a unique reasoning facility with the aid of powerful graphical visualisation which other systems lack. It will allow biologists to more easily harness the advanced reasoning capabilities offered by Pathway Logic and visualise the solutions that are computed. This will make results more understandable to biologists. It will reduce the time required for modelling biological networks dramatically with models as closest as possible to biologists mental model. Thus it will accelerate experiments and hence approaching anticipated results more quickly. This way hypotheses can be simpler to prove and test.

Moreover, further work can be done in future to adopt generalisation of experiments editing. This may lead to discoveries if applied in other fields of science like chemistry where chemical reactions could be edited graphically. The new tool has been developed using Java graphical toolkits. The created classes and packages can be also generalised and adopted for other graphical applications.

1.3 This Report

This document aims at providing the necessary background and walks through the graphical editor implementation lifecycle in order to provide a clear drive towards the real value of this project via evaluation and conclusion.

The report starts by reviewing the literature to gain better understanding. PLA is the main concern in this project, so before reviewing the literature for PLA, a number of important concepts are reviewed to better understand PLA. Since PLA combines logic and graphical interface aspects, Background chapter has been broken down into three main areas of interest: logic (the core PL), interoperation (connecting logic to visualisation), visualisation (graphical part). Some other important concepts are discussed like model representation formats and biology data sources. After reviewing all these pre-requisite knowledge, PLA is discussed in the last section of Background chapter.

Understanding PLA will make it easier to present the project motivation. Motivation chapter spotlights the current PLA representations; Petri nets and Maude. It then motivates the project by outlining use cases of some PLA usage scenarios. This chapter ends by discussing some initiatives for improving the current PLA representation methods.

The design steps of the new editor is then discussed in the Designing a Graphical Rule Editor chapter. This chapter charts the current and proposed processes of creating and editing rules, it then designs the architecture of the system. Detailed design specifications and a proposed GUI design with

preliminary UML class diagram are then explained. Critical planning considerations are finally discussed regarding the development environment and integration with PLA requirements.

After this critical planning, Implementing the Editor chapter takes the reader through the implementation steps; starting from setting up the development environment, to implementing the PLA integration, and finally building the GUI.

Analysis of results chapter provides a detailed procedures for application scenarios each of which is followed by a discussion on results obtained. The chapter ends with a summary of the results obtained.

The developed tool is evaluated by surveying current PLA users and new editor potential users. Some other evaluation methods are also outlined. The report ends with a conclusion and a discussion on future directions.

2 Background

Pathway Logic Assistant is a graphical tool that assists biologists in analysing biological models and applying reasoning by exploiting the underlying symbolic systems biology logic: Pathway Logic. This definition contains a number of important terminologies that we need to explore to better understand PLA. To decode these concepts, it is important to categorise them into broader areas to see where each concept rests. In the following sections, PLA is explained after understanding its logic part, model exchange formats, how its internal components are interoperated, its visualisation part, and biology data sources it links to.

2.1 Logic in Systems Biology

Pathway logic is the underlying logic of PLA. Pathway Logic is a symbolic systems biology formalism based on rewriting logic called Maude.

2.1.1 Symbolic Systems Biology and Rewriting Logic

Traditional research methods in biology involve studying biological processes as isolated parts. In contrast, **Systems biology** is a research field were biological processes – with different levels of abstractions – and multiple viewpoints are studied as integrated systems. This inter-disciplinary paradigm is based on obtaining and analysing experimental data from various sources.

Symbolic systems biology (SSB) is an emergent research stream in systems biology in which biological components and rules of their interaction are represented symbolically. This research area is concerned with qualitative reasoning in terms of logical abstractions obtained from algebraic and computational frameworks [2] by applying formal logic-based methods. In recent years, symbolic systems biology has been applied to natural systems, particularly for modelling molecular networks [3].

A sample of SSB formalisms, which include Petri nets, statecharts, pi-calculus, life sequence charts, membrane calculi, hybrid systems and Rewriting Logic, is discussed in [4]. As an example, Petri nets are a mathematical formalism, with a natural graphical representation, initially proposed for modelling concurrent and distributed systems; while hybrid systems provide formalisms for modelling and analysing biological systems evolution over time using differential equations [5]. In this project, we are concerned with the SSB methodology of PL, which is based on a rewriting logic system called Maude [1].

> **Rewriting Logic** – is a symbolic formalism in which system states are represented as algebraic *terms*, and transitions between states are determined by *rewrite rules*. Rules represent reactions and dependencies between molecules and they transform (rewrite) systems from one state to another.

Thus, rewriting logic consists of two parts: *equational logic* that specifies system data types and functions; and rewrite rules that specifies how systems may change.

Rewriting logic provides both a *logical framework* (a *metalogic*) for representing and implementing other logics; and a *semantic framework (a metalanguage)* for specifying systems and languages [6]. This makes it a suitable logic for executable specification and analysis of concurrent, distributed, and also mobile systems. One of its valuable capabilities is that rewriting logic supports reflection; where it expresses its metatheory at the object level and manipulates its future behaviour while examining intermediate results. [7] gives a detailed proof that rewriting logic is reflective through the use of Maude language as an experimental vehicle.

At this point, it is useful to understand the relevancy of these concepts to better understand PL.

> According to **[4]**, Pathway Logic is "*a symbolic systems biology approach to the modeling and analysis of molecular and cellular processes based on rewriting logic*".

The rewriting logic formalism used by PL is Maude language. Maude and PL are discussed in more details in later sections.

2.1.2 Executable Models and Symbolic Analysis

In order to get a better intuition about Maude and PL before moving into their details, it is useful to understand the symbolic systems **models** and the various kinds of model analysis methods available.

Developing symbolic systems models requires selecting the suitable abstractions – to represent biological networks – and to combine models into integrated systems. This can be achieved by adopting well-defined semantics languages that produce *executable* models. These models describe biological systems as states and rules from which transition graphs can be derived. Consequently, different ways of system evolvement can be discovered by finding *pathways* (starting from an initial state to find all reachable states following connections between states; i.e. rules). A specific pathway is selected among all possible pathways based on an *execution* strategy adopted by the system.

Executable models can be analysed in different ways [4]: static analysis, forward simulation or prototyping, forward search, backward search, constraint solving, meta-analysis and model checking. These methods provide the ability to explore models, inferring control flow and dependencies between model elements, search and investigate if pathways satisfy a specified property, and moreover provide the ability to map models to other formalisms that offer further analysis tools.

Model checking – for instance – is an approach that is widely used in the computer science community and spread to applications in life sciences as well. It is used for verification of models in relation to a specified property. [8] simply defined model checking as following:

> "Given a model $\boldsymbol{M}$ and a property $\boldsymbol{\varphi}$, an algorithm is defined that verifies whether $\boldsymbol{\varphi}$ is a property of $\boldsymbol{M}$ (denoted $\boldsymbol{M} \vDash \boldsymbol{\varphi}$)."

In systems biology, model checking is used to determine reachable pathways that satisfy a property starting from a given initial state. Counter examples are returned if verification failed. An example of a property might be: molecule x is never produced after molecule y. A counter example is: a pathway in which molecule y is produced before x.

Understanding symbolic analysis methods of executable models makes it easier to understand the core logic of PL; Maude.

2.1.3 Maude as Equational and Rewriting Logic

Maude is a high-level, formal language that provides a high-performance environment and a set of tools for modelling based on rewriting logic (which contains equational logic).

The formal language of equational logic – supported by Maude – is constituted of terms. Terms build up the sides of equations. A term can be a variable, a constant symbol, or a function symbol (operation) applied to terms. Each function has an arity that indicates the number of arguments.

To get some intuition about rewriting using equational logic, the following example helps understand the basic concepts and terminologies.

Example:

A language $\mathcal{L}$ with constant symbols $\mathcal{C} = \{\mathbf{0}\}$ and function symbols $\mathcal{F} = \{+, \boldsymbol{n}\}$, having:

$+$: addition function
(takes 2 arguments; arity = 2, it is binary)

$\boldsymbol{n}$: next 2-multiple function
(takes 1 arguments; arity = 1, it is unary)

Note that constants and functions that assign a value to terms are called *constructors*, where
$\boldsymbol{n}(\mathbf{0}), \boldsymbol{n}(\boldsymbol{n}(\mathbf{0})), \boldsymbol{n}\left(\boldsymbol{n}(\boldsymbol{n}(\mathbf{0}))\right)$,... means 2,4,6,...

Then this language can be expressed in terms of its constant and function symbols as following:

$$\mathcal{L} = \{+, \boldsymbol{n}, \mathbf{0}\}$$

Some possible terms are: 0, $\boldsymbol{n}(\mathbf{0})$, $\boldsymbol{x}$, $\boldsymbol{n}(\boldsymbol{y})$, $\boldsymbol{x} + \mathbf{0}$, and $\boldsymbol{n}(\mathbf{0}) + \boldsymbol{y} + \boldsymbol{n}(\boldsymbol{x})$

Let us assume the equations:

$\mathbf{0} + \boldsymbol{x} = \boldsymbol{x}$ (1)

$\boldsymbol{n}(\boldsymbol{x}) + \boldsymbol{y} = \boldsymbol{x} + \boldsymbol{n}(\boldsymbol{y})$ (2)

The term: $\boldsymbol{n}\left(\boldsymbol{n}(\boldsymbol{n}(\mathbf{0}))\right) + \boldsymbol{n}(\mathbf{0})$

corresponding to $(\mathbf{6} + \mathbf{2})$ is a *data term*; a term with no variables.

This data term can be *rewritten* – using equations 1 and 2 – until a *canonical representation* is achieved; i.e. getting a term with constants and constructors only.

Matching this term with equation 2, we get $\boldsymbol{x} \coloneqq \boldsymbol{n}(\boldsymbol{n}(\mathbf{0}))$ and $\boldsymbol{y} \coloneqq \boldsymbol{n}(\mathbf{0})$

By substituting the *sub-terms* $\boldsymbol{x}$ and $\boldsymbol{y}$ in the right-hand side of equation 2, we obtain:
$\boldsymbol{n}(\boldsymbol{n}(\mathbf{0})) + \boldsymbol{n}(\boldsymbol{n}(\mathbf{0}))$, repeating the same process:
$\boldsymbol{x} \coloneqq \boldsymbol{n}(\mathbf{0})$ and $\boldsymbol{y} \coloneqq \boldsymbol{n}(\boldsymbol{n}(\mathbf{0}))$
$\boldsymbol{n}(\mathbf{0}) + \boldsymbol{n}\left(\boldsymbol{n}(\boldsymbol{n}(\mathbf{0}))\right)$,
$\boldsymbol{x} \coloneqq \mathbf{0}$ and $\boldsymbol{y} \coloneqq \boldsymbol{n}\left(\boldsymbol{n}(\boldsymbol{n}(\mathbf{0}))\right)$
$\mathbf{0} + \boldsymbol{n}\left(\boldsymbol{n}\left(\boldsymbol{n}(\boldsymbol{n}(\mathbf{0}))\right)\right)$,

Matching and applying equation 1:
$\boldsymbol{x} \coloneqq \boldsymbol{n}\left(\boldsymbol{n}\left(\boldsymbol{n}(\boldsymbol{n}(\mathbf{0}))\right)\right)$

and the term becomes:
$\boldsymbol{n}\left(\boldsymbol{n}\left(\boldsymbol{n}(\boldsymbol{n}(\mathbf{0}))\right)\right)$
(which corresponds to 8)

Data types used in equations are called *sorts* and functions are called *operations*. Sorts, operations, and equations compose the *equational theory* or system *signature* which describes system data types and state. Subsequently, a *rewrite theory* is composed of system signature, labels and rules.

A rewrite rule takes the form: $label : t \Longrightarrow t' \ if \ condition$, where t and t' are terms, and condition is a boolean term to be satisfied. Rewriting with rules follows the same procedure applied in rewriting with equations. But rules rewriting has the advantage that it is used to describe system evolution over time. In addition, rules rewiring provides the ability to handle infinite and non-deterministic behaviours, while equational rewriting is used to calculate values of functions and - therefore – it should terminate.

[9] considered the equational logic – supported by Maude – influenced by OBJ3; a high-level declarative specification, prototyping, and functional language [10]. Maude is used to express models – like biological processes – as rewrite theories in the context of the semantic framework of the rewriting logic. It offers the model analysis methods supported by symbolic systems like search, model checking, and reflection. In addition, Maude features Associativity, Commutativity, and Identity (or Idempotence) axioms (ACI property) that can be easily adopted in its notation. Maude language[1] is based on defining *modules*.

The equational part; signature, is specified in *functional modules*; (a code block that starts with fmod and ends with endfm). The system behaviour; rules, is specified in *system modules* (a code block that starts with mod and ends with endm). [4] introduces Maude notation via a simple example; *Magic Marbles*. The example gives an intuition about how representation and analysis of concurrent systems' structure is performed using Maude.

Following is a simple Maude module on positive numbers:

```
fmod POS is
        sort Pos .
        op 1 : -> Pos .
        op p_ : Pos -> Pos .
        op _*_ : Pos Pos -> Pos .
        vars X Y : Pos .
        eq 1 * X = X .
        eq p X * Y = p (X * Y) .
endfm
```

Where `sort` defines a data type, `op` defines an operation, `vars` define variables of type `Pos`, and `eq` defines the equations to be applied on terms. `1` is a constant because it takes no arguments. `1` and `p` are constructors as they generate data.

This module can be used to *reduce* a term. For instance, if following is typed in Maude:

```
Maude> reduce p(1) + p(p(1)) .
```

Maude performs rewriting using the equations, and outputs:

```
reduce in POS : p 1 * p p 1 .
rewrites: 2 in 0ms cpu (0ms real) (~ rewrites/second)
result Pos: p p p 1
```

[1] Maude manual, documentation and publications are available from the official website maude.cs.uiuc.edu

Having had a fair idea about Maude language and rewriting logic, let us see how Maude is exploited in Pathway Logic for modelling biological processes.

2.1.4 Pathway Logic

PL has been defined as one symbolic systems biology approach for modelling biological processes using the rewriting-logic-based formalism Maude [4]. It can be seen as a general framework for representing and reasoning about complex biological interactions from metabolic networks to signalling pathways and genetic interactions. Each network defines a set of basic transformations, called reactions, between a set of basic biochemical entities, called occurrences in PL (but called species or physical entities in some other modelling languages like BioPAX[1]). Depending on the biological context, occurrences may denote enzymes or metabolites, molecules or ions, ligands or receptors, proteins in various states of posttranscriptional modification, or complexes formed from other occurrences. Rules represent biochemical transformations and dependencies between occurrences and they transform (rewrite) systems from one state to another.

Recall that Rewriting Logic is a symbolic formalism in which system states are represented as algebraic terms and transitions between states are determined by rewrite rules. Referring to the rule matching computations performed in rewriting logic, in biological processes context, initial data terms represent initial system states, and the computations (transformations) applied by matching with sub-terms in rule sides represent pathways. Biological systems' behaviour and their response to physiological transformations and stimuli are handled by altering genes and protein activities. Collectively, these modifications (i.e. interactions or concurrent computations) form what is called a signalling network. Finding signalling pathways and regulation methods on such networks is a key effort in biomedical research [11].

PL can be used to model both biological molecules and processes in different abstraction levels. For example, assume two protein molecules; protein A and protein B. At one level of abstraction, it is adequate to know that protein A interacts with protein B in a specific signalling pathway. In other level, it might be required to model the internal structure of proteins A and B to study in depth how the interaction is regulated [2].

PL uses a variety of available symbolic analysis methods; like model execution, model checking, search, and mapping to other formalisms such as Petri nets for further analysis. Biological processes like signal transduction or metabolism are modelled in PL as collections of rules representing process steps. A PL knowledge base (KB) is composed of these rewrite rules together with data type specifications and evidences to justify each

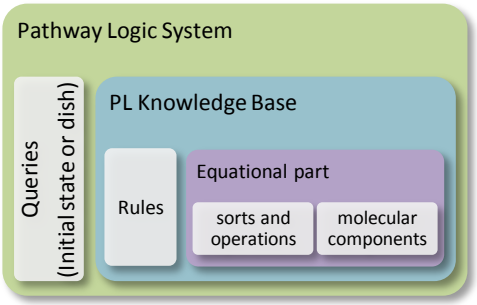


Figure 1 Components of a Pathway Logic Model System

[1] BioPAX and other biological model representation languages are discussed in section 2.2

rule. PL uses information from the KB to generate models from a given initial state (called a "dish"). Figure 1 illustrates the different components of a PL model system.

The equational part determines a controlled vocabulary for representing basic biological entities, modifications and locations. Biological entities – like proteins and chemicals – interact with each other in various locations with respect to the cell resulting in assembled complexes. In this context, entities sorts in PL include: Cell, Location, Chemical, Protein, and Complex. Equational part also provides operations for representing molecular and cellular states. For example the term **`[Raf1 - act]`** represents the protein **`Raf1`** in an activated state.

Molecular components; i.e. the second component of the equational part in the KB, are those entities that are specified for grouping proteins in families. Examples of such components are proteins, genes, and chemicals. Each component is declared by a sort and metadata that can be linked to standard database. Following example is from [4] showing metadata:

```
op EgfR : -> Protein [metadata "(\
(spname EGFR_HUMAN)\
(spnumber P00533)\
(hugosym EGFR)\
(category Receptor)\
(synonyms \"Epidermal growth factor
receptor precursor\" \
\"Receptor tyrosine-protein kinase
ErbB-1, ERBB1 \"))"] .
```

The heart of the system is the rules part, which contains the rewrite rules representing reaction steps. These rules are curated from literature with evidence to justify each rule. Queries contain terms representing initial system states (i.e. dishes) of interest. An initial state is often used to model an experimental setup – the state of a cell and any external stimuli.

Understanding these concepts is very important to provide the best features and functionalities to serve biologists in a satisfactory way.

2.2 Biological Model Representation

PL models need to be visualised in an interactive user-friendly format. Thus, there is a demand for a unified language for representing and exchanging PL models between the modelling tools and visualisation/analysis tools. Figure 2 illustrates the popular standards for pathway representation.

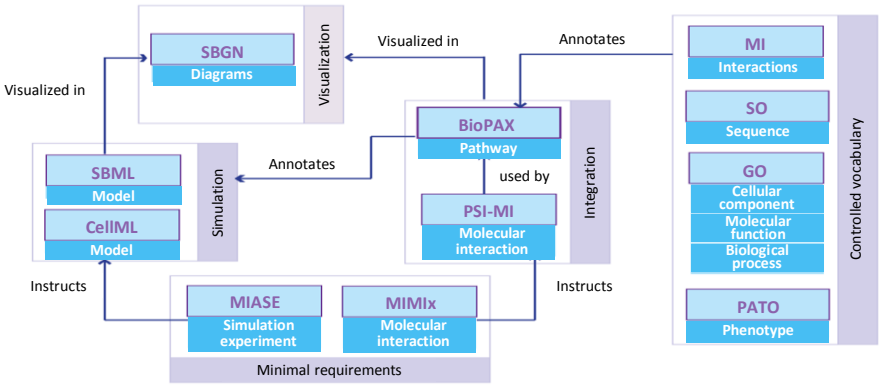


Figure 2 – Popular pathway standard formats and their relationships, from **[14]**

Systems Biology Markup Language (SBML) is a standardized exchange format for systems biology models. SBML is formally defined using UML, the Unified Modelling Language. Other modelling languages include BioPAX and CellML. ML-Rules is another multi-level rule-based language. These languages with many others are required to provide greater interaction between systems biology tools and address the incompatibilities between them.

Understanding modelling languages would assist in achieving a robust connectivity between the various tools by talking to each other a common language. Thus it is important to get an intuition about the various modelling languages.

2.2.1 SBML

Systems Biology Markup Language[1] is a modelling language that is formally defined by the Unified Modelling Language UML. UML definition is in turn used to define an eXtensible Markup Language XML representation for models to be seamlessly exchanged between analysis and visualisation tools taking advantage of the portability of XML. This language addresses interoperability problems discussed in [12]. SBML serves as a standard for model information exchange in computational biology and specifically in molecular pathways. Reaction networks like cell signalling, gene regulation, and metabolic pathways can be encoded in SBML to ensure compatibility with analysis and simulation tools. Thus, an SBML model generated from a legacy, not-supported tool can be reused effortlessly in other tools that support SBML.

SBML elements mainly include: Compartment (container for substances reactions), Species (reaction substance), Reaction (description of species transformation or binding), Parameter (symbolic name representing a quantity), and Rule (mathematical expression). [12] describes the various components of SBML and aids the discussion with a concrete example. Figure 3 shows an illustration of a biological model.

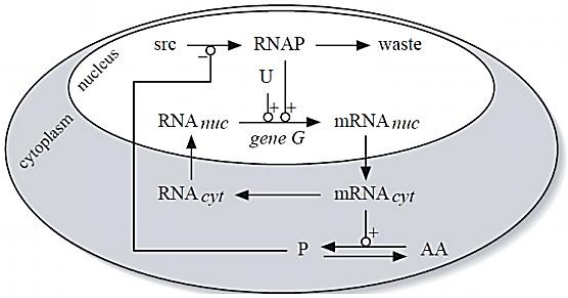


Figure 3 - A two-compartment model of a hypothetical single-gene oscillatory circuit in a eukaryotic cell from **[12]**

The corresponding SBML model looks something like following (code snippets collected from [12] and upgraded to SBML level 3):

[1] SBML specification is available from http://www.sbml.org

```
<?xml version="1.0" encoding="UTF-8"?>
<sbml
xmlns="http://www.sbml.org/sbml/level3/versi
on1/core" level="3" version="1">
  <model id="gene_network_model">

    <listOfUnitDefinitions>
     <unitDefinition id="mmls">
      <listOfUnits>
         <unit kind="mole" scale="-3"/>
         <unit kind="litre" exponent="-1"/>
         <unit kind="second" exponent="-1"/>
      </listOfUnits>
     </unitDefinition>
     ...
    </listOfUnitDefinitions>

    <listOfCompartments>
     <compartment id="Cyt" size="1.5" />
     <compartment id="Nuc" outside="Cyt" />
    </listOfCompartments>

    <listOfSpecies>
     <species id="mRNA nuc"
         compartment="Nuc"
         initialAmount="0.0032834" />
     <species id="RNA_nuc"
         compartment="Nuc"
         initialAmount="96.117" />
     <species id="src" compartment="Nuc"
         initialAmount="1"
         boundaryCondition="true" />
     <species id="P" compartment="Cyt"
         initialAmount="22.035" />
     ...
    </listOfSpecies>

    <listOfParameters>
      <parameter name="Vi" value="10" />
      <parameter name="Ki" value="0.6"/>
    ...
</listOfParameters>

    <listOfRules>
     <rateRule variable="src">
      <math xmlns="... " xmlns:sbml="...">
         <apply>
         <divide/>
           <ci> Vi </ci>
           <apply>
           <plus/>
             <cn sbml:units="conc"> 1 </cn>
             <apply>
             <divide/>
               <ci> P </ci>
               <ci> Ki </ci>
             </apply>
           </apply>
         </apply>
      </math>
     </rateRule>
    ...
    </listOfRules>

    <listOfReactions>
     <reaction id="R1" reversible="false">
      <listOfReactants>
         <species Reference species="src" />
      </listOfReactants>
      <listOfProducts>
         <species Reference species="RNAP"/>
      </listOfProducts>
      <kineticLaw formula="Vi/(1+P/Ki)" />
     </reaction>
    ...
    </listOfReactions>

 </model>

</sbml>
```

Assuming the rule representing the reaction rate equation:

$\boldsymbol{V_i}/(\mathbf{1} + \boldsymbol{P}/\boldsymbol{K_i})$ for the reaction $\boldsymbol{src} \rightarrow \boldsymbol{RNAP}$

Readers can refer to SBML language specification for details.

2.2.2 BioPAX

Biological PAthways eXchange[1] is a standard biological model exchange format. BioPAX aims at defining a unified framework for exchanging pathway information. It is defined as an ontology of concepts and attributes [13]. The main class used in BioPax is Entity. Entity has three subclasses: PhysicalEntity (interacting objects), Interaction, and Pathway (a model of interactions). PhysicalEntity itself has five subclasses: complex, small molecule, protein, DNA and RNA. Modifications performed on physical entities like activation, relocation, or phosphorylation are called features in BioPAX. This format requires tools that are capable of handling reasoning with inheritance of such class extendable hierarchy.

BioPAX covers all major pathway concepts familiar to biologists; like molecular interactions, gene regulatory networks, genetic interactions and metabolic and signalling pathways [14].

[1] Available from http://www.biopax.org

Figure 4 illustrates a high level view of the BioPAX ontology showing class hierarchy.

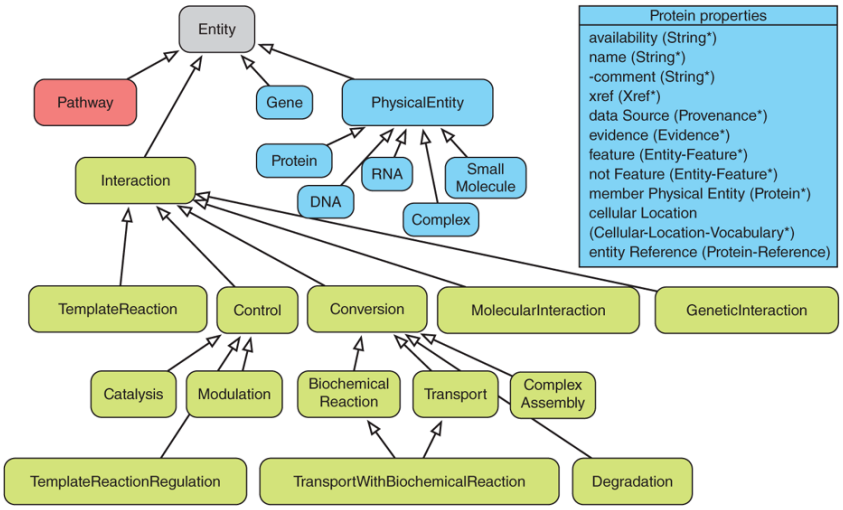


Figure 4 - BioPAX ontology, from **[14]**

In addition to the constructs illustrated in Figure 4, BioPAX is provide other constructs like cross-references (called xref classes) to biological databases[1]. It is also capable of reusing and linking to vocabularies in other ontologies, like PSI-MI which is discussed in the following section. Appendix I shows how graphical models are expressively represented in BioPAX language.

2.2.3 Other Systems Biology Modelling Languages

Finding a common format for describing and exchanging biological models has become a demand specially with the vast growth of formal tools as well as simulation and analysis tools. In addition to SBML, many other modelling languages have been rapidly developed in the field to address the requirement of compatibility. CellML [15] is yet another XML-based language used as a standard for facilitating quantitative representation and exchange of biological processes. Figure 5 illustrates an example of a CellML model showing the language elements and their relationships.

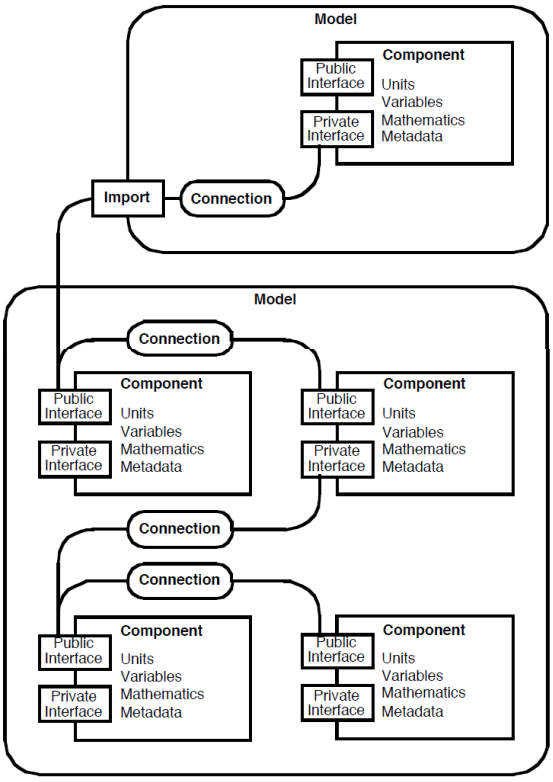


Figure 5 – an illustration of CellML model showing language elements, from **[15]**

[1] Such databases are discussed in Systems Biology data sources section),

CellML[1] is an open development project with on-going efforts to broaden its scope and provide tools for working with the language efficiently. These tools include the CellML Model Repository [16] which provides free access to over 520 biological models. Repository models are under on-going model curation process to ensure accurate reproduction of published results. The Physiome Model Repository 2 (PMR2) [17] is the foundation for the repository with the advantages of enhanced collaboration and thorough model change history. Additionally, a CellML API [18] is developed to allow model information retrieval and modification.

Other model description languages are available like PSI MI. [13] evaluates and compares PSI MI, BioPAX, and SBML and finds them similar in format structures. However, it shows that SBML is more suitable for modelling molecular pathways, while PSI MI is tuned towards interactions and experiments representation. The paper finds BioPAX format more expressive and more general.

"ML-Rules" [19] is another language used to represent molecules and their interactions. Remarkably, ML-Rules provides the facility to represent high level organisation like cells and organelles; i.e. it provides the ability to represent objects as a collection of other objects. Systems Biology Graphical Notation Markup Language SBGNML[2] and Biological Connection Markup Language BCML [20] are also other biological model description languages.

Despite all specific advantages that each language provide, they all share the aim to provide a standard format for describing biological models to address the incompatibility between available tools and seamlessly exchange information between them. A lot of efforts have been accomplished to make SBML a compatible and easy to use standard. These efforts include developing APIs and libraries like JSBML[3] [21]; and developing interfaces and toolboxes for other software such as MATLAB. Furthermore, lots of other formats like CellML are now finding ways to seamlessly convert to SBML format and CellML2SBML [22] is one of these suites. In 2007, an online survey was conducted by [23] and found that SBML is widely recognised as a standard format (60% of respondents were users of SBML).

Having realised the need for a standard exchange format, and the wide availability of languages providing such standard, it is important to understand how the standard format is exchanged between the tools. The standardised-format (understandable) models are required to be supplied to the visualisation tools in a compatible way to provide a complete interactive experience. There are tools like the InterOperability Platform IOP, JAMES II, and the Systems Biology Workbench SBW that features such functionality and offers a high level of interoperability and communication interfaces with formal tools like Maude. These platforms are explained in the following section.

[1] The CellML Project resources can be accessed from http://www.cellml.org
[2] Specification available from http://www.sbgn.org
[3] JSBML is a Java-based library for SBML available from http://sbml.org/Software/JSBML

2.3 Systems Biology Tools Interoperability

Many Systems biology tools have emerged providing a variety of features and functionalities to work with biological models. Among all, it is hard to find one application that combines all tools a researcher would require. The key here is to provide means for seamless integration between the entailed tools to provide a rich experience for biologists and curators with the ability to connect to further tools whenever needed. In the last few years, a number of platforms facilitating such integration have become apparent. In addition to allowing tools to communicate, some of these platforms; like the Systems Biology Workbench SBW; come with their own set of integrated tools. Others; like the InterOperability Platform IOP; are mainly dedicated for providing the interoperation between different tools. JAMES II is another kind of such platforms with the feature of being a development framework in its own. Following sections provide a brief overview about each of these platforms with a more focus on IOP for its motivation of communicating Maude to other tools within PLA.

2.3.1 SBW[1]

The Systems Biology Workbench [24] is a modular framework that connects modelling and analysis tools based on broker architecture to reinforce reusability of capabilities between tools. It supports SBML as a native format. SBW employs a message-passing mechanism to implement the connectivity using a binary formatted messages for better performance.

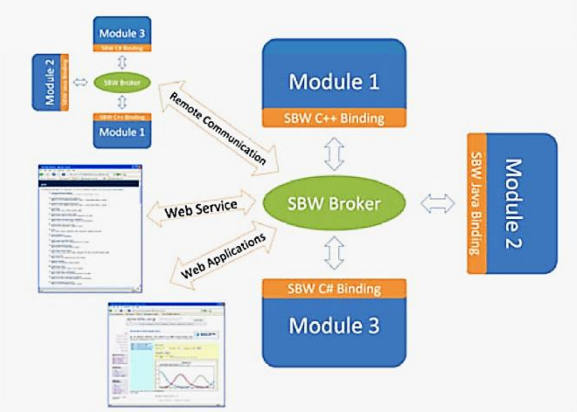


Figure 6 - SBW communication flow, from **[24]**

SBW is a language-neutral architecture providing binding libraries for a variety of common languages (to become SBW-enabled) like C, C++, C# (.NET languages), Java, Matlab, Perl, Delphi, and Python. In SBW, an SBW-enabled application is a module. Figure 6 illustrates the SBW communication flow. SBW comes with its own modules like: utility modules which provide the core features for SBML analysis; simulator; and analysis tools. Today, there are many SBW-enabled tools in the fields of modelling, analysis, simulation and visualisation of biological molecules and processes.

2.3.2 JAMES II[2]

JAva-based Multipurpose Environment for Simulation II (formerly known as Java-based Agent Modeling Environment for Simulation) is an open framework for

[1] SBW is an open source. Free downloads and resources are available from the project website http://www.sys-bio.org

[2] JAMES II is an open framework available from http://www.jamesii.org

developing and integrating the diversity of modelling and simulation methodologies in a flexible plugin-based development environment.

JAMES II aims at providing flexibility, reusability and extendibility through offering over 500 plugins for modelling, analysis and simulation. It supports any modelling formalism and is not restricted to specific formalism. JAMES II integrates SBML models and alternatively has its own custom language to formulate models. It also offers specialised simulators with different levels of abstraction support [25]. Additionally, JAMES II enables the configuration and exploitation of dry-lab (simulated) experimentation by featuring repeatability and reuse [26]. ML-Rules are realised within JAMES II with a set of plugins to work on ML-Rules models [19].

2.3.3 IOP[1]

Formal tools – like Maude – need to interact with other tools to make benefit of its features with a user friendly means of interaction. Maude current interaction method is via a command line interpreter. The InterOperability Platform IOP has come to furnish interoperation; specifically between Maude and other tools, and generally between other tools. IOP enables Maude to communicate with web resources, theorem prover applications like PVS [27], visualisation tools and even communicate with another instance of Maude.

Similar to SBW, IOP adopts a message passing interaction paradigm with a registry (corresponds to Broker in SBW) that acts as a communication centre for routing messages to the intended tools. Each tool is represented by an actor (module bindings in SBW). IOP comes with some built-in actors like Maude actor (interactive extension of Maude, also called IMaude[2]); Graphics actor (used for visualisations and interactive graphs); PVS actor, and other actors for executing programs, accessing file system, and supporting communication sockets. Interestingly, further actors for communicating to other tools can be added easily to the platform. Yet this is not required in case if the tool is connected to the internet via a connection socket as communication actors can be utilised to interact with such tools.

IOP and IMaude are the basis for the Pathway Logic Workbench PLW; a substantial pathway logic application that provides an integrated environment for working on biological models and cellular networks. Figure 7 illustrates the PLW architecture and the role of IOP in communicating between the different tools.

PLW integrates fundamental tools like Maude and Pathway Logic Assistant (an IMaude program), with other supporting tools like BioNet Petri net tool and the Dot graph tool (explained in a later section). [28] sketched two more applications which also have driven IOP development: Mobile Maude and SCRover. Figure 8 shows actor communication in IOP.

[1] IOP binaries and resources are available from http://jlambda.com/~iop
[2] IMaude code available from http://www.csl.sri.com/users/clt/IMaudeWeb

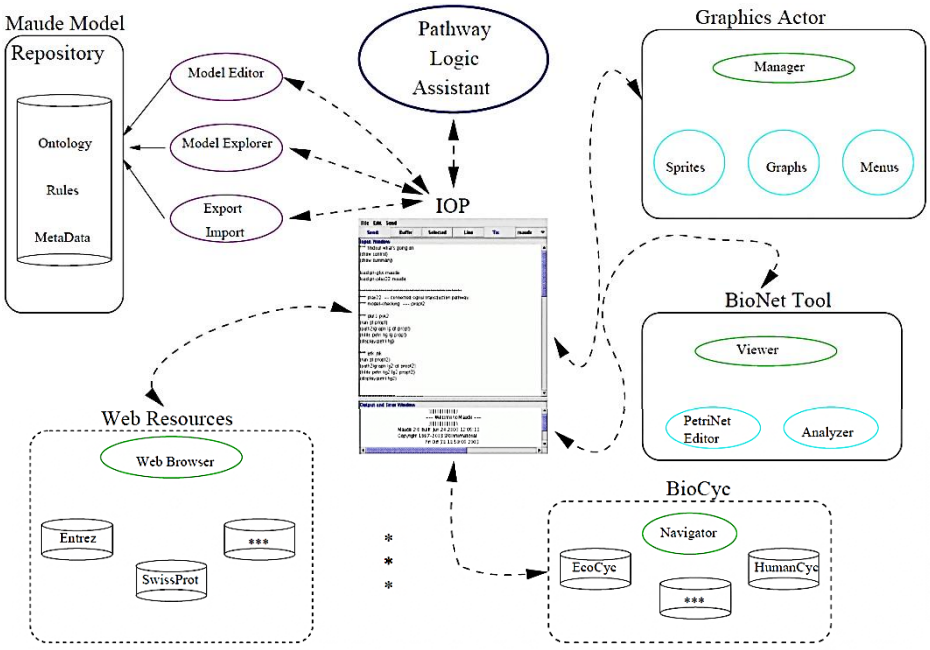


Figure 7 – IOP role in Pathway Logic Workbench Architecture, from **[28]**

In IOP, actors are either a single UNIX-style process[1] or dual processes. For example, Maude and PVS actors consists of two processes: one for running the tool and the other for facilitating the communication with the message centre; i.e. the registry. There are three other independent non-actor processes interacting in IOP: first for system creation and configuration (called main), second is the registry, and the third is the GUI which is written in Java's Swing platform.

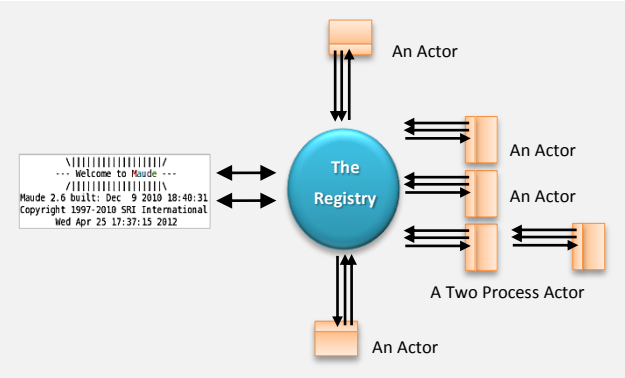


Figure 8 - IOP Interaction Architecture, reproduced from **[28]**

IOP registry maintains three forms of communication in the platform; inter-actor, meta-actor, and interface communication [28]. Following is an example, from [28], of a messages sent from the Graphics actor to the Maude actor:

```
maude
graphics
show mauderule 23
```

Being as such communication infrastructure with all the provisions of interoperation and robust interaction, IOP has become the underlying infrastructure for the Pathway Logic project with future plans to further extend PLW to connect to other platforms like SBW.

[1] A process is a running program instance

2.4 Model Visualisation Tools

PL models represent networks of biological entities and complex reactions in living cells. This huge and diverse amount of data need to be analysed in an intuitive visualised way to enable biologists to make inferences about the underlying biological processes. The new system needs to utilize one of the available pathway visualization tools if it is feasible. An alternative would be to develop a totally new visualisation and model editing features tailored for PLA.

[29] examined a variety of pathway visualisation systems and analysed the end-user perceptions of such systems as evaluated by domain experts. It proposed an agenda for visualisation systems requirements: (i) systems need to implement automatic pathway construction and update based on literature databases, (ii) standard representation of pathway information (information overlay), (iii) capability of linking data from high-throughput experiments, (ix) ability to overview and interconnect simultaneous pathways, and (x) scaling pathways to higher abstraction levels. In addition to these requirements, I can say that a robust pathway visualisation system needs also to enable biologists to make inferences about biological processes by integrating necessary logic formalisms.

There are a variety of tools in the literature and each has its own strengths and weaknesses. We are going to discuss some of the popular tools in the field like GraphViz, CellDesigner, CytoScape, and few others.

Graph Visualization Software GraphViz[1] is an open-source package of tools for drawing graphs written in dot language. Dot language is a text-based specification language used for describing hierarchical directed graphs. Dot graphs are utilised for visualising protein interaction pathways and applied in many applications including PLA. Dot graphs beauty lies in exposing hierarchical structure in a balanced and symmetrical favour keeping edges between nodes as short as possible while avoiding crossings and sharp bends [30].

CellDesigner[2] is a free biochemical networks modelling and simulation tool. It uses systems biology graphical notation SBGN and standard markup language SBML for describing models. CellDesigner provides references to literature databases and it is Java-based, which makes it platform independent. Moreover, CellDesigner is able to integrate any SBW-enabled module [31] which makes it a flexible tool for connecting and integrating with other tools.

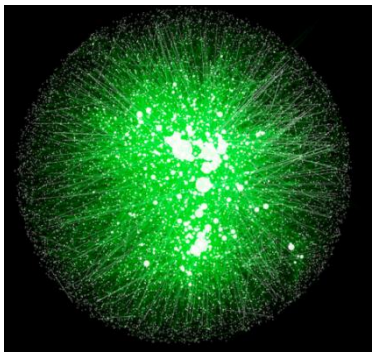


Figure 9 – Human Interactome visualized in Cytoscape, from http://www.cytoscape.org

Cytoscape[3] is another open-source pathway visualisation platform that is capable of visualising and integrating complex networks. Cytoscape provides the provision of representing biological properties of pathway elements via manipulating graphical nodes properties like color, size, and

[1] Available from http://www.graphviz.org
[2] Available from http://celldesigner.org
[3] Available from http://www.cytoscape.org

shape [29]. This tool can handle large scale networks such as human interactome (interactions of genes) or social network datasets. Figure 9 illustrates a human interactome network visualised in Cytoscape. Some scientists finds Cytoscape models complex to understand according to [29].

Ondex[1], the data integration and visualisation application is yet another tool used in analysing experiment data in relation to a variety of databases and formats. Ondex is used in systems biology as well as other applications that require graphical representations and data analysis. Text and data mining are also one of the provided functionalities for its power in information retrieval and integrating data from diverse sources.

Other visualisation tools include PathVisio[2] which is a Java-based implementation that mimics GenMAPP tool [32]. PathCase [33] is another tool used for store, query, visualise and analyse metabolic pathways at different levels of abstraction ranging from gene level to organism level.

Apart from all the available spectrum of pathway visualisation tools, and in addition to all the strengths these tools provide, a good pathway visualisation tool needs – at the first place – to satisfy biologists' requirements in a simple and intuitive way to be able to integrate their mental models as well as analysing experiments data effectively. For this project, visualisation tools are explored for integration feasibility to achieve the intended objectives of the new rule graphical editor.

2.5 Systems Biology Data Sources

Proteins and other molecules normally have naming variations depending on biologists or curators naming style. To prevent ambiguity, these names need to be referred to unique entities. Thus many standard databases have emerged to maintain standard knowledge bases to provide appropriate conventions of these concepts and terminologies with detailed descriptions.

In PLA, all protein names have links to entries in the UniProt[3] database; a biological database of manually curated protein sequences. PLA links other non-protein molecules – like lipids and sugars – to entries in the Kyoto Encyclopedia of Genes and Genomes KEGG[4]; a database resource for biological systems. Additionally, BioCyc; a collection of metabolic pathway databases, is exploited in the PL Workbench. Another extensive database source is PANTHER[5] (Protein ANalysis THrough Evolutionary Relationships) system for gene classification based on experimental evidence publications. Other available sources include – but not limited to – Reactome, PubChem, HUGO, and BioCarta (an interactive source for pathway and cellular information). [34] describes MetaBase, another resource database that serves as a hub for the most commonly used biological databases.

[1] A free open source application available from http://www.ondex.org
[2] Available from http://www.pathvisio.org
[3] Available from http://www.uniprot.org
[4] Available from http://www.genome.jp/kegg
[5] Browsable database from http://www.pantherdb.org

Linking to any of these curated databases becomes a demand for sharing a common understanding and to prevent any ambiguity that might arise while developing the new tool.

2.6 Pathway Logic Assistant

Understanding all previous concepts builds the ground for having a more clear intuition about the current system; PLA. PLA is a Java based tool that provides an interactive graphical interface to PL models [4]. It provides a means of performing logical computations on PL models (also called PLMaude models) in response to user requests.

In PLA, models are visualised as Petri nets with nodes representing components (occurrences) or rules, and edges connect reactant occurrences to rules and rules to reaction product occurrences (see Figure 10). In a PL knowledge base, rules are stored in Maude syntax.

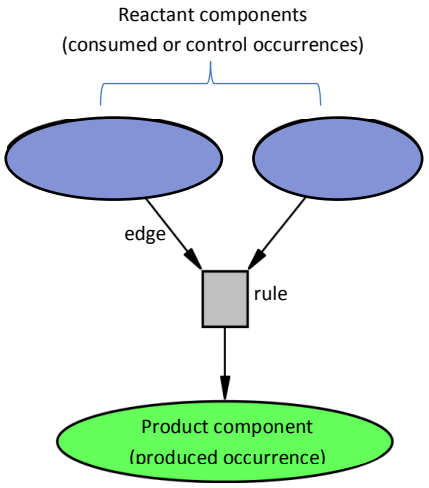


Figure 10 – Petri net representation of rules

Each PLA occurrence (analogous to physical entities in BioPAX); is constituted from an entity, modifications, and location. Table 1 outlines these three components with possible values of each.

Table 1 – PLA occurrence components with some possible values.
(Values are obtained from http://pl.csl.sri.com)

Component	Possible values
Entity	protein, small molecule, RNA, or DNA
Modifications	`Yphos` – phosphorylated on a tyrosine `act` – activated `reloc` – relocated `ubiq` – ubiquitinated `GDP` – loaded with Guanosine diphosphate (GDP) `GTP` – loaded with Guanosine triphosphate (GTP)
Location	`Out` – outside the cell, the medium or supernatant `CLm` – in/across the cell membrane `CLi` – attached to the inside of the cell membrane `CLc` – in the cytoplasm

PLA provides biologists with the ability to study and explore available dishes (initial system states) as well as creating or modifying existing dishes. Biologists can display signalling reactions networks of specific model. They can find and compare pathways. Additionally, Gene expression data can be visualised with color-coded scheme [4].

In addition to the above, PLA enables users to access PL Knowledge Base interactively by means of Petri net interaction. With Petri nets, PLA visualises reactants and products, as well as reaction controls (or modifiers) which are

compulsory for the reaction to take place but are unchanged after the reaction. Controls can be enzymes, catlysts, scaffolds, or any other controlling substance. Thus each rule (reaction) in PLA has participant occurrences of one of the roles: consumed (reactants), produced (products), or controls. Controls are connected to rule nodes with dash arrows.

Petri net rules that correspond to Knowledge Base rules are called Transition Knowledge Base TKB. Any rule is enabled if all its connected reactants and controls are present in the current system state. When an enabled rule is executed, the reactants are removed and products are added to the state keeping the controls unchanged.

In PLA, each PLMaude model maps to a Petri net model; [35] defines this model as following:

> "A Petri net model is a pair (T, I) consisting of a set of transitions T, and an initial state I (a set of occurrences)."

Where transition is a rule long with its connected occurrences. The petri net model can be executed then by firing 'rewrite' rules that becomes enabled upon user selection (by putting tokens on occurrences to indicate their presence in the current state). Users then can perform analysis tasks like setting some 'goal' or 'avoid' occurrences to find a *subnet* from the full network that leads to this goal and avoiding the avoids. A Petri subnet is defined as following:

> "A Petri subnet is a tuple (T, I, G, A) consisting of a set of transitions, T, an initial marking, I a goal marking G, and an avoids set A." [35]

Subnet results can also be further analysed. PLA also finds knockouts which are "omissions from the initial state that prevent reaching a goal" [4]. Figure 11 illustrates the process sequence and some of PLA model analysis features.

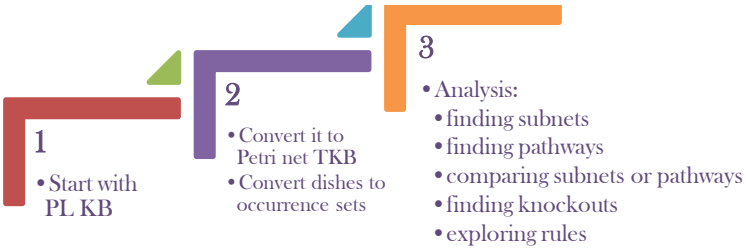


Figure 11 - PLA process and analysis features overview

2.6.1 PLA GUI front end

As mentioned, PLA is a Java based application (Figure 12 shows a screen shot from PLA). In fact, GUI creation in PLA is driven by JLambda[1]: a Java-based language for interactive visualisation with syntax close to Scheme language [36].

> "**JLambda** is an untyped Scheme-like lexically scoped interpreted language, that provides a runtime interface to available Java classes, using Java's built-in reflective capabilities." [37]

[1] Available from http://www.jlambda.com

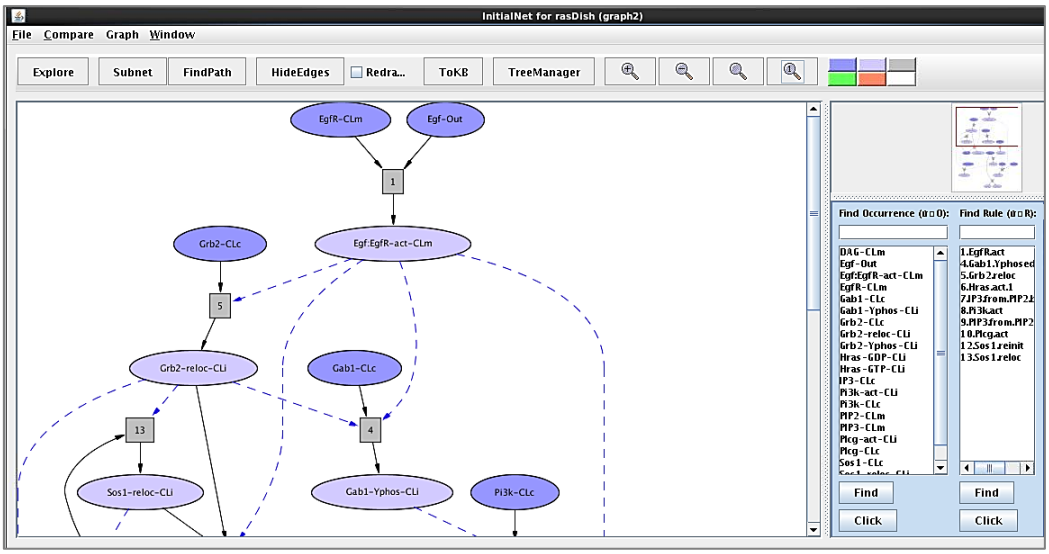


Figure 12 - Screen shot of the PLA. It provides features like zooming, searching occurrences and rules, exploring, finding pathways, and other features.

JLambda is a core component (actor) in IOP (the interoperability platform implemented in PLA). JLambda comes with a parser, an interpreter, and a java class hierarchy called Glaphish hierarchy used for run time construction of interactive graphical objects [38].

PLA supports exporting models to a number of formats like JLambda, Petri net, and SBML. Work on integrating representations through BioPax language is in progress. Appendix II shows an example of an SBML format exported from PLA.

Figure 13 illustrates PLA architecture showing technologies integrated in this architecture.

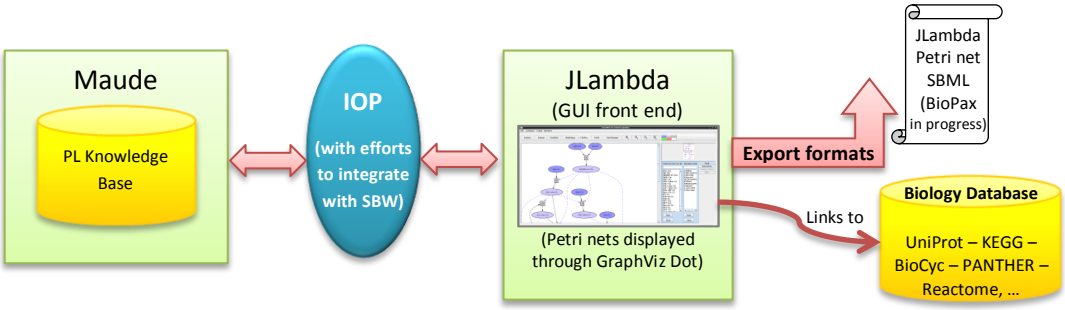


Figure 13 - PLA architecture

PLA tool and all related technologies included in its architecture provide a framework for biologists to create and analyse biological models. Improving such framework by adding new facilities, like a graphical rule editor, would require the developers to understand the current framework very well. This will make the integration much easier and the implementation a success. PLA is explored for improvement opportunities in order to achieve higher user satisfaction level and hence better throughputs. Motivation section spots the light on these prospects and is driven by examples to better understand the requirements.

3 Motivation

This project aims at providing a more user-friendly interface for the symbolic systems biology tool PLA. The aim of the work is to develop a graphical rule editor for PL models which would make the PL approach more accessible to users with no background in formal logic. Such graphical tool would simplify and accelerate the construction as well as modification of large models. This chapter focuses on In this section, I motivate my approach using a small example and, in a following section, I describe the methodology for achieving this goal.

3.1 A spotlight on the current representation

3.1.1 Taking PLA guided tours

SRI International provides a number of online guided tours in some PL models for familiarising users with PLA. STM6[1] is one model used to study the changes to cell proteins as a result of stimulating the cell by binding some ligands to some protein receptors on the surface. To motivate the methods in this project, and after taking the STM6 guided tour, we select one of the many rules (reactions) in this model. This rule is then studied to better understand its PLA representation.

3.1.2 The Rule - Reaction Background

Living cells encompass protein molecules on their surface to receive extracellular signal or stimulations that would change the state of the cell. These molecules are called Receptors, and EgfR is one of these receptors that receive signals from Egf (Epidermal Growth Factor) protein family. When an Egf protein is located outside the cell, and if the cell membrane contains an Egf receptor (EgfR), Egf gets bound to EgfR and EgfR gets activated.

Many cancer drugs target EgfR, as mutations involving this receptor may lead to uncontrolled divisions of the cell, which is an inclination for cancer [39]. Figure 14 illustrates EgfR in complex with Egf.

Figure 14 - Egf receptor in complex with Egf outside the cell, from [46]

3.1.3 Maude and Petri net representations

To represent the above rule in Maude, required 'sorts' need to be defined to be used in the rule code. The rule is coded as following:

[1] This tour is available from http://pl.csl.sri.com/stm6-guide.html

```
vars out clm : Soup .

rl[001.EgfR.irt.Egf]:
{ Out |  out Egf }
{ CLm |  clm EgfR }
  =>
{ Out |  out }
{ CLm |  clm  Egf : [EgfR -  act] } .
```

PLA visualises rules as Petri nets as we realised. The Egf rule is represented in Petri net as in figure 14.

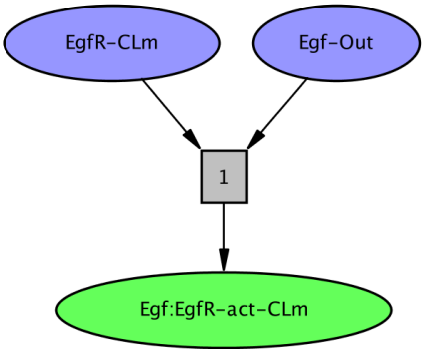


Figure 15 - Petri net transition of rule **001.EgfR.irt.Egf** obtained from PLA software

3.2 The weak link

Unfortunately, once drawn as Petri net, the rule cannot be modified unless the Maude code is modified and the dot graph is re-drawn again. Although PLA assists biologists to explore and analyse biological models, yet it lacks the power of editing its knowledge bases graphically. SRI International, the creators of PLA, are working on improving PLA by seeking integration opportunities with other platforms and standards like SBW and BioPax. Nonetheless, PLA remains a very useful tool for biologists and curators. I will not emphasis more on what functionalities PLA provides, as the previous chapter elaborated all these aspects. Rather, the focus here will be on the situations where the biologists find PLA complicated or not able to meet their requirements.

3.2.1 Motivating use cases

In a series of private communications with SRI International, we agreed to have biologist curators to perform some reality checks on PLA to find out what might prevent PLA from meeting their requirements.

Table 2 outlines three scenarios of biologists working in the context of a PL network in PLA GUI editor. This network could be specific to a dish, a subnet resulted from finding pathways to goal(s), or even it could be a Knowledge Base.

Table 2 - PLA use scenarios, from SRI International

Scenario 1	Scenario 2	Scenario 3
A biologist studying a particular pathway segment, is building a little network from scratch, to represent his/her interpretation of experiments.	A curator adding new information to an existing network, either updating an existing rule, or adding new rules to expand coverage of the network.	The biologist of scenario 1 decides to integrate his little network into a larger network (in PLA or external database).

The biologist in scenario 3 is required to be unambiguous about protein names, terms used for location, and modification. Currently, biologists have to resolve all

names by hand. PLA needs to provide an automated mechanism to ensure unambiguity without biologist's intervention.

3.3 Initiatives for improvement

3.3.1 Biologists- friendly Representation

SRI International has provided a graphical idea of how a rule would look like in a more biologist friendly form based on encoding occurrence locations inside the cell visually. Figure 15 illustrates how a petri net representation would be represented in this biologist-friendly form. The friendly form is drawn by Merrill Knapp from SRI International (private communication).

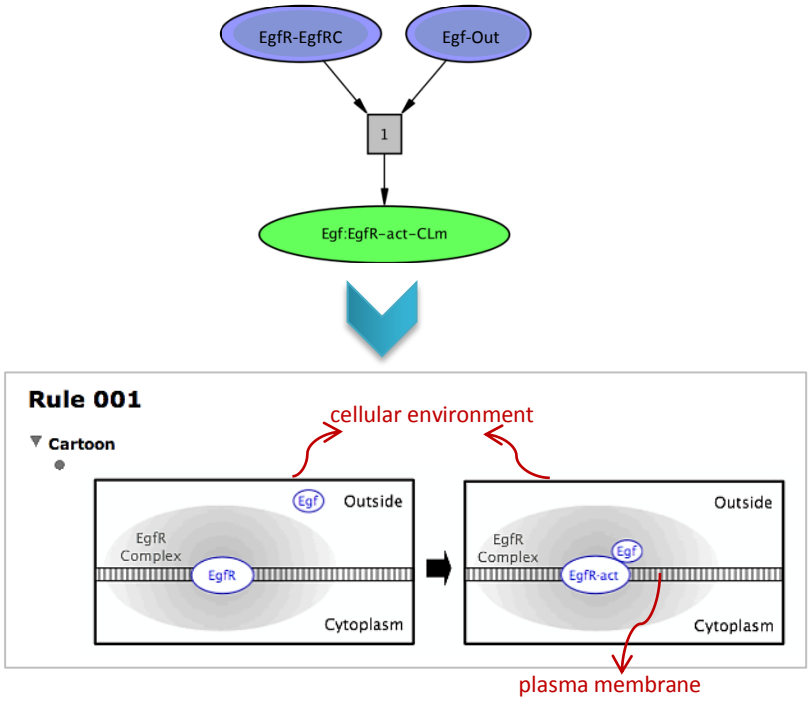


Figure 16 - Rule representation in petri net (drawn by dot) and biologist-friendly form (from SRI International)

Rule 001 shows Egf (consumed occurrence) moving from the outside to the EgfR Complex (consumed occurrence) resulting in binding Egf to the Egf receptor, EgfR and activating EgfR. Another rule in the friendly representation can be found in appendix III.

Therefore, to achieve the discussed specifications, the new system would be developed as an integrated solution that is capable of connecting a suitable graphical representation to the conceptual layer (PLMaude) via the interoperation layer.

3.3.2 Rule Paraphrase

In another attempt to make PLA more user friendly in representing rules, SRI has proposed a textual representation of rules as in figure 16:

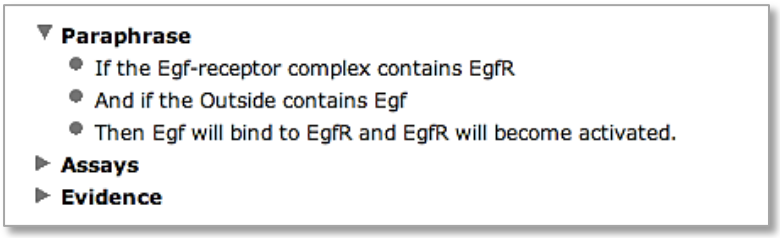


Figure 17 - Paraphrase of rule 001 in figure 15

3.4 The essence

It can be concluded from these scenarios and initiatives that the main requirement is – in fact – the ability to create new rules or update existing ones in a user friendly way. Creating or updating rules will require manipulating occurrences. PLA needs to provide the relevant features in its GUI for editing rules as well as the internal knowledge base.

In addition to these requirements, there is a need for custom software to visually project back the complex reasoning outputs generated by Pathway Logic onto the graphical model.

In response to these essential demands, this project is proposed to overcome the difficulties biologist curators face with the current PLA system. Additionally, it is necessary to address the requirements for extra features that would reduce the overhead and hence save much of scientists' time. The visualisation tools discussed in the background chapter would not be applicable for such specific requirements and would require complex integration interfaces. The ultimate decision is to develop a graphical tool that closely meet all these demands. Following chapter discusses and plans for building a graphical rule editor to address the issues with the current PLA system.

4 Designing a Graphical Rule Editor

It is critical to plan for any software development carefully in order to achieve success. Those applications that involve improving or extending an existing system would require a thorough study of the current process to find out how it can be improved. This chapter reviews the current process for adding/editing rules to PLA KB, and moves forward to planning for developing a new graphical editor with all design aspects to achieve successful implementation that would meet the pre-defined objectives.

4.1 Charting the Process

Adding or editing rules in PLA knowledge bases would require Maude programming background and would involve working on code level. Current PLA users (either with or without Maude programming skills) would require to create rules to model their experiments data or to map curated models from the literature. Figure 18 sketches the process a biologist would follow to model his knowledge in PLA.

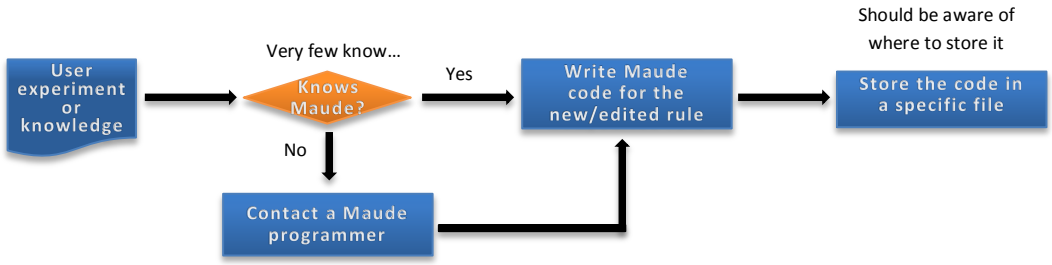
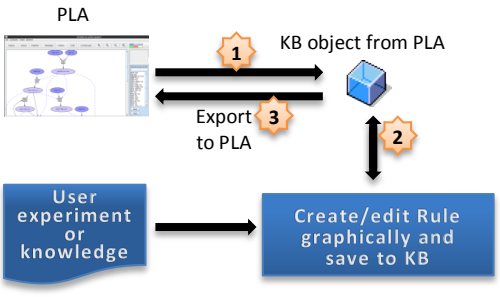


Figure 18 - Current rule creation and editing process

In all cases, Maude code needs to be added or modified in a specific file stored in a specific location in order to maintain any PLA KB.

The new tool is planned on the following improved process:

From user perspective:	Technical perspective:
	

In the new process, users are not required to have any programming skills for modelling PLA networks. They would be able to maintain their rules through a graphical interface supported by all features required for adding or editing rules.

From technical perspective, the new tool will be linked to PLA through maintaining an object that contains all information of a KB.

4.2 Layered-Object-Oriented Architectural Style

Architectural style (or pattern) is a set of principles combining a group of systems in one abstract framework[1]. Layered Architectural Style is the concept of grouping related functionalities in an application into vertically stacked layers. Realising the diverse technologies integrated in PLA, which has inspired the idea of visualising PLA – with all its auxiliaries – as layers. The main rule I devised and followed with this style is defined as follows:

> With a system S consisting of interacting layers L, a layer $l_i \in L$ needs only to interact directly to its adjacent layers l_{i-1} and l_{i+1}.
> I call this the *Adjacency Interaction Rule* (AIR).

Thus, any layer needs not to bypass an adjacent layer to interact to non-adjacent layer. Building on AIR rule, the system consists of three logical layers as illustrated in Figure 19.

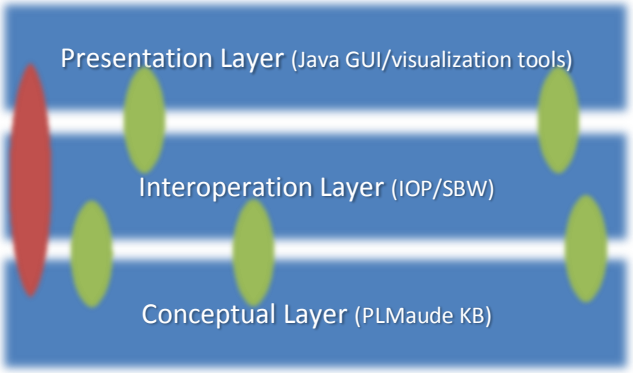


Figure 19 - Proposed system logical architecture

Any work involved would be in one of these layers or interfaces between them. According to AIR, green ovals represent valid interactions, while the red oval is an invalid interaction. Practically, an example would be that any attempt to modify or update the KB should not be done directly to the conceptual layer (as the case currently), instead, the system should provide the necessary operations in the presentation layer, presentation/interoperation interaction, and then interoperation layer will take care of the interaction with the conceptual layer.

Getting closer inside each layer, another design paradigm is adopted which is the Object-Oriented Architectural Style. This style is selected to have more flexibility while dealing with system components as objects. With this style, we can exploit all object-oriented programming benefits like Abstraction, Encapsulation, Polymorphism, and Inheritance to achieve a reusable and highly cohesive architecture. Combining these two styles into one application would acquire the advantages of both architectural styles.

[1] Chapter 3: Architectural Patterns and Styles, Microsoft Application Architecture Guide, 2nd Edition, 2009, MSDN Library on http://msdn.microsoft.com/en-us/library

Thus a properly designed user interface should consider all the modules in this architecture. The aim is to improve PLA in each of these layers whenever feasible, and/or develop new software modules that can be plugged or integrated seamlessly into PLA.

4.3 Design Specification

In PLA, the process of creating or editing rules needs to be compatible with internal representation of PLA. Each rule should have a unique name. Recall that a rule has participants: consumed and produced occurrence, and controls. Each occurrence in turn is an entity, location, and modifications.

It is very important to sketch the data structures considerations adopted in PLA system to maintain consistency. The editor will communicate with data structures specs (short for specifications). The intent is that a spec has all the information needed by the editor to provide choices and information to the user. Figure 20 summarises data structures' specifications to be considered while developing the new editor.

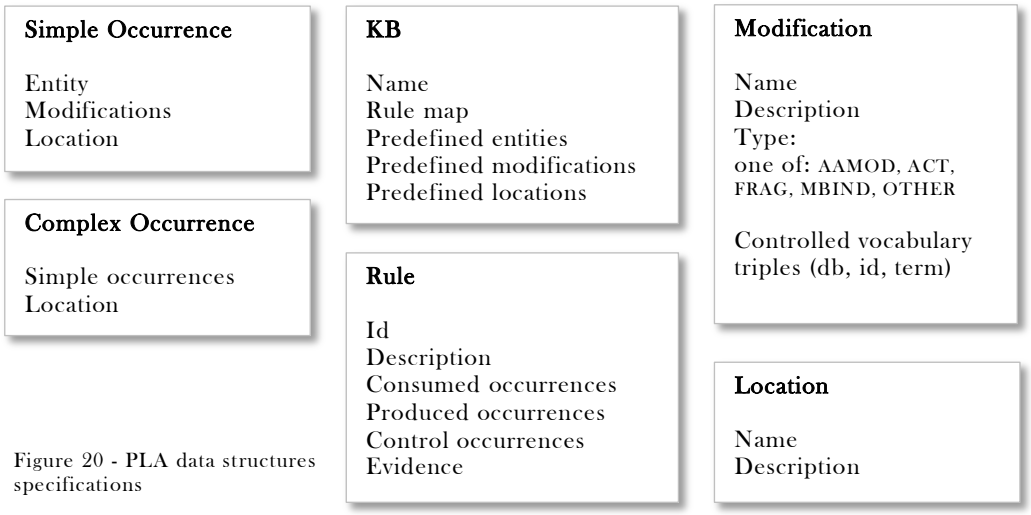


Figure 20 - PLA data structures specifications

Additionally, table 3 outlines some design specifications thought of in coordination with SRI team.

Table 3 - PLA rule design specification, as agreed with PLA creators.

Rule data	Specification
Name	Mandatory field specified by user Unique in current context Rule can have both short and long names
description	A descriptive text string
Evidence	- Can be specified by a pubmed id (a code for life sciences and biomedical articles) - Can have fields to specify article table or figure numbers - Text string for description and remarks

Rule Participants data	Specification
Occurrences & roles	User should specify occurrences of each role: consumed, produced, and controls. For occurrences: - User can select known occurrences from a list - User can edit a selected occurrence - User can create new occurrences
Occurrence data	**Specification**
Occurrence name	Short and long name. long name is computed so that occurrence with same parts has same long names Initially, an occurrence has an entity, location, and empty set of modifications (if not specified at creation).
Entity	- User can create entities - Has unique name - Has link to external standard database entry (like UniProt) - Link is optional: - No link required for families, composites, and complexes - Other database links or synonyms can be specified by user - User can select known entities from a list
Location	- User can create locations - Has unique name - Known locations can be selected from a list
Modifications	- User can create modifications - Has unique name - Known modifications can be selected from a list

4.4 Designing the GUI

Having it as its visual representation of models, Petri net representation is one possible candidate that might be used as the graphical notation in the new graphical tool for adding and editing rules. Although would have been consistent with the current PLA visualisation method, it would replicate the difficulties that biologists face in understanding the current representation. Accordingly, providing the new facility in a new innovative visualisation method would improve PLA user experience. Adopting a graphical representation that biologists would prefer to use and that they were involved in its innovation would be the best choice.

Referring to the biologist's friendly form discussed in the previous chapter, adapting such form would satisfy user expectations. Figure 21 illustrates a GUI design proposal for the new tool which was designed to emphasise the notion of visualising locations graphically. This design has been accepted by PLA creators and efforts were paid to come up with the final acceptable interface through continuous feedback adoption. The figure shows an intermediate stage in building one rule; i.e. 529.

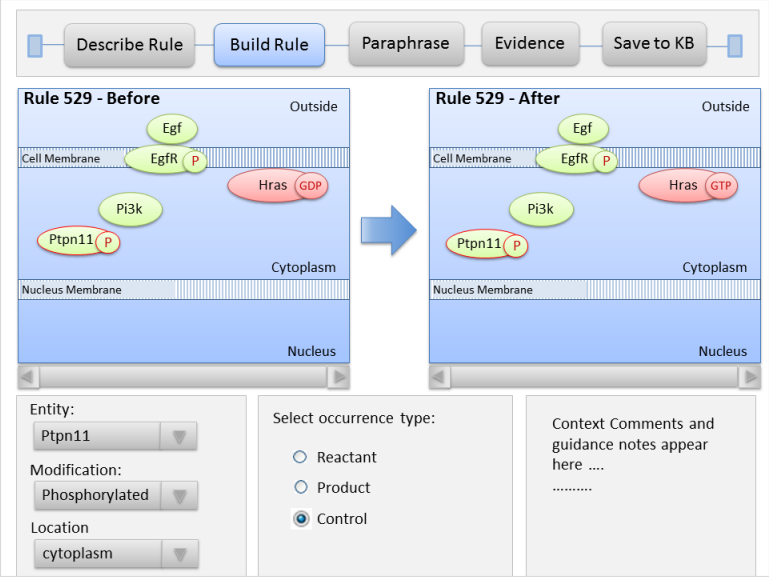


Figure 21 – A proposed design for the rule editor

The GUI is based on a guided procedure approach to make the whole process more intuitive for biologists. This interface guides the user through the process as following:

1. Describe Rule: to maintain basic rule information like name and description.

2. Build Rule: to assemble rules graphically through adding occurrences into two panels representing cell states before and after the reaction. An occurrence would be added by selecting an entity, modifications, and location form predefined lists. Occurrences are highlighted in different colours to recognise their roles; i.e. consumed, produce, or controls. Consumed occurrences are added to Before cell panel, produced to After, and controls to both panels as they are not changed by the reaction. The added occurrence can be recognised by a red border. Functionalities like dragging occurrences can be provided for more flexibility and more enhanced graphic.

3. Paraphrase: to see an automatically generated textual paraphrase of the rule.

4. Evidenc: to specify links to literature evidences.

5. Save to KB: to add the created rule to the PL knowledge base.

4.5 UML class diagrams

Having a design proposal would enable the decision of Java classes required for building the tool. The aim is to achieve a proper design that would maintain the main object-oriented programming principles. A high-level UML class diagram is used to plan the main data structures and classes that would be required for building the editor – see figure 22.

The diagram might have to be changed according to the programming decisions taken while implementing the system.

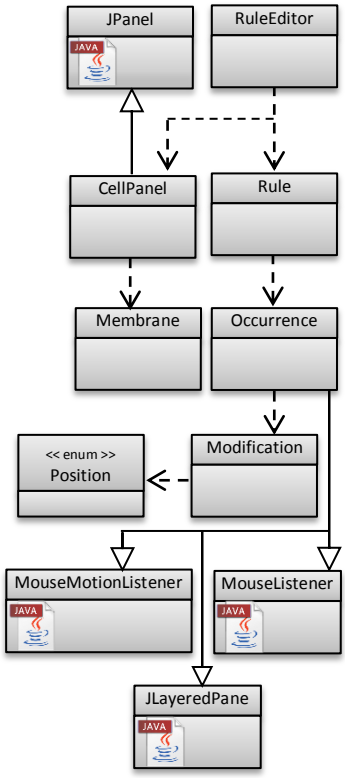


Figure 22 - High level UML class diagram

4.6 Planning the development environment

PLA uses a diversity of technologies for building, analysing, and visualising biological networks. To develop a new tool that can integrate and communicate to PLA, the same set of tools used need to be setup to simulate the real environment. Therefore all software instructed to be installed and any configuration required to run these software need to be applied in the development environment. Documentation and instructions on downloading these tools are all available from Pathway Logic official web site http://pl.csl.sri.com/software.html.

As it is important to setup a development environment very close to the real one, selecting the right development technologies is also important for the success of this project.

PLA is developed in Java and JLambda invokes Java methods seamlessly, thus selecting Java as the development programming language is the right decision. This decision supports the decision to have an object-oriented architectural style. Eclipse would be a suitable development suite for Java programming.

4.7 Planning the integration with PLA

As it would be part of the PLA tool, integration should be planned carefully to achieve success. All integration will be through a KB which would be encapsulated in its API (see figure 23).

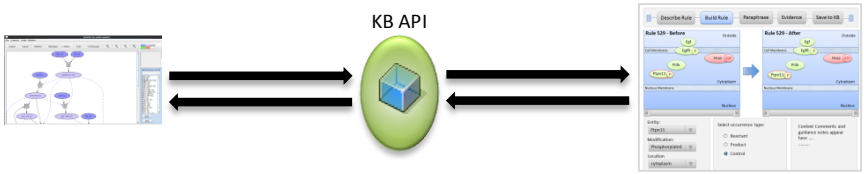


Figure 23 - Integration with PLA

The plan is to integrate with PLA on three stages:

Stage 1: Creating rules in a blank KB

Editing starts by passing a blank KB to the editor. Rules occurrences are created from predefined vocabularies for entities, modifications, and locations based on

the PL KB. Rules are then added to the KB through a PLA java class that provides methods to query the KB. There will also be a function to export a KB as a Maude model. Maude model can be loaded in PLA for analysis.

Stage 2: Creating and editing rules in an existing KB

When editing is done, the KB is exported as a Maude model and loaded into PLA as above.

Stage 3: Creating and editing rules in a KB loaded in PLA

This is considered to be online editing while the network is open in PLA, where changes can be done and committed as Maude model which should update the KB on the fly. At this stage we have a fully integrated editor.

One technical prerequisite to make the integration with PLA possible is to have a certain folder structure in the context of PLA various tools. Figure 24 illustrates the required folder structure and where the new editor executable file (Jar file) should be located; i.e. ruleEd.jar.

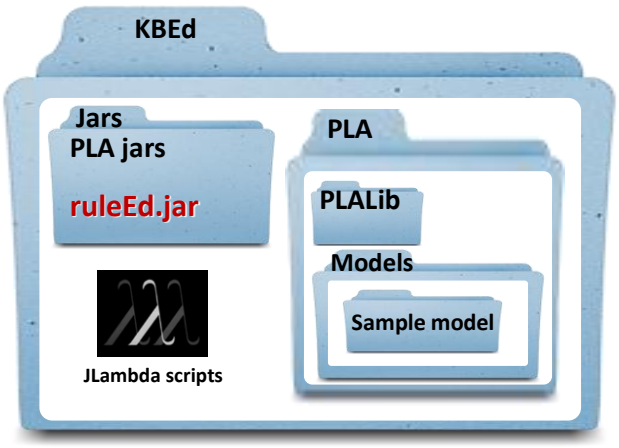


Figure 24 - Planned folder structure

5 Implementing the Editor

5.1 Development box setup

As planned, the development environment needs to be prepared in such a way to simulate the real PLA environment. On a Linux machine, all software required to run PLA were installed and any required configuration were applied. Following the online guide for PLA installation including pre-requisite software like IOP and GraphViz dot, I have prepared a light version of guidelines to easily follow-up with installation steps. This installation guide is available in appendix IV.

From programming perspective, eclipse was used - as planned - for developing the new tool. To enrich the development experience, and to focus more on the editor functionality rather than positioning and aligning things on the interface, I installed an eclipse Java GUI designer called WindowBuilder[1]. I used WindowBuilder for initial positioning of panels and components on the screen. It reduced the time required for implementing such non-core parts of the editor.

In addition to having all software setup on the development box, the program needs to be run within a certain folder structure as planned. Folders were created as illustrated in figure 25 with all JLambda scripts and executable Jars placed properly. bp2pl.jar is a package supplied by SRI for keeping all PLA internal java representation to facilitate the integration. iop.jar is the package containing binaries of IOP. PLALib and TinyKBMaude model are required for running PLA. JLambda scripts provided by SRI are used to initialise the KB object, adding rules to KB through shell terminal, drawing dot graphs, exporting to PLA, and some other useful functionalities.

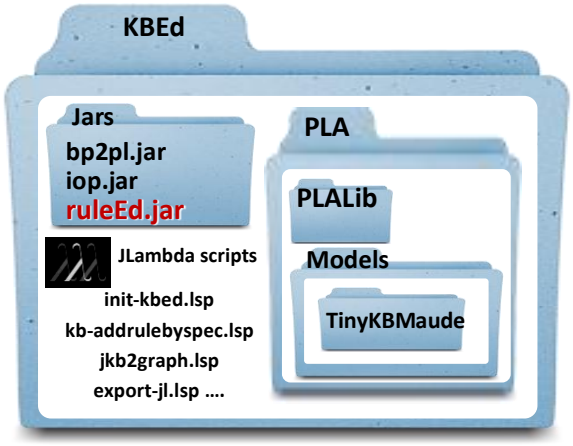


Figure 25 - Implemented folder structure

5.2 JLambda for Integration

In the simulated environment, some process needs to invoke the new editor as of being in a real PLA application. Layered architectural style and the AIR rule suggest an intermediate layer that would communicate with PLA and pass a KB object from PLA to the graphical editor for creating or editing KB rules. The layer needs to return back the updated KB object to PLA to apply required network analysis.

[1] Available from http://www.eclipse.org/windowbuilder

JLambda - a core IOP actor - initialises a KB object with empty rules map and all other specifications included. This action can be invoked by calling the following script on shell terminal:

```
./kbed init-kbed.lsp
```

The command execution output shows all actions performed by JLambda for initialising the KB object as following:

```
bash-4.1$ ./kbed init-kbed.lsp
query.lsp loaded
coloring.lsp loaded
showNewGraph defined
showGraph.lsp loaded
showGraphFuns.lsp loaded
labels.lsp loaded
graphMenu.lsp loaded
ko.lsp loaded
hideEdges.lsp loaded
occs.lsp loaded
color-loc.lsp loaded
exploreRule loaded

Welcome to the JLambda interface to Java (version 1.183),
type ? for help.


>
```

Some rules can also be added using JLambda script - an example of which is provided below:

```
(define r1id (invoke kbed "addRule"  r1spec))
(define r172id (invoke kbed "addRule"  r172spec))
```

Then the editor can be instantiated using the following code on shell:

```
(define kbedGui (object ("ruleEditor.RuleEditor" kbed) ))
```

Following the folder structure and having the editor Jar file ready in Jars folder, this JLambda command would bring up the editor window having the KB data ready for use.

Now we have everything ready for implementing the new editor.

5.3 Building the GUI

5.3.1 Editor foundation

As discussed earlier, each time the editor is started, it will be invoked as a new object with a KB object (of empty rules map) passed as a parameter. By creating

the editor object, all included objects, structures, or fields will be configured for fresh use. To have an intuition of all the objects involved, figure 26 sketches an updated UML class diagram illustrating programming decisions made for implementing the editor. For details about each class fields and methods, refer to the source code in appendix VII.

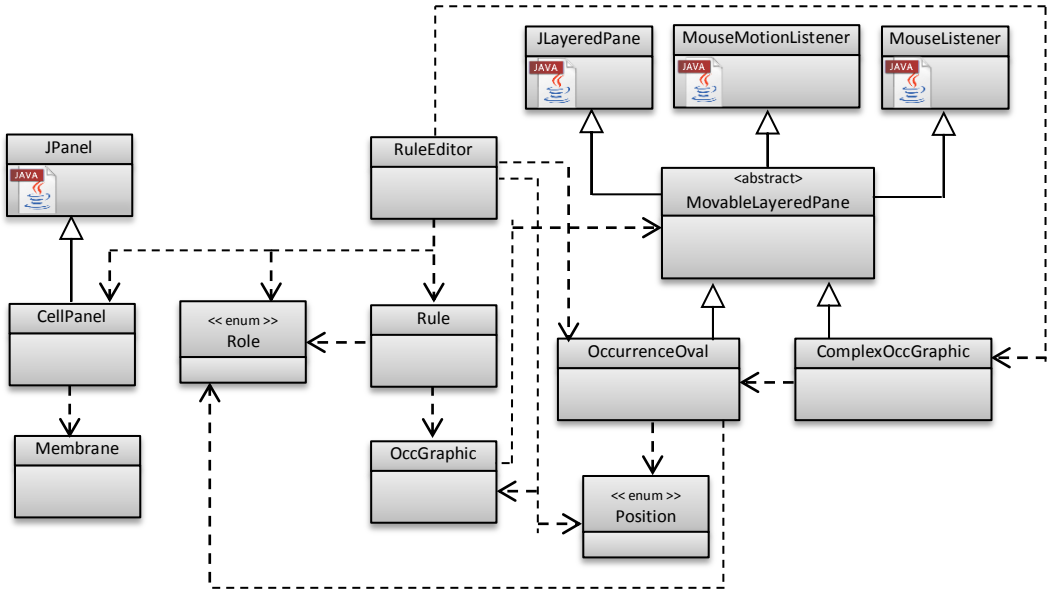


Figure 26 - Detailed UML class diagram

5.3.2 Rule Identifier

Figure 27 - Rule Identifier screen

Basically, this section is the first page displayed for users for starting the process of editing rules. In this view, the user can either select an existing rule for editing or select New Rule for creating a new one. These options are provided through a drop-down list that is populated with rule names from the KB object passed to the editor on instantiation. The list is retrieved from the KB as following:

```
kb.ruleIds()
```

Selecting to create a new rule would refresh all rule information and clear the content of the rule object and all other data structures used for storing editor information while working.

In contrast, selecting an existing rule from the drop-down list would bring up all the information related to this rule from the KB object as following:

```
kb.ruleSpecById(ruleid)
```

Rule information would then be placed in the corresponding fields on the editor and the user then enters or modifies information in the fields as appropriate.

5.3.3 Rule Builder

The main purpose of this editor lies in this part. When users selects wish to start building the rule, two graphical cell spaces (panels) would be provided for representing the cell states before and after the reaction. These spaces are surrounded by editing features to facilitate the graphical tasks. Each cell space will come blank in the case of creating a new rule.

In contrast, if the user is editing an existing rule the cell spaces will be populated with rules occurrences and the panels will display the rule name. These are drawn as a result of retrieving rule graphics entry from *kbGraphics* hash map when selecting the rule to be edited. In the case that a rule graphics entry is not found in the hash map, which means that this is the first time this rule is brought to the editor, a default graphics orientation will be created and the consumed, produced, and controls occurrences will be distributed in default locations on the before and after panels appropriately.

The builder provides the following components to facilitate creating and editing rules:

Sample occurrence:

For convenience, a sample occurrence for demonstrating user selections is implemented to assist users while creating occurrences. A sample is illustrated in figure 28.

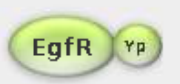


Figure 28 - Occurrence sample

Entity list:

This list reads its items from predefined entity names stored in the KB object. The list is currently limited to proteins which are the most commonly used type of entities. This would be extended in future to include other entities like chemicals. This field is mandatory and selecting any entity would display its name in the sample occurrence.

Modifications bubbles:

The builder provides a list of predefined modifications (obtained from the KB) to be selected to describe the state of each entity. Multiple selections are allowed and can be done via clicking on the required bubbles. Modifications can be de-selected as required by clicking on the corresponding bubble again (see figure 29).

Figure 29 - Occurrence modifications multiple selection

Selecting modification makes the entity oval surrounded by the small bubbles. For convenience and much clarity, modifications that can be visible at a time around an entity can be up to eight. This decision has been made based on SRI recommendation that suggests that there is no case of having more than eight modifications at a time in practical. However, there is no restriction on adding modifications to occurrences and - technically - there is no problem of adding any number of modifications. The only issue would be that modification bubbles will be displayed overlapping each other due to shortage of space. Just for demonstration, an occurrence with eight modifications is illustrated in figure 30.

Figure 30 - A demo of an occurrence with 8 modifications

Locations list:

A predefined list of locations is obtained from the KB object to aid user selection. This field is mandatory as it will determine where each occurrence should be located on the cell panels.

Occurrence role:

A role for each occurrence is specified through a radio button list. Selecting the role will determine to which panel the occurrence should be added; consumed added to Before panel, produced added to After panel, and control added to both of the panels as these substances do not change during reactions. The role also controls the colour of the occurrence; i.e. red for consumed and produced occurrences, and green for controls. This colour coding would assist in recognising the main change in the reaction especially for complex rules.

Adding occurrences:

When the user is done with designing his occurrence, it can be easily added to the proper panel (based on its role) with a button press. This action would add the occurrence to the corresponding occurrence list of the rule object. It also adds the graphics objects to the rule graphics map. The graphics entry is added with occurrence long name as a key for easy retrieval. Long names can be obtained by requesting the KB object for the name through *mkSOccLongName* method. This method would require occurrence specification as a parameter.

Enjoying playing around!

Once an occurrence is added to the proper panel, it can be dragged smoothly with mouse motion as required and dropped wherever appropriate. This way user would have the control over occurrence locations inside the cell and would have much comfort and flexibility to work on their models. Complex occurrences (group of simple occurrences) can also be dragged around together in the same smooth motion. Constructing complex occurrences is discussed below.

Occurrences tree:

A tree listing all occurrences in the rule grouped by role is implemented to give the user better insight about his created rule. Additionally, this tree is used to multi-select occurrences to create a complex occurrence as required using create complex button. This would combine both of the selected occurrences in one. The complex occurrence can be dragged inside panels and dealt with as of individual occurrences.

All components contained in the rule builder are illustrated in figure 31.

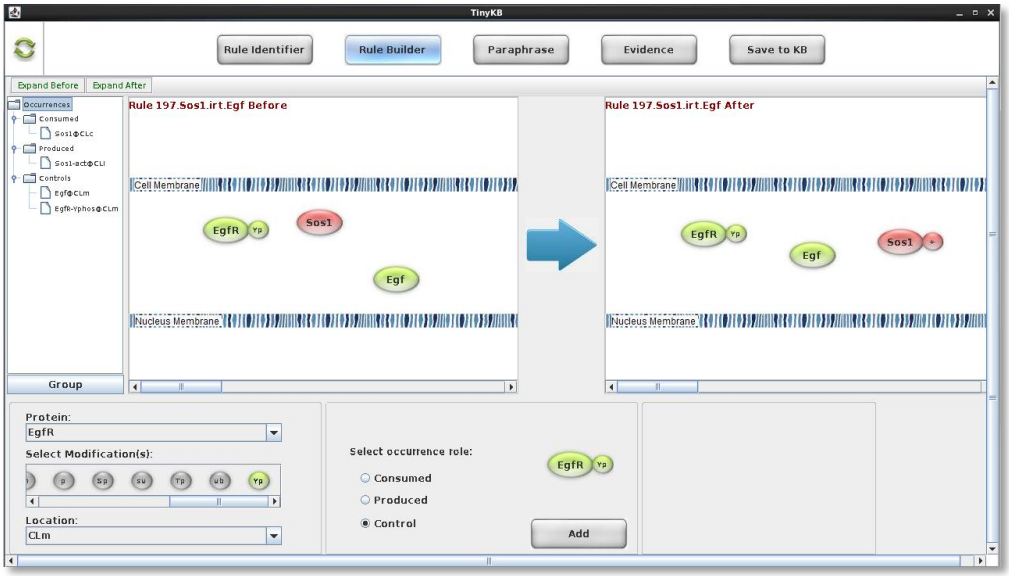


Figure 31 - Rule Builder screen

5.3.4 Rule Paraphrase and Rule Evidence

These two sections are kept for future work as agreed with SRI in order to make more focus on the creation of the rules. Nevertheless, design specifications have been already agreed for evidence as outlined in design specification section. For paraphrase, the implementation would follow SRI initiative for the textual representation. As an example, please refer to figure 17.

Paraphrase would be implemented by retrieving rule information from the rule object and phrase the information as sentences representing consumed, produced, and controls occurrences. This implementation is doable and would not consume much of development time as it can be added as a public method returning a string (e.g. *getParaphrase*) in the rule API with no additional data structures requirement.

Following is a pseudo code for paraphrase implementation:

```
Public static string getParaphrase()
{
        Initialise paraphrase = "If:" + newLine

        loop on consumed and control occurrences
                paraphrase + = "the " + occurrenceLocation + " contains "
                                + entityName + modificationsDescriptions
        end loop
```

```
        paraphrase + = "Then:" + newline
        loop on produced occurrences
               paraphrase + = centityName + consumedModificationsDescriptions
                            + " will " + producedModificationsDescriptions
        end loop
        return paraphrase
}
```

These two sections are in the "to do list" agreed with SRI that would start immediately after submitting this report.

5.3.5 Save to KB

Through the KB object API, the new created rule can be added to the KB rules collection by invoking the *addRule* instance method. This method requires the new rule specification to be passed as a parameter. This would secure having the new rule data saved in the KB.

In addition to saving rule data, the graphical representation designed on the Rule Builder panel need to be stored for future retrievals. For this, the new rule graphics object (including before and after) needs to be added to kbGraphics hash map. The new Graphics entry is stored with rule name as its key to avoid overlapping with other rules. The hash map is serialised and stored as a .ser file in the KB folder to become one of its data sources. On retrieval, this .ser file is loaded into the hash map again as described in the Editor Foundation section.

5.3.6 Apache Ant JAR builder

Apache Ant[1] is a well-known Java library commonly used for building Java applications. Eclipse has a built-in Ant standard feature for facilitating application building. Developing the new tool would not be possible without simulating a PLA environment. The specific folder structure requires the built Jar file of the new tool to be stored a specific location in order to work. Building the application to the specified location (each time a code needs to be tested) is not practical especially with this kind of application. Ant facility provides an automatic building of the application which was very useful for saving time. Ant xml script is available in appendix V.

5.4 Exporting to PLA

The KB object - with all rules stored in it - needs to be exported as Maude code so PLA can invoke it as one of the available KBs in the PLA KB Manager (the first window that appears when running PLA). As one of the integration decisions agreed in coordination with SRI, JLambda is used to export the KB rule map to Maude code. The script was supplied by SRI to test it on the simulated environment. Exporting to Maude code can be applied by executing the following script:

```
(apply exportRules (lookup jkb "rulemap") "PLA/Models/TinyKBMaude/rules.maude")
```

[1] http://ant.apache.org

6 Analysis of results

To test the new editor and analyse the results obtained, following scenarios with detailed procedures were conducted and results were noted and discussed. These procedures serve as a good resource for prospective users. Although most are, some reactions may not be real ones and they are created just for demonstration purposes.

6.1 Invoking the editor

1. From shell terminal[1], cd to KBEd folder (containing JLambda scripts)
2. Execute the command:

```
./kbed init-kbed.lsp
```

3. Enter the following command to check the KB rule map (should be empty on start):

```
(lookup jkb "rulemap")
```

4. Now, enter the following statement to invoke the editor:

```
(define kbedGui (object ("ruleEditor.RuleEditor"
kbed) ))
```

This procedure would be required for every scenario.

6.2 Creating rules with simple occurrences

Rule:

Sos1-Clc + Egf-Clm → Sos1-act-Cli

Procedure:

1. Invoke the editor
2. In the Rule Identifier screen: select <New Rule> under Find Rule
3. Enter Rule id: e.g. Sos1.001
4. Enter Rule description
5. Click Rule Builder:
    a. From Entity list, select Sos1 – watch the sample oval
    b. From Location list, select Clc
    c. Under occurrence role, select consumed radio option
    d. Click Add and watch the Before cell panel
    e. Repeat the process for adding Egf with selecting Clm as location and control as role – watch both cell panels
    f. For Sos-act-Cli, follow the same process but this time: click on + bubble under Modifications, and select produced as role and Cli as location – watch the After cell panel
6. Click Save to KB (see section 6.6)

[1] The new editor runs on Linux Operating system, for resources and tutorials about Linux, you can refer to http://linuxsurvival.com/

Results discussion:

- The occurrences get added to the appropriate cell panels: consumed in Before, produced in After, and control in both.
- Occurrences are added to same default position on top of each other (selecting default positions to reflect the selected location is kept for future work)
- Sample occurrence reflects user selections correctly and occurrences tree gets updated correctly.

Figure 32 shows the outcome of this scenario[1].

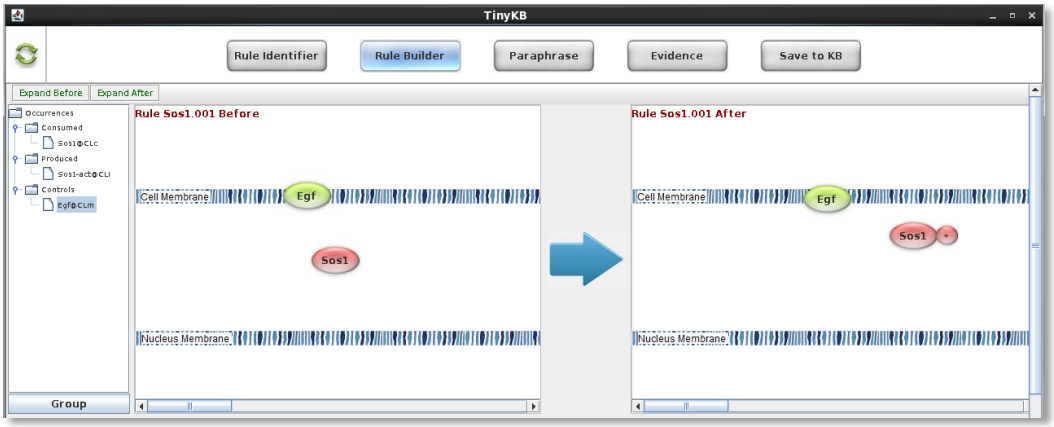


Figure 32 - Creating rules with simple occurrences

6.3 Creating rules with complex occurrences

Rule:

Sos1-Clc + Egf:EgfR-Yphos-Clm → Sos1-act-Cli

Procedure:

1. Invoke the editor
2. Follow the procedure in section 6.2 to create all occurrences involved in this reaction as if they were simple; i.e. the control Egf:EgfR-Yphos-Clm should be created as two separate simple occurrences Egf-Clm and EgfR-Yphos-Clm
3. From the occurrences tree, multi-select the two occurrences included in the complex occurrence (multiple selection is performed by holding the shift key while selecting nodes by mouse or keyboard arrows).
4. Click Group button - watch cell panels
5. Click Save to KB (see section 6.6)

Results discussion:

- The selected occurrences get grouped together showing modifications appropriately on the cell panels
- The occurrence entries in the tree get replaced by one entry representing the new complex occurrence

[1] For clarity, all occurrence graphics in this chapter were dragged and re-positioned as required so they do not overlap.

- The new complex occurrence is positioned in a default location.

Figure 33 shows the outcome of this scenario.

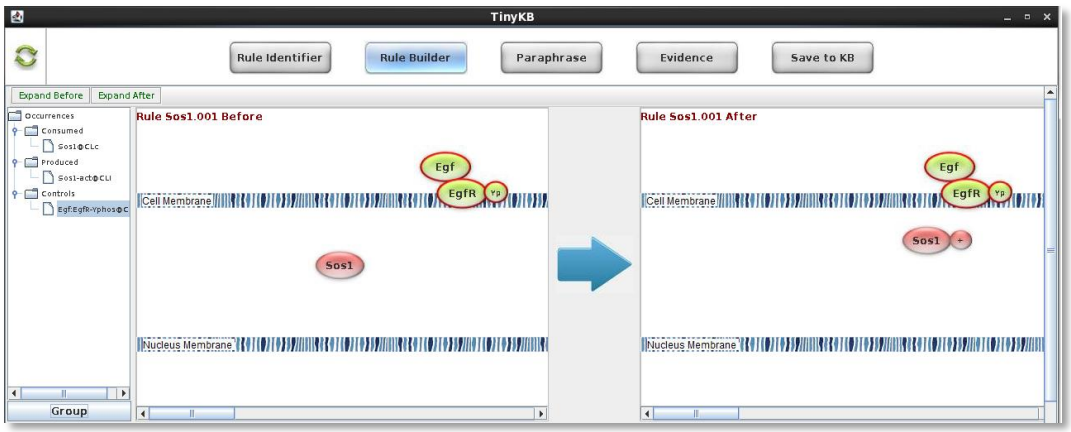


Figure 33 - Creating rules with complex occurrences

6.4 Multiple modification occurrences

Rule:

Sos1-Clc + Egf-ubq-Yphos-Clm → Sos1-act-Cli

Procedure:

1. Invoke the editor
2. Follow the procedures explained in the previous sections to create the occurrences, except when you want to select modifications, click on both the bubbles ubq and Yp – watch the sample
3. After completing your rule, Click Save to KB (see section 6.6)

Results discussion:

- The occurrences get added to the appropriate cell panels: consumed in Before, produced in After, and control in both.
- Sample occurrence reflects user selections and occurrences tree gets updated correctly.

Figure 34 shows the outcome of this scenario.

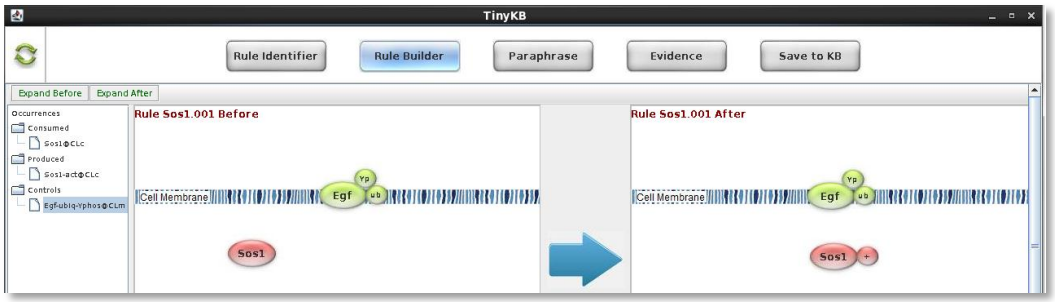


Figure 34 - Rules with multi-modification occurrences

6.5 Selecting and relocating occurrences

- Clicking on any occurrence would make the occurrence highlighted with red border. This applies to both simple and complex occurrences.
- The border shows on all components on the occurrence (i.e. entity ovals and modification bubbles).
- Clicking on the occurrences again de-selects it; i.e. the red border is removed from all components.
- Newly added occurrences are selected by default.
- Selecting occurrences in the tree also selects the corresponding occurrences graphics. De-selecting works similarly.
- Dragging occurrences works perfectly with smooth motion.
- Dragging a complex occurrence works properly as the whole complex moves together.

6.6 Saving rules

1. After completing each rule, click Save to KB
2. Check Find Rule list under Rule Identifier, your new rule id should appear
3. To make sure your rule gets saved to the KB, do the following:
    a. In JLambda already open in shell terminal, invoke the rule map again and the output becomes:

```
(lookup jkb "rulemap")
```

Note the difference between simple and complex occurrences entries in the rule map.

b. Enter the following command in JLambda to see your rule as dot graph (`g3` is the graph name and can be any name):

```
(apply displayJKB jkb "g3")
```

The dot graphs of some created rules are illustrated in figure 35.

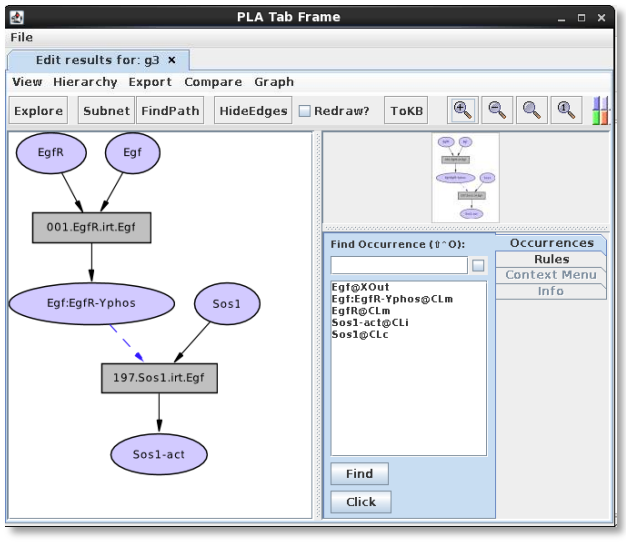


Figure 35 - Dot graph of some rules in the KB created from JLambda

6.7 Exporting rules

After completing all rules in the model, do the following to export your model to PLA:

1. In JLambda already open in shell terminal, run the following command:

```
(apply exportRules (lookup jkb "rulemap") "PLA/Models/TinyKBMaude/rules.maude")
```

2. Leave the terminal, and go to the TinyKBMaude folder
3. Open rules.maude file and see the code of your created rules.
4. The Maude code created as a result of exporting the rule map of some created rules is shown below:

```
rl[172.Pi3k.irt.Egf]:
{ CLc |  clc Pi3k }
{ CLi |  cli }
{ CLm |  clm  Egf : [EgfR -  Yphos] }
  =>
{ CLc |  clc }
{ CLi |  cli Pi3k }
{ CLm |  clm  Egf : [EgfR -  Yphos] }  .

rl[001.EgfR.irt.Egf]:
{ XOut |  xout Egf }
{ CLm |  clm EgfR }
  =>
{ XOut |  xout }
{ CLm |  clm  Egf : [EgfR -  Yphos] }  .

rl[197.Sos1.irt.Egf]:
{ CLc |  clc Sos1 }
{ CLi |  cli }
{ CLm |  clm  Egf : [EgfR -  Yphos] }
  =>
{ CLc |  clc }
{ CLi |  cli [Sos1 -  act] }
{ CLm |  clm  Egf : [EgfR -  Yphos] }  .
```

5. Having this file with all the rules suffices to get your model loaded into PLA
6. To check your model in PLA, cd to TinyKBMaude and run the command `iop`
7. From the PLA KB Manager window > Select Dish > PreDefined > tinyDish
8. PLA window with the new created rules shows as one dot network – see figure 36.

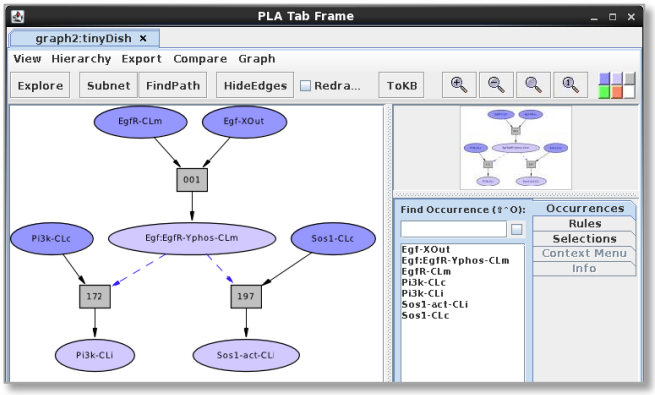


Figure 36 - A model created in the editor loaded into PLA

6.8 Retrieving rules

Retrieving a rule with saved graphics information

Rule:

Sos1-Clc + Egf:EgfR-Yphos-Clm → Sos1-act-Cli

Procedure:

1. Invoke the editor
2. In the Rule Identifier screen: select Sos1.001 under Find Rule (or the name you entered at creation)
3. Watch the Rule Id and the Rule description
4. Click Rule Builder – note the cell panels titles and graphical content. Note the occurrences positions and the tree entries.
5. You can drag some occurrences around and save, then retrieve the rule again and watch the graphics.

Results discussion:

- The rule is retrieved correctly with correct Id and description populated in the corresponding fields
- The cell panels displays the Rule Id correctly
- The tree shows all occurrences properly
- Occurrences graphics are positioned exactly as saved when creating the rule

Retrieving a rule with no graphics information

Rule:

Egf-XOut + EgfR-Clm → Egf:EgfR-Yphos-Clm

Procedure:

1. Using JLambda, from shell terminal, initialise the KB:

```
./kbed init-kbed.lsp
```

2. Create the following rule using JLambda:

```
(define r1id (invoke kbed "addRule"  r1spec))
```

3. Invoke the editor
4. In the Rule Identifier screen: select 001.Egf.irt.EgfR under Find Rule
5. Watch the Rule Id and the Rule description
6. Click Rule Builder – note the cell panels titles and graphical content. Note the occurrences positions and the tree entries.
7. You can drag some occurrences around and save, then retrieve the rule again and watch the graphics.

Results discussion:

- The rule is retrieved correctly with correct Id and description populated in the corresponding fields
- The cell panels displays the Rule Id correctly
- The tree shows all occurrences properly
- Occurrences graphics are created correctly and positioned in default positions

6.9 Conclusion on results

- Users can create rules with both simple and complex occurrences.
- The created rules get saved successfully to the KB and the KB rule map becomes updated.
- Rules can be viewed using JLambda as dot graphs and the graphs reflect the rules correctly.
- Users can open existing rules that are not created in the editor. The editor creates the graphics in default positions. Users can modify the graphics and save it.
- Exporting the updated rule map to PLA as Maude updates the rules code of the PLA model with the new rules code.
- Invoking the PLA model through IOP shows the Petri net network with the newly created rules appropriately.
- Now, the network is ready for PLA analysis...

7 Evaluation

The current PLA system has a number of biologist curators as key users. It is thus important to involve these users in the development life cycle including evaluation stage. To achieve user satisfaction, a series of evaluation procedures need to take place after releasing the system. Following are the procedures used for evaluating the new system:

7.1 User surveys and usability checks

Two surveys were conducted to be in a better position to evaluate this system; one for assessing user satisfaction with the current approach, and the other to evaluate to which extent the approach of the new system would add value.

Surveys were conducted online through SurveyMonkey[1]. Survey samples and result statistics are available in appendix VI.

7.1.1 Evaluation of the PLA Model Creation Approach Survey

Objectives

This survey is intended to measure the level of satisfaction with the current approach used by PLA system for building and manipulating PL networks; i.e. rules.

Participants

SRI International development team members, biologists, and biological models curators.

Results

The respondents of the survey were mostly model creators with some being PLA developer with moderate to extensive use of PLA. One third of them thinks rated the complexity of creating PLA network to be 3. Another one third rated 5, while the rest rated the complexity of performing this task as 8 out of 10.

The time taken for creating PLA networks were specified from being few minutes to more than a week. Respondents' response to clarity of Petri net representation also ranged from being very clear to confusing.

Two third of the respondents know how to program in Maude, while the rest are welling to learn. However, all of them are able to understand the language.

Some respondents think that it is very critical to have a graphical editor for creating rules, others find it not important. Respondents ranged in their view about creating their own models; some of them find it important for understanding particular cell phenomena. Others don not create other than those in PLA.

[1] www.surveymonkey.com

It would have been very valuable to have a wider range of respondents to this survey. However, the acquired results show some limitations in PLA and the willingness to improve specially regarding the clarity of the Petri net representation. The dependency on Maude to create models is a weakness in its own. This makes PLA restricted only to Maude developers and model creators. Users willingness to learn Maude for creating rules proves the high demand for having a way to edit knowledge bases.

Next, we will see how users find the new tool and to which extent it meets their expectations.

7.1.2 New editor satisfaction survey

The system can be evaluated (from user perspective) in terms of whether it is easy to use, easy to install, easy to understand and learn, and any other measure that relates to using the system.

This survey reviews all these impressions.

Objective

This survey is aimed at discovering how satisfying the new graphical editor is and how successful it is in achieving the intended objectives. Results of this survey would assist also in enhancing the tool and seeking new opportunities to improve PLA innovatively.

Participants

SRI International development team members, biologists, and biological models curators.

Results

Survey respondents were developers and model creators with extensive use of PLA. 50% of the respondents find it very simple to create rules in the new tool and in few minutes. While the rest rated the complexity of the task as 5 in scale of 10 in a fair time not exceeding a week.

Half of the respondents find that they would not need Maude any more to create models while the rest disagree.

Respondents rated the clarity, simplicity, and practicality of the new tool between 3 and 4 in a scale of 5.

Respondents find that the new tool would add much value to community by expanding the user community and possibly leading to more comprehensive models.

Others see that this tool would allow users with no Maude background to make rules.

Finally, respondents suggested to have some more features in the tool like search facility.

Although there are many opportunities to improve the new tool, responses proves its importance in serving those users with no logic programming background and this itself shows the achievement of the main project objective.

Software evaluation methodologies may not be limited to the strategy above, other new methodologies can be adopted after releasing the system to ensure up-to-date and high standard compliance. Some possible methods would be possible after releasing the final system and starting using it in reality. A plan is communicated to SRI International for performing these methods and getting feedbacks from users.

7.2 Health check:

Performing a system health check while running the new system would be a sensible check since the software runs diverse technologies that might put overhead on any system. This check will help monitoring system information like disk usage and used memory.

Since the system would be deployed to a Linux operating system, a variety of tools can be used for monitoring Linux system while running the editor with PLA tools. Some tools can be utilised for comparing performance monitored while running PLA with and without the graphical editor. These tools are described in [40].

This method would not be valid in the context of a simulated environment (like the one implemented in this project). This is due to the differences from the real PLA system setup environment and in the IOP implementation.

7.3 Parallel run

Running both systems in parallel after completion (the current PLA and the new one) would be a practical method for evaluating the new system. Key users can be involved to use both systems: current PLA as normal, and the new system as pilot. This way, users can figure out the pros and cons of the new system. Another advantage of this parallel run is to help user migrate from the old to the new system without interruption of their work.

7.4 Specification compliance evaluation

Since the proposed system integrates diverse technologies and standards, external audit would be useful to ensure compliance to standards and specifications. One way would be to involve representatives from each field for evaluation (like IOP and JLambda).

Conclusion and Further Work

I enjoyed abstracting over problems while programming. Just like an interesting game, whenever I find a functionality that needs to be coded to proceed, I just assume it and continue ignoring its details for later... this simplified the work significantly.

Rewrite

Pathway Logic (PL) is a symbolic systems biology formalism based on rewriting logic Maude. PL models are analysed through a graphical Java-based tool PLA. PLA represents biological networks in Petri net representation where each rule (reaction) in the network has an underlying Maude code in the PL Knowledge Base. Creating or editing rules - the fundamental cells in any PL network – requires working on the code level which requires background in Maude programming language.

To extend the user community and make PLA more accessible to users with no Maude background, this project has brought a useful tool for creating and editing rules graphically. I reviewed the literature to provide the necessary background for understanding the related underlying concepts and technologies. These concepts mainly included SSB, PL, PLA, IOP and other alternatives in each field of application. I motivated my method by reviewing the current PLA representation methods followed by some real-world use cases and initiatives for improvement. I then planned - from all perspectives - the graphical editor with integration to PLA as a basis. I went through the implementation details of the new tool with all stages involved.

The new editor was evaluated for verifying its reliability and for revisiting the objectives for assessing achievements. Evaluation methods included surveys conducted with PLA users. Other evaluation methods applicable to a real production environment were also proposed.

Build knowledge bases graphically would assist biologists in documenting and analysing experiments outcomes with no background requirement in formal logic. The graphical representation can extend to include dishes as well. This would improve PLA accessibility to a wider range of users, which in turn could lead to new insights and discoveries in the field of biology. The importance of this project were realised by SRI as well as the University of Bristol. A position paper [41] were presented in the Workshop on Learning and Discovery in Symbolic Systems Biology LDSSB'12 conducted in the University of Bristol, UK, September 2012. The paper discusses the motivation and methodology of this project[1].

This project has big opportunities for future work as it has some parts with on-going work like paraphrase and evidence. Some improvement can be applied on the Rule Builder screen like reflecting location graphically while adding

[1] This report may include information presented in this paper. Those were already included in the original research review

occurrences. The rule editor completed two stages of the planned integration with PLA; creating and editing rules in new or existing KB. The last stage of this integration would be a good future direction were rules are required to be edited on online basis while they are loaded in the PLA.

From programming perspective, the classes developed in this editor for representing objects graphically are a good basis for developing a general graphics Java package. Possible future application might include using such library to create a simulated environment of chemical reactions where substances can be represented graphically and chemical reactions take place when substances are brought together by dragging.

In another context, rule editor aims at developing a visual representation compliant with SBGN [20]. There are effort by SRI for exploring the possibility of semi-automation of the process of inferring rules from the datum knowledge base[1]. Biologists could use the rule editor as an interface to develop rules by working with the Datum inference engine.

[1] Searchable databases for collecting datums (formalisms of simulation models inferred on experimental evidence)

Bibliography

[1] M. Clavel, F. Durán, S. Eker and C. Talcott, All About Maude - A High-Performance Logical Framework: How to Specify, Program and Verify Systems in Rewriting Logic, Springer, 2007.

[2] M. G. Sriram, "Modelling protein functional domains in signal transduction using Maude," *Briefings in Bioinformatics,* pp. 4(3): 236-245, 2003.

[3] C. Chaouiya, "Petri net modelling of biological networks," *Briefings in Bioinformatics,* pp. V(8), No 4, 210 - 219, 2007.

[4] C. L. Talcott, "Pathway Logic," in *In Proceedings of SFM*, 2008.

[5] P. Lincoln and A. Tiwari, "Symbolic Systems Biology: Hybrid Modeling and Analysis of Biological Networks," *Hybrid Systems: Computation and Control HSCC, volume 2993 of LNCS,* pp. 660 - 672, 2004.

[6] N. Martí-Oliet and J. Meseguer, "Rewriting Logic as a Logical and Semantic Framework," *Electronic Notes in Theoretical Computer Science,* pp. Vol. 4, no.1, pp.1 - 36, 1996.

[7] M. Clavel and J. Meseguer, "Reflection and Strategies in Rewriting Logic," *Electronic Notes in Theoretical Computer Science,* p. v(4), 1996.

[8] P. Ballarini, R. Guido, T. Mazza and D. Prandi, "Taming the complexity of biological pathways through parallel computing," *Briefings in Bioinformatics,* pp. Vol 10. No 3. 278 - 288, 2009.

[9] M. Clavel, F. Duran, S. Eker, P. Lincoln, N. Marti-Oliet, J. Meseguer and J. F. Quesada, "Maude: specification and programming in rewriting logic," *Theoretical Computer Science 285,* p. 187 – 243, 2002.

[10] J. A. Goguen, T. Winkler, J. Meseguer, K. Futatsugi and J.-P. Jouannaud, Software Engineering With OBJ: Algebraic Specification in Action, Dordrecht: Kluwer Academic Publishers, 2000, p. 3 – 167.

[11] A. Paz, Z. Brownstein, Y. Ber, S. Bialik, E. David, D. Sagir, I. Ulitsky, R. Elkon, A. Kimchi, K. B. Avraham, Y. Shiloh and R. Shamir, "SPIKE: a database of highly curated human signaling pathways," *Nucleic Acids Research,* pp. (39), D793 – D799, 2011.

[12] M. Hucka, A. Finney, H. M. Sauro, H. Bolouri, J. C. Doyle, H. Kitano and e. al, "The systems biology markup language (SBML): a medium for representation and exchange of biochemical network models," *Bioinformatics,* pp. 19: 524-531, 2003.

[13] L. Strömbäck and P. Lambrix, "Representations of molecular pathways: an evaluation of SBML, PSI MI and BioPAX," *bioinformatics,* pp. Vol. 21 no. 24, pages 4401–4407, 2005.

[14] E. Demir, M. P. Cary, S. Paley, K. Fukuda, C. Lemer, I. Vastrik, G. Wu, P. D'Eustachio, C. Schaefer, J. Luciano, F. Schacherer and I. Martinez-Flores, "The

BioPAX community standard for pathway data sharing," *Nature Biotechnology,* pp. V(28), No. 9, pp 935 - 942, 2010.

[15] P. F. Nielsen and M. D. Halstead, "The evolution of CellML," *Proceedings of the 26th Annual International Conference of the IEEE EMBS,* pp. V(2), 5411 - 5414 , 2004.

[16] C. M. Lloyd, J. R. Lawson, P. J. Hunter and P. F. Nielsen, "The CellML Model Repository," *Bioinformatics,* pp. V(24) No. 18, pp 2122–2123, 2008.

[17] T. Yu, C. M. Lloyd, D. P. Nickerson, M. T. Cooling, A. K. Miller, A. Garny, J. R. Terkildsen, J. Lawson, R. D. Britten, P. J. Hunter and P. M. F. Nielsen, "The Physiome Model Repository 2," *Bioinformatics,* pp. V(27), No. 5, pp 743–744, 2011.

[18] A. K. Miller, J. Marsh, A. Reeve, A. Garny, R. Britten, M. Halstead, J. Cooper, D. P. Nickerson and P. F. Nielsen, "An overview of the CellML API and its implementation," *BMC Bioinformatics,* p. 11:178, 2010.

[19] C. Maus, S. Rybacki and A. M. Uhrmacher, "Rule-based multi-level modeling of cell biological systems," *BMC Systems Biology,* p. 5:166, 2011.

[20] L. Beltrame, E. Calura, R. R. Popovici, L. Rizzetto, D. R. Guedez, M. Donato, C. Romualdi, S. Draghici and D. Cavalieri, "The Biological Connection Markup Language: a SBGN-compliant format for visualization, filtering and analysis of biological pathways," *Bioinformatics,* pp. V(27), No.15, pp 2127 – 2133, 2011.

[21] A. Dräger, N. Rodriguez, M. Dumousseau, A. Dörr, C. Wrzodek, N. Le Novère, A. Zell and M. Hucka, "JSBML: a flexible Java library for working with SBML," *Bioinformatics,* pp. V(27), No. 15, pp 2167 – 2168, 2011.

[22] M. J. Schilstra, L. Li, J. Matthews, A. Finney, M. Hucka and N. Le Novère, "CellML2SBML: conversion of CellML into SBML," *Bioinformatics,* pp. V(22), No. 8, pp 1018 – 1020, 2006.

[23] E. Klipp, W. Liebermeister, A. Helbig, A. Kowald and J. Schaber, "Systems biology standards—the community speaks," *Nature Biotechnology,* pp. 25, 390 - 391, 2007.

[24] F. T. Bergmann and H. M. Sauro, "SBW – A MODULAR FRAMEWORK FOR SYSTEMS BIOLOGY," in *Proceedings of the 2006 Winter Simulation Conference*, 2006.

[25] J. Himmelspach and A. Uhrmacher, "The JAMES II framework for modeling and simulation," *IEEE Computer Society, International Workshop on High Performance Computational Systems Biology,* pp. 101 - 102, 2009.

[26] R. Ewald, J. Himmelspach, M. Jeschke, S. Leye and A. M. Uhrmacher, "Flexible experimentation in the modeling and simulation framework JAMES II- implications for computational systems biology," *Briefings in Bioinformatics,* pp. VOL 11. NO 3. 290-300, 2010.

[27] S. Owre, J. M. Rushby and N. Shankar, "PVS: A Prototype Verification System," *11th International Conference on Automated Deduction (CADE),* pp. 748 - 752, 1992.

[28] I. A. Mason and C. L. Talcott, "IOP: The InterOperability Platform & IMaude: An Interactive Extension of Maude," *Electronic Notes in Theoretical Computer Science (ENTCS),* pp. V(117), pp 315 - 333, 2005.

[29] P. Saraiya, C. North and K. Duca, "Visualizing biological pathways: requirements analysis, systems evaluation and research agenda," *Information Visualization,* pp. 1-15, 2005.

[30] E. R. Gansner, E. Koutsofios, S. C. North and K.-P. Vo, "A Technique for Drawing Directed Graphs," *IEEE Transactions on Software Engineering,* pp. 214 - 230, 1993.

[31] A. Funahashi, Y. Matsuoka, A. Jouraku, M. Morohashi, N. Kikuchi and H. Kitano, "CellDesigner 3.5: A Versatile Modeling Tool for Biochemical Networks," *Proceedings of the IEEE,* pp. V(96), No. 8, pp. 1254 - 1265, 2008.

[32] M. Iersel, T. Kelder, A. R. Pico, K. Hanspers, S. Coort, B. R. Conklin and C. Evelo, "Presenting and exploring biological pathways with PathVisio," *BMC Bioinformatics,* p. 9:399, 2008.

[33] B. Elliott, M. Kirac, A. Cakmak, G. Yavas, S. Mayes, E. Cheng, Y. Wang, C. Gupta, G. Ozsoyoglu and Z. M. Ozsoyoglu, "PathCase: pathways database system," *Bioinformatics,* pp. Vol. 24 no. 21, pages 2526–2533, 2008.

[34] D. M. Bolser, P.-Y. Chibon, N. Palopoli, S. Gong, D. Jacob, V. D. D. Angel, D. Swan, S. Bassi, V. González, P. Suravajhala, S. Hwang, P. Romano, R. Edwards, B. Bishop, J. Eargle, T. Shtatland, N. J. Provart, D. Clements, D. P. Renfro, D. Bhak and J. Bhak, "MetaBase—the wiki-database of biological databases," *Nucleic Acids Research,* 2011.

[35] C. L. Talcott and D. L. Dill, "The Pathway Logic Assistant," *Third International Workshop on Computational Methods in Systems Biology,* pp. 228--239, 2005.

[36] C. Hanson, MIT Scheme Reference Manual, Massachusetts Institute of Technology, 2002.

[37] S. Santiago, C. Talcott, S. Escobar, C. Meadows and J. Meseguer, "A Graphical User Interface for Maude-NPA," *Electronic Notes in Theoretical Computer Science,* pp. V(258), No. 1, pp 3 - 20, 2009.

[38] L. Briesemeister, I. A. Mason, D. Porter and C. L. Talcott, "JLambda: A Language for Interactive Visualization of Formal Models," 2006.

[39] T. J. Lynch, D. W. Bell, R. Sordella, S. Gurubhagavatula, R. A. Okimoto, B. W. Brannigan, P. L. Harris, S. M. Haserlat, J. G. Supko, F. G. Haluska, D. N. Louis, D. C. Christiani, J. Settleman and D. A. Haber, "Activating Mutations in the Epidermal Growth Factor Receptor Underlying Responsiveness of Non–Small-Cell Lung Cancer to Gefitinib," *New England Journal of Medicine,* pp. 2129-2139, 2004.

[40] D. Hoch, Linux System and Performance Monitoring, Redwood City, CA: StrongMail Systems, 2009.

[41] A. Y. Abbas, C. Talcott, M. Knapp and O. Ray, "Towards a graphical rule editor for the Pathway Logic Assistant," *ECML/PKDD 2012 Workshop on Learning and*

Discovery in Symbolic Systems Biology (LDSSB'12), 2012.

[42] J. R. Faeder, "Toward a comprehensive language for biological systems," *BMC Biology,* p. 9: 68, 2011.

[43] C. L. Talcott, "Symbolic Modeling of Signal Transduction in Pathway Logic," *Simulation Conference, WSC 06. Proceedings of the Winter,* pp. 1656 - 1665, 2006.

[44] A. Abate, Y. Bai, N. Sznajder, C. Talcott and A. Tiwari, "Quantitative and Probabilistic Modeling in Pathway Logic," *the IEEE proceedings of the 7th International Symposium on Bioinformatics and Bioengineering,* pp. 922 - 929, 2007.

[45] P. D. Karp, S. M. Paley, M. Krummenacker, M. Latendresse, J. M. Dale, T. J. Lee, P. Kaipa, F. Gilham, A. Spaulding, L. Popescu, T. Altman, I. Paulsen, I. M. Keseler and R. Caspi, "PathwayTools version 13.0: integrated software for pathway/genome informatics and systems biology," *Briefings in Bioinformatics,* pp. VOL 11. NO 1. 40-79, 2009.

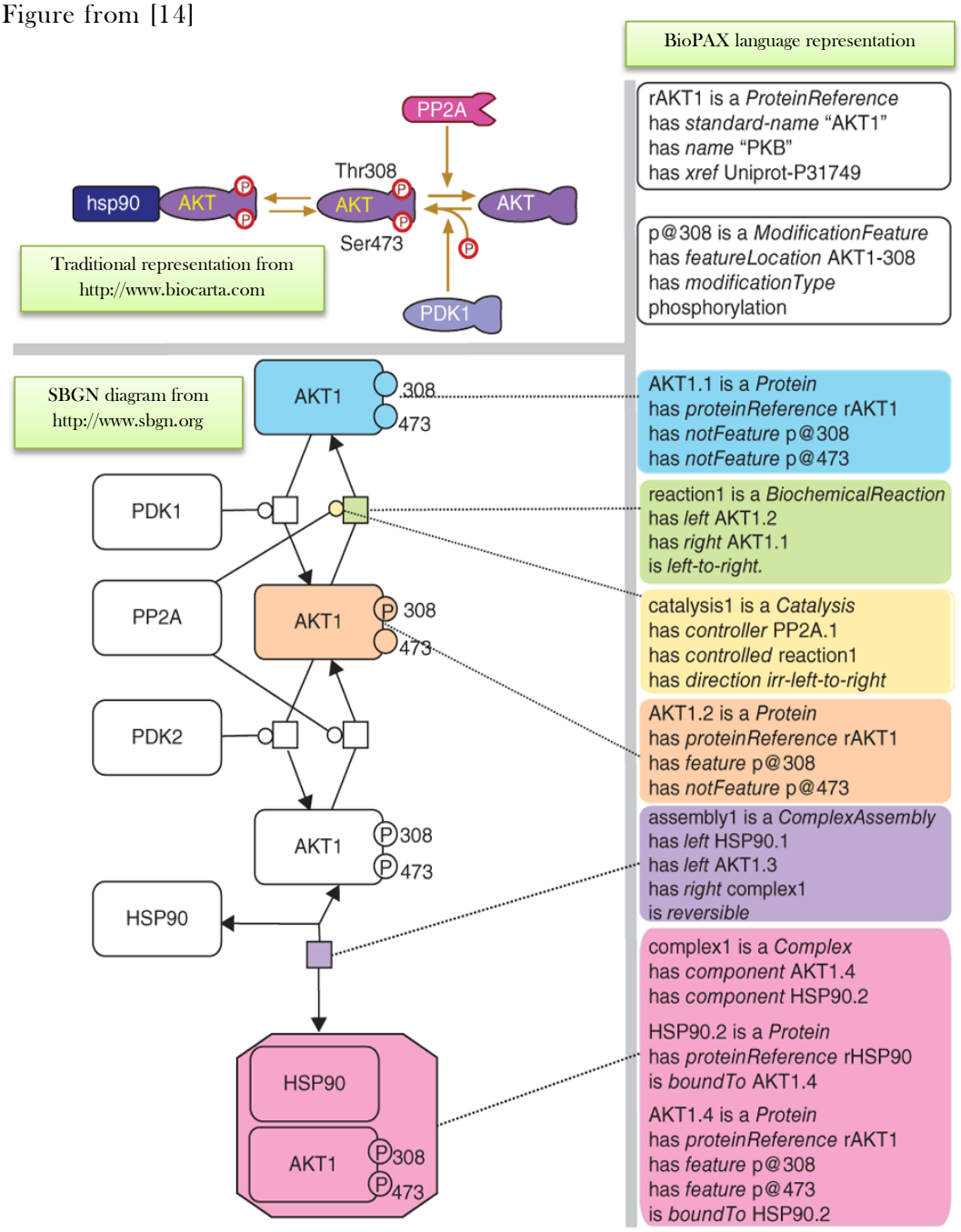
[46] F. N. Demers and J. Malenfant, "Reflection in logic, functional and object-oriented programming: a Short Comparative Study," *Proceedings of the IJCAI'95 Workshop on Reflection and Metalevel Architectures and their Applications in AI.,* p. 29–38, 1995.

[47] K. M. Ferguson, M. B. Berger, J. M. Mendrola, H. S. Cho, D. J. Leahy and M. A. Lemmon, "EGF Activates Its Receptor by Removing Interactions that Autoinhibit Ectodomain Dimerization," *Molecular Cell,* p. 507–517, 2003.

Appendices

Appendix I – BioPAX representation of AKT protein signalling pathway

Figure from [14]



Appendix II- SBML model as exported from PLA

Below representation corresponds to the SmallKB model ; a small part of a model of the response to Egf stimulation in epithelial-like cells (SmallKB is available from http://pl.csl.sri.com).

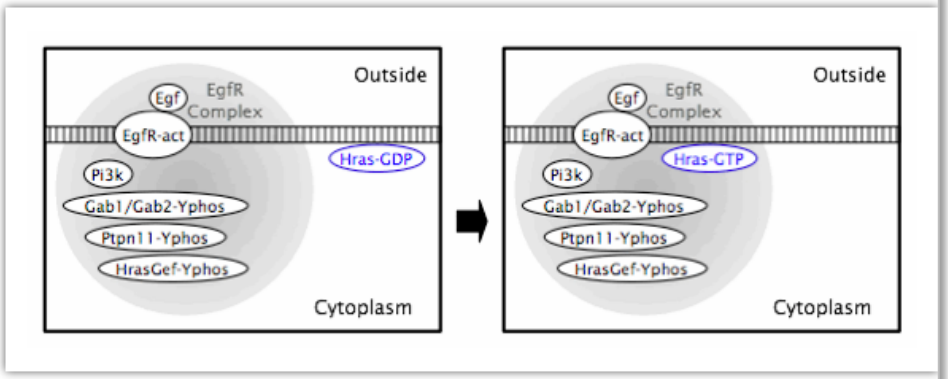
```
<?xml version="1.0" encoding="UTF-8"?>
<sbml level="2" version="1" xmlns="http://www.sbml.org/sbml/level2">
<model id="graph4" name="graph4" >
  <listOfCompartments>
    <compartment id="unknown" name="unrecognized locate"/>
    <compartment id="out" name="outside the cell"/>
    <compartment id="cytosol" size="2.5"/>
    <compartment id="mitochondria" size="0.3"/>
    <compartment id="CLo" name="Cell out" />
    <compartment id="CLm" name="Cell membrane" />
    <compartment id="CLi" name="Cell in" />
    <compartment id="CLc" name="Cell cytosol" />
    <compartment id="NUo" name="Nucleus out" />
    <compartment id="NUm" name="Nucleus membrane" />
    <compartment id="NUi" name="Nucleus in" />
    <compartment id="NUc" name="Nucleus cytosol" />
    <compartment id="MOo" name="Mitochondria Outer out" />
    <compartment id="MOm" name="Mitochondria Outer membrane" />
    <compartment id="MOi" name="Mitochondria Outer in" />
    <compartment id="MOc" name="Mitochondria Outer cytosol" />
    <compartment id="MIo" name="Mitochondria Inner out" />
    <compartment id="MIm" name="Mitochondria Inner membrane" />
    <compartment id="MIi" name="Mitochondria Inner in" />
    <compartment id="MIc" name="Mitochondria Inner cytosol" />
    <compartment id="ERo" name="EndoplasmicReticulum out" />
    <compartment id="ERm" name="EndoplasmicReticulum membrane" />
    <compartment id="ERi" name="EndoplasmicReticulum in" />
    <compartment id="ERc" name="EndoplasmicReticulum cytosol" />
    <compartment id="PXo" name="PeroXisome out" />
    <compartment id="PXm" name="PeroXisome membrane" />
    <compartment id="PXi" name="PeroXisome in" />
    <compartment id="PXc" name="PeroXisome cytosol" />
    <compartment id="GAo" name="Golgi Apparatus out" />
    <compartment id="GAm" name="Golgi Apparatus membrane" />
    <compartment id="GAi" name="Golgi Apparatus in" />
    <compartment id="GAc" name="Golgi Apparatus cytosol" />
    <compartment id="LEo" name="Late Endosome out" />
    <compartment id="LEm" name="Late Endosome membrane" />
    <compartment id="LEi" name="Late Endosome in" />
    <compartment id="LEc" name="Late Endosome cytosol" />
    <compartment id="EEo" name="Early Endosome out" />
    <compartment id="EEm" name="Early Endosome membrane" />
    <compartment id="EEi" name="Early Endosome in" />
    <compartment id="EEc" name="Early Endosome cytosol" />
    <compartment id="LYo" name="Lysosome out" />
    <compartment id="LYm" name="Lysosome membrane" />
    <compartment id="LYi" name="Lysosome in" />
    <compartment id="LYc" name="Lysosome cytosol" />
    <compartment id="CPo" name="Clathrin Coated Pits out" />
    <compartment id="CPm" name="Clathrin Coated Pits membrane" />
    <compartment id="CPi" name="Clathrin Coated Pits in" />
    <compartment id="CPc" name="Clathrin Coated Pits cytosol"/>
    <compartment id="Sig" name="a place for signatures"/>
  </listOfCompartments>
  <listOfSpecies>

    <species id="o6" name="Egf-Out" compartment="Out" initialConcentration="1"/>
    <species id="o7" name="EgfR-CLm" compartment="CLm" initialConcentration="1"/>
    <species id="o2" name="Egf:EgfR-act-CLm" compartment="CLm" initialConcentration="0"/>
  </listOfSpecies>
  <listOfReactions>

    <reaction id="t8" name="1.EgfR.act">
  <listOfReactants>
    <speciesReference species="o6"/>
    <speciesReference species="o7"/>

  </listOfReactants>
  <listOfProducts>
    <speciesReference species="o2"/>

  </listOfProducts>
</reaction>

  </listOfReactions>
</model>
</sbml>
</xml>
```

Appendix III – Additional example of biologists friendly rule

Figures drawn by Merrill Knapp from SRI International (private communication) – the creators of PLA.

Rule 529

▼ **Cartoon**



▼ **Paraphrase**

- ▼ If the Egf-receptor complex contains:
  - Egf bound to activated EgfR
  - Gab1 or Gab2 phosphorylated on tyrosine
  - Pi3k
  - Ptpn11 phosphorylated on tyrosine
  - a RasGef phosphorylated on tyrosine
- And if the inside of the cell membrane contains Hras bound to GDP
- ▼ Then:
  - Hras bound to GDP will relocate to the Egf-receptor complex and become bound to GTP

▶ **Assays**

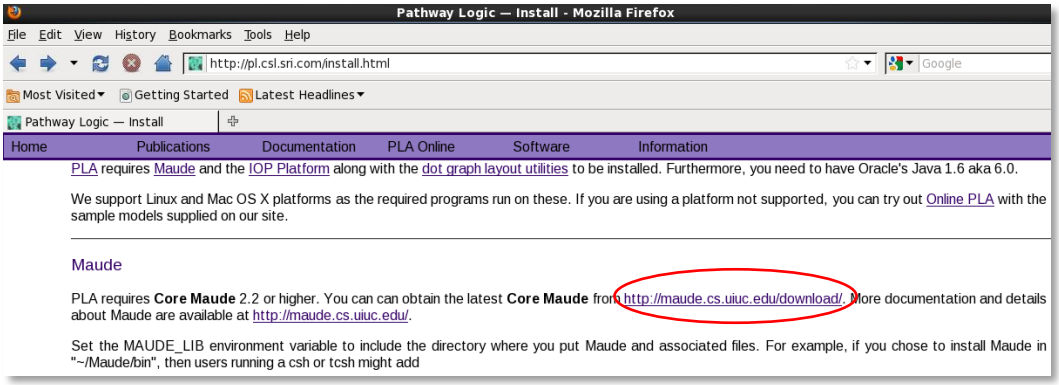
▶ **Evidence**

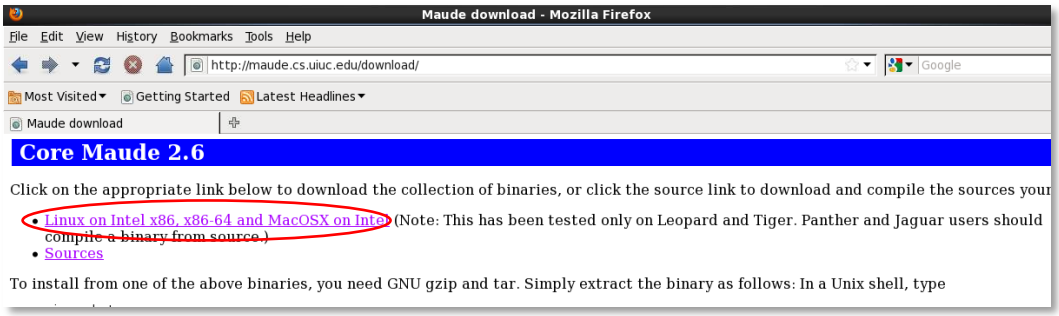
Appendix IV – PLA installation guide

This guide is based on the procedure provided on http://pl.csl.sri.com/install.html (Installing the Pathway Logic Assistant (PLA)).

1. Maude

PLA requires Core Maude 2.2 or higher.





- Zipped folder is downloaded: maude.tar.gz
- Just extracted: name is *maude*
- Renamed to *bin* (for binaries)

- Create a folder anywhere; e.g. Maude folder under Home in linux.
- Move bin folder under Maude, the path is:
  ~\Maude\bin

- Set the MAUDE_LIB environment variable to include the folder above,
  In .bashrc file:

```
export MAUDE_LIB=$HOME/Maude/bin
export PATH=$PATH:$HOME/Maude/bin
```

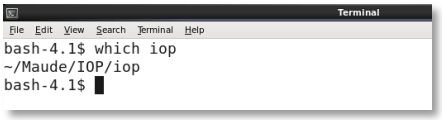
  maude command can now be called from a terminal anywhere in linux – try it!

2. The IOP Platform

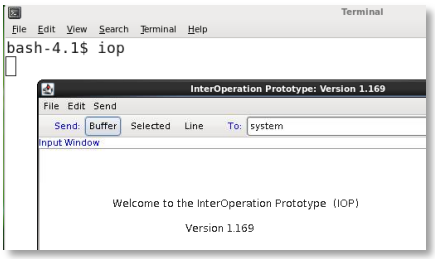
Download and install the binary for your platform from the IOP Binaries page. You will need version 1167 or higher for this release of PLA.

- Download IOP-v1167-linux.zip
- Unzip to:
  ~\Maude\IOP

- Following to be added to .bashrc:
  `export PATH=$PATH:$HOME/Maude/IOP`
  `export IOPBINDIR=$HOME/Maude/IOP`
- Now Linux 'which' command can be used to view iop path:



- Try running IOP as following:



3 Install "dot"

For IOP to support PLA you need to have the GraphViz program dot installed. See the GraphViz website http://www.graphviz.org for more details on installation.

3 Install PLA

Download some models from http://pl.csl.sri.com/download.html.
Now:
To run PLA on a particular model, e.g. SmallKB, just cd into the model's directory, e.g "cd Models/SmallKB", and type "iop". This will start IOP running PLA, there will be an IOP control window which you can ignore, and the PLA KB Manager window, which you should use to interact with PLA.

Appendix V- Ant xml script for building PLA rule editor

The code is generated from the Runnable JAR Export Wizard built-in functionality in Eclipse. The file is called PLAjar.xml

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<project default="create_run_jar" name="Create Runnable Jar for Project PLA_Editor">
    <!--this file was created by Eclipse Runnable JAR Export Wizard-->
    <!--ANT 1.7 is required                                        -->
    <target name="create_run_jar">
        <jar
destfile="/home/masters/2011/aa1641/linux/Desktop/project/KBEd/Jars/ruleEd.jar"
filesetmanifest="mergewithoutmain">
            <manifest>
                <attribute name="Main-Class" value="ruleEditor.RuleEditor"/>
                <attribute name="Class-Path" value="."/>
            </manifest>
            <fileset
dir="/home/masters/2011/aa1641/linux/Desktop/project/PLA_Editor/bin"/>
            <zipfileset excludes="META-INF/*.SF"
src="/home/masters/2011/aa1641/linux/Desktop/project/KBEd/Jars/bp2pl.jar"/>
            <zipfileset excludes="META-INF/*.SF"
src="/home/masters/2011/aa1641/linux/Desktop/project/KBEd/Jars/iop.jar"/>
        </jar>
    </target>
</project>
```

Appendix VI- Survey Samples and Result Statistics

Survey 1: Evaluation of the PLA Model Creation Approach Survey
Objective:
This survey is intended to measure the level of satisfaction with the current approach used by PLA system for building and manipulating PL networks; i.e. rules.

PLA Usage:

1. Describe your category with respect to PLA usage:
    a. Normal user
    b. Model creator
    c. Developer, analyst, or designer
    d. Other, specify: _____
    (Please select more than one when appropriate)

2. How frequently you work with PLA:
    a. Extensively
    b. moderate
    c. Very rare
    d. I don't use it

Assessing PLA model creation

3. In a scale of 10, 1 being *simple* and 10 as *complex*, how do you find creating PLA networks?

4. How long it takes normally to create a small network of no more than 10 rules in PLA?
    a. Very long time (more than a week)
    b. Fair (less than a week)
    c. Fast (less than a day)
    d. No time (in few minutes)
    (If you don't create networks yourself, please select the time the process would take when you request for such network to be created)

5. In PLA, each rule is represented graphically by a Petri-net where reactants, products, and controls are specified by name, modification and location. How clear is this representation specially in a context of a huge network:
    a. Confusing
    b. Fair
    c. Very clear

6. In PLA, there is an option for viewing Maude code when you click on any rule in the network, do you understand Maude?
    a. Yes
    b. No
    c. I don't know about this option

7. In PLA, you need to know how to program in a programming language called Maude in order to create your own models, how confident you find yourself to learn this programming language:
    a. Welling to learn
    b. Not confident
    c. I prefer not to
    d. I already know Maude

Opportunities for improvement

8. To which extent you find it important to have facilities for creating your own models in PLA (like having a simple graphical tool within PLA for creating your rules):
    a. Very critical and would add much value
    b. Important, but not insisting
    c. Not important
    d. I don't know

9. Why do you think it is important to create your own models?
    (those that you cannot create with PLA)

10. In your view, what might make PLA more accessible to users?
    Please list all what you think would make PLA a more user-friendly tool.

Evaluation of the PLA Model Creation Approach Survey

Response Summary

Total Started Survey: 3
Total Finished Survey: 3 (100%)

PAGE: PLA USAGE:

1. Which of the following categories best describes your PLA usage? (Please select all those that are applicable)

	Response Percent	Response Count
Normal user	0.0%	0
Model creator	**66.7%**	**2**
Developer, analyst, or designer	33.3%	1
	Other (please specify)	0
	answered question	**3**
	skipped question	**0**

2. How frequently you work on PLA?

	Response Percent	Response Count
Extensively	**66.7%**	**2**
moderate	33.3%	1
Very rare	0.0%	0
I don't use it	0.0%	0
	answered question	**3**
	skipped question	**0**

PAGE: ASSESSING PLA MODEL CREATION:

3. In a scale of 10, how do you find creating PLA networks?

	simple 1	2	3	4	5	6	7	8	9	complex 10	N/A	Rating Average	Response Count
Complexity of creating PL models:	0.0% (0)	0.0% (0)	**33.3% (1)**	0.0% (0)	**33.3% (1)**	0.0% (0)	0.0% (0)	**33.3% (1)**	0.0% (0)	0.0% (0)	0.0% (0)	5.33	3
											answered question		**3**
											skipped question		**0**

4. How long it takes normally to create a small network of no more than 10 rules in PLA? (If you don't create networks yourself, please select the time the process would take when you request for such network to be created)

	Response Percent	Response Count
Very long time (more than a week)	**33.3%**	**1**
Fair (less than a week)	**33.3%**	**1**
Fast (less than a day)	0.0%	0
No time (in few minutes)	**33.3%**	**1**
	answered question	**3**
	skipped question	**0**

5. In PLA, each rule is represented graphically by a Petri-net where reactants, products, and controls are specified by name, modification and location. How clear is this representation specially in a context of a huge network:

	Response Percent	Response Count
Confusing	**33.3%**	**1**
Fair	**33.3%**	**1**
Very clear	**33.3%**	**1**
	Other (please specify)	1

Responses (1)

Showing **1** text responses

PathNets are clear, SubNets are less clear, Entire Networks are confusing (which is why we use PLA)

answered question	**3**
skipped question	**0**

6. In PLA, there is an option for viewing Maude code when you click on any rule in the network, do you understand Maude?

	Response Percent	Response Count
Yes	**100.0%**	**3**
No	0.0%	0
I don't know about this option	0.0%	0
	answered question	**3**
	skipped question	**0**

7. In PLA, you need to know how to program in a programming language called Maude in order to create your own models, how confident you find yourself to learn this programming?

	Response Percent	Response Count
Welling to learn	33.3%	1
Not confident	0.0%	0
I prefer not to	0.0%	0
I already know Maude	**66.7%**	**2**
	answered question	**3**
	skipped question	**0**

PAGE: OPPORTUNITIES FOR IMPROVEMENT:

8. To which extent you find it important to have facilities for creating your own models in PLA (like having a simple graphical tool within PLA for creating your rules):

	Response Percent	Response Count
Very critical and would add much value	**33.3%**	**1**
Important, but not insisting	0.0%	0
Not important	**33.3%**	**1**
I don't know	**33.3%**	**1**
	answered question	**3**
	skipped question	**0**

9. Why do you think it is important to create your own models? (those that you cannot create with PLA)

	Response Count
	3

Responses (3)

Showing **3** text responses

I created models only through PLA

To understand particular cellular phenomena.

Are there models that cannot be created with PLA?

answered question	**3**
skipped question	**0**

10. In your view, what might make PLA more accessible to users? Please list all what you think would make PLA a more user-friendly tool.

	Response Count
	2

Responses (2)

Showing **2** text responses

Easier ways to build models and to import models from other sources.

A way to view a rule in a topographical setting - ie a picture of the state of the proteins in the cell before the rule fires and another picture of the state after the fires.

answered question	**2**
skipped question	**1**

Survey 2: New editor satisfaction survey

Objectives:

This survey is aimed at discovering how satisfying the new graphical editor is and how successful it is in achieving the intended objectives. Results of this survey would assist also in enhancing the tool and seeking new opportunities to improve PLA innovatively.

PLA Usage

1. Describe your category with respect to PLA usage:
    a. Normal user
    b. Model creator
    c. Developer, analyst, or designer
    d. Other, specify: _____

    (Please select more than one when appropriate)

2. How frequently you work with PLA:
    a. Extensively
    b. moderate
    c. Very rare
    d. I don't use it

Assessing the new graphical editor for model creation

3. In a scale of 10, how do you find creating PL networks using the new tool?

4. How long does it take to create a small network of no more than 10 rules in using the new tool?
    a. Very long time (more than a week)
    b. Fair (less than a week)
    c. Fast (less than a day)
    d. No time (in few minutes)

    (If you don't create networks yourself, please select the time the process would take if you request for such network to be created)

5. In comparison to Petri net representation, how clear is the new Rule Builder representation (Before and After panels with the graphical representation of locations and modifications):
    e. Confusing
    f. Fair
    g. Very clear

6. With the new graphical editor (with complete functionality), would you still need to know Maude to create your models:
    a. Yes
    b. No
    c. Not sure

7. To which extent you find it important to have such graphical tool for creating your own models in PLA:
    a. Very critical and would add much value
    b. Important, but not insisting
    c. Not important
    d. I don't know

8. In a scale of 5, how do you rate this graphical tool - compared to other graphical tools in the same field – in terms of clarity, simplicity, and practicality

Opportunities for improvement

9. What value such graphical tool would add to community (considering any possible further work)?

10. In your view, what might make the new tool more satisfying and much valuable?

New editor satisfaction survey

Response Summary

Total Started Survey: 2
Total Finished Survey: 2 (100%)

PAGE: PLA USAGE:

1. Which of the following categories best describes your PLA usage? (Please select all those that are applicable)

	Response Percent	Response Count
Normal user	0.0%	0
Model creator	**50.0%**	**1**
Developer, analyst, or designer	**50.0%**	**1**
Other (please specify)		0
answered question		**2**
skipped question		**0**

2. How frequently you work on PLA?

	Response Percent	Response Count
Extensively	**100.0%**	**2**
moderate	0.0%	0
Very rare	0.0%	0
I don't use it	0.0%	0
answered question		**2**
skipped question		**0**

PAGE: ASSESSING THE NEW GRAPHICAL EDITOR FOR MODEL CREATION:

3. In a scale of 10, how do you find creating PL networks using the new tool?

	simple 1	2	3	4	5	6	7	8	9	complex 10	N/A	Rating Average	Response Count
Complexity of creating PL models:	**50.0% (1)**	0.0% (0)	0.0% (0)	0.0% (0)	**50.0% (1)**	0.0% (0)	0.0% (0)	0.0% (0)	0.0% (0)	0.0% (0)	0.0% (0)	3.00	2
												answered question	**2**
												skipped question	**0**

4. How long does it take to create a small network of no more than 10 rules in using the new tool? (If you don't create networks yourself, please select the time the process would take if you request for such network to be created)

	Response Percent	Response Count
Very long time (more than a week)	0.0%	0
Fair (less than a week)	**50.0%**	**1**
Fast (less than a day)	0.0%	0
No time (in few minutes)	**50.0%**	**1**
	answered question	**2**
	skipped question	**0**

5. In comparison to Petri net representation, how clear is the new Rule Builder representation (Before and After panels with the graphical representation of locations and modifications):

	Response Percent	Response Count
Confusing	**50.0%**	**1**
Fair	0.0%	0
Very clear	**50.0%**	**1**
	Other (please specify)	1

Responses (1)

Showing **1** text responses

The Rule Builder does not make Petri nets - it only shows one reaction.

answered question	**2**
skipped question	**0**

6. With the new graphical editor (with complete functionality), would you still need to know Maude to create your models:

	Response Percent	Response Count
Yes	**50.0%**	**1**
No	**50.0%**	**1**
Not sure	0.0%	0
	answered question	**2**
	skipped question	**0**

7. To which extent you find it important to have such graphical tool for creating your own models in PLA:

	Response Percent	Response Count
Very critical and would add much value	0.0%	0
Important, but not insisting	0.0%	0
Not important	**50.0%**	**1**
I don't know	**50.0%**	**1**
	answered question	**2**
	skipped question	**0**

8. In a scale of 5, how do you rate this graphical tool - compared to other graphical tools in the same field – in terms of clarity, simplicity, and practicality:

	Low 1	2	3	4	High 5	Rating Average	Response Count
Clarity	0.0% (0)	0.0% (0)	**50.0% (1)**	**50.0% (1)**	0.0% (0)	3.50	2
Simplicity	0.0% (0)	0.0% (0)	**50.0% (1)**	**50.0% (1)**	0.0% (0)	3.50	2
Practicality	0.0% (0)	0.0% (0)	**50.0% (1)**	**50.0% (1)**	0.0% (0)	3.50	2
					answered question		**2**
					skipped question		**0**

PAGE: OPPORTUNITIES FOR IMPROVEMENT:

9. What value such graphical tool would add to community (considering any possible further work)?

	Response Count
	2

Responses (2)

Showing **2** text responses

It would considerably expand the user community and possibly lead to more comprehensive models.

It would allow people who do not want to learn Maude to make rules.

answered question	**2**
skipped question	**0**

10. In your view, what might make the new tool more satisfying and much valuable?

	Response Count
	2

Responses (2)

Showing **2** text responses

Additional search capability to find proteins of interest.

The ability to drag proteins around without having to push all those buttons.

answered question	**2**
skipped question	**0**

Appendix VII- Rule Editor Source Code

The sources code of the rule editor is available in the following pages. Please note that the code is note complete and some (non critical) parts have been suppressed to provide more space.

1

```
package ruleEditor;
//imports suppressed
public class Rule {
        private String ruleid;
        private HashMap<String,Object[]> consumed;
        private HashMap<String,Object[]> produced;
        private HashMap<String,Object[]> controls;
        private String description;
        private String evidence;
        private String paraphrase;
        private HashMap<String,OccGraphic> ruleGraphics;
        public Rule(String ruleid, String description) {/* create a rule*/
                this.ruleid = ruleid;
                this.description = description;
                //instantiations suppressed
        }
        public Rule() {* create an empty rule
                //instantiations suppressed
        }
        public Rule(Object[] rulespec) { /* create a rule from rule specification */
                ruleid = (String) rulespec[1];
                consumed = (HashMap<String,Object[]>) rulespec[2];
                produced = (HashMap<String,Object[]>) rulespec[3];
                controls = (HashMap<String,Object[]>) rulespec[4];
                description = (String) rulespec[5];
                evidence = (String) rulespec[6];;
                //instantiations suppressed
        }
        //Suppressed: setter and getter methods
        public Object[] rulespec()
        {/* get rule info as array with rule specs */
                Object[] rulespec = new Object[7];
                rulespec[0] = "rule";  rulespec[1] = ruleid; rulespec[2] = getConsumed();
                rulespec[3] = getProduced(); rulespec[4] = getControls();
                rulespec[5] = description; rulespec[6] = evidence;
                return rulespec;
        }
        public boolean addOccurrence(Object[] occ, String occId, Role role) {
                if(occurrenceExist(occId)) return false;
                ruleGraphics.put(occId, null);
                switch (role) {
                  case CONSUMED: consumed.put(occId,occ); break;
                  case PRODUCED: produced.put(occId,occ); break;
                  case CONTROL: controls.put(occId,occ); break;
                } return true;
        }
        private boolean occurrenceExist(String occId) {/* check if occurrence exists */
                if(ruleGraphics.containsKey(occId))
                        return true;
                return false;
        }
        public void removeOccurrence(String occId, Role role) {
                ruleGraphics.remove(occId);
                switch (role) {
                  case CONSUMED: consumed.remove(occId); break;
                  case PRODUCED: produced.remove(occId); break;
                  case CONTROL: controls.remove(occId); break;
        }}
        public Object[] getOccSpec(String occId, Role role) {
                if(!occurrenceExist(occId)) return null;
                Object[] occspec = new Object[4];
                switch (role) {
                  case CONSUMED: occspec = consumed.get(occId); break;
                  case PRODUCED: occspec = produced.get(occId); break;
                  case CONTROL: occspec = controls.get(occId); break;
                } return occspec;
}       }
//-------------------------------------------------------------------------------------------
package ruleEditor;
//imports suppressed
public class OccurrenceOval extends MovableLayeredPane {
    private ArrayList<String> mods;
```

2

```
    private Position nextModPosition;
    private JLabel entity;    private Role occRole;
    private ArrayList<JLabel> modGraphics;
    private boolean selected;
    public OccurrenceOval(String ename, Role occRole, ArrayList<String> mods,
boolean movable) {/* create occurrence */
         super(movable);
         selected = false;  this.occRole = occRole;
        entity = new JLabel(ename);        //the main entity graphic
        setRole(occRole);
        add(entity,1);
        nextModPosition = Position.EAST;
        this.mods.addAll(mods);
        addModifications();
         // instantiation and formatting code suppresses
    }
    @Override
    public void mousePressed(MouseEvent e) { super.mousePressed(e); select(!selected); }
    @Override
    public void mouseDragged(MouseEvent e) {super.mouseDragged(e); select(true); }
    @Override
    public void mouseEntered(MouseEvent e) {paintOccurrence(true);}
    @Override
    public void mouseExited(MouseEvent e) {if(!selected)paintOccurrence(false);}
    public void addModification(String mod) {
         if(!mod.equals("")) { addModGraphic(mod); mods.add(mod); }
    }
    public void removeModification(String mod) {
         if(!mod.equals("")) { int i = mods.indexOf(mod); removeModGraphic(mod);
                               mods.remove(i); repaint();}
    }
    private void removeModGraphic(String mod) {
         for(Iterator<JLabel>it=modGraphics.iterator();it.hasNext();) {
                 JLabel modLabel = it.next();
                 if(modLabel.getText().equals(mod)) { it.remove(); remove(modLabel);
                 nextModPosition = Position.getPosition(nextModPosition.id()-1); break;
    }    }        }
    private void addModGraphic(String label) {
         JLabel mod;               //create the modification graphic
         if(occRole == Role.CONSUMED || occRole == Role.PRODUCED)
                 mod = new JLabel(label,
             new ImageIcon(getClass().getResource("images/mod_red.png")), JLabel.CENTER);
         else mod = new JLabel(label,
            new ImageIcon(getClass().getResource("images/mod_green.png")), JLabel.CENTER);
         //formatting suppressed
         add(mod,0);
         modGraphics.add(mod);
         switch (nextModPosition) {
  case EAST: mod.setBounds(70, 13, 50, 50); nextModPosition = Position.NORTH_EAST; break;
  case NORTH_EAST: mod.setBounds(56, -7, 50, 50); nextModPosition = Position.NORTH; break;
  case NORTH: mod.setBounds(30, -12, 50, 50); nextModPosition = Position.NORTH_WEST; break;
  case NORTH_WEST: mod.setBounds(4, -7, 50, 50); nextModPosition = Position.WEST; break;
  case WEST: mod.setBounds(-11, 13, 50, 50); nextModPosition = Position.SOUTH_WEST; break;
  case SOUTH_WEST: mod.setBounds(4, 33 ,50, 50); nextModPosition = Position.SOUTH; break;
  case SOUTH: mod.setBounds(30, 39, 50, 50); nextModPosition = Position.SOUTH_EAST; break;
  case SOUTH_EAST: mod.setBounds(56, 33, 50, 50); nextModPosition = Position.EAST; break;
  }      }
    private void addModifications() {
                 for (String m : mods) if(!m.equals(""))addModGraphic(m);
    }
    // setters and getters suppressed
   public void clear() { //suppressed: clear all contents}
   public void select(boolean select) { selected = select;  paintOccurrence(selected); }
   public void paintOccurrence(boolean border) { //suppressed: change graphics icon image
   with/without border}
```

```
import java.awt.Point;
public abstract class MovableLayeredPane extends JLayeredPane implements MouseListener,
MouseMotionListener {
    private Point position;
    private int xCorrection, yCorrection;
    private boolean movable;
    public MovableLayeredPane(boolean movable) {
        this.movable = movable;
         if(movable) { addMouseListener(this); addMouseMotionListener(this); }
    }
    @Override
    public void mousePressed(MouseEvent e) {
           Point parentLocation = getParent().getLocation();
           xCorrection = parentLocation.x - e.getX();
           yCorrection = parentLocation.y - e.getY();
    }
    @Override
    public void mouseDragged(MouseEvent e) {
           position = SwingUtilities.convertPoint(this, e.getX(), e.getY(), getParent());
           setLocation(position.x + xCorrection , position.y + yCorrection );
    }
    public abstract void select(boolean select);
    // setters, getters and overridden mouse events methods suppressed
//-------------------------------------------------------------------------------------------
   //Imports suppressed
   public class ComplexOccGraphic extends MovableLayeredPane {
        private ArrayList<OccurrenceOval> occGraphics;
        private int lastX, lastY;
        private boolean selected;
        public ComplexOccGraphic(boolean mouseListener) {
           super(mouseListener);
            // instantiation and formatting suppressed
        }
        @Override
        public void mousePressed(MouseEvent e) {super.mousePressed(e); select(!selected); }
        @Override
       public void mouseDragged(MouseEvent e) {super.mouseDragged(e); select(true); }
        @Override
        public void mouseEntered(MouseEvent e) {paintOccurrence(true);}
        @Override
        public void mouseExited(MouseEvent e) {if(!selected)paintOccurrence(false);}
        private void paintOccurrence(boolean border) {
                for(OccurrenceOval occ : occGraphics) occ.paintOccurrence(border);
        }
        public void addOccurrence(OccurrenceOval occ) {
                lastX = lastX + 20;  lastY = lastY + 32;
                occGraphics.add(occ);  occ.setBounds(lastX,lastY,110,77); add(occ);
        }
        //setters and getters suppressed
        public void select(boolean select) {
                for(OccurrenceOval occ : occGraphics) occ.select(select);
                selected = select;
   }    }
//-------------------------------------------------------------------------------------------
import java.io.Serializable;
public class OccGraphic implements Serializable {
        private MovableLayeredPane occBefore, occAfter;
        public OccGraphic(MovableLayeredPane occBefore, MovableLayeredPane occAfter) {
                this.occBefore = occBefore; this.occAfter = occAfter;
        }
        // setters and getters suppressed
}
```



```
import java.awt.Color;
public class CellPanel extends JPanel {
        private JScrollPane scrollableContainer;
        private JLayeredPane cellPanel;
        private int width, height;
        private Membrane cellMembrane, nucleusMembrane;
        private boolean expanded;
        private String title;
        private int layer;
        private JLabel lblTitle;
        public CellPanel(String title, int width, int height, Membrane cm, Membrane nm) {
                this.width = width; this.height = height;
                cellMembrane = cm; nucleusMembrane = nm;
                expanded = false;
                createScrollableContainer(); createCellPanel();
                this.title = title;
                addToCell(cellMembrane); addToCell(nucleusMembrane);
                lblTitle = new JLabel(title);
                //formatting suppressed
                addToCell(lblTitle);
        }
        // setters and getters suppressed
        private void createCellPanel() {
                cellPanel = new JLayeredPane();
                //formatting suppressed
                //add the cell panel inside the scrollable container
                scrollableContainer.setViewportView(cellPanel);
        }
        private void createScrollableContainer() {
                scrollableContainer = new JScrollPane();
                add(scrollableContainer);
                //formatting suppressed
        }
        public void expand(boolean toExpand) {
                if(!expanded && toExpand) { width = ( width * 2 ) + 220;
                        cellPanel.setPreferredSize(new Dimension(width, height-20));
                } else if(expanded && !toExpand) {
                        width = ( width - 220 ) / 2;
                cellPanel.setPreferredSize(new Dimension(( width * 2 ) + 220, height-20));
                }
                expanded = toExpand;
                scrollableContainer.setPreferredSize(new Dimension(width/2, height));
        }
        public void addToCell(Component comp) {
                layer++; cellPanel.add(comp, new Integer(layer));
        }
        public void clear() { // suppressed: clear all content }
 }}
//-------------------------------------------------------------------------------------------
// imports from Java libraries suppressed
import bp2pl.KBEd;
public class RuleEditor {
        private int editMode;     //to specify either new or edit existing
        protected static final int NEW = 1;
        protected static final int EDIT = 2;
        private KBEd kb; // to hold KB object passed from PLA
        // suppressed: Normal components like buttons and text fields
        private CellPanel panelWest;
        private CellPanel panelEast;
        //a hash map to hold KB rules graphics information
        private HashMap<String, HashMap<String,OccGraphic>> kbGraphics;
        private Rule rule;        //current rule
        private ArrayList<String> modnames;
```

```
        private Role occRole;
        private String[] modifications;
        private HashMap<String, String> abbrevMap;
        private ArrayList<String> mods;
        public RuleEditor( KBEd kb) { /* Create the rule editor instance */
                this.kb = kb;
                /*instantiations suppressed */
                editMode = NEW;
                occRole = Role.CONSUMED;
              modifications = getData(kb.modSpecs(),1,4);//get modifications list from KB
                Arrays.sort(modifications, new stringComparator());
                abbrevMap = (HashMap<String, String>)
                                kb.getAttribute("modificationAbbreviations");
                if(abbrevMap.size()==0) createAbbrevMap();
                setupGUI();      //create the editor window components
                getKBGraphics();
                frame.setVisible(true);   //show the editor window
        }
        private void createAbbrevMap() { /* creates abbreviations list for modifications
                for(String m : modifications)
                        if(m.length()>3) abbrevMap.put(m, m.substring(0,3));
                        else abbrevMap.put(m, m);
        }
        private void getKBGraphics() { /* Gets the rules graphics information from file
                try{     FileInputStream fileStream =
                                new FileInputStream(kb.getKBname()+".ser");
                        ObjectInputStream objStream = new ObjectInputStream(fileStream);
                        kbGraphics = (HashMap<String, HashMap<String, OccGraphic>>)
                                        objStream.readObject();
                        objStream.close();
                        fileStream.close();
                } catch(Exception ex){  //error - suppressed }
        }
        public void refreshAll() { /* resets all editor contents for a new rule */ }
        private void clearPanels() { panelWest.clear();panelEast.clear(); }
        private Object[] rulesFromKB() { /* get rule id's from the KB */
                List<Object> list = new ArrayList<Object>(kb.ruleIds());
                return list.toArray();
        }
        private void setupGUI() { /* Setup the contents of the editor GUI. */
        // suppressed for space: setting the main frame and containing panels
        // and adding all contents with layouts settings
          btnSaveToKb.addActionListener(new ActionListener() {
                public void actionPerformed(ActionEvent e) {
                  if(verifyAll()){
                        save();
                        //suppressed: reset and clear fields
                }} });
                btnRefresh.addActionListener(new ActionListener() {
                        public void actionPerformed(ActionEvent e) {
                                cbFindRule.setModel(new
                                        DefaultComboBoxModel(rulesFromKB()));
                                cbFindRule.addItem("< New Rule >");
                                cbFindRule.setSelectedItem("< New Rule >");
                } });
                setupTree();

        occTree.getSelectionModel().setSelectionMode(TreeSelectionModel.CONTIGUOUS_TREE_SEL
ECTION);
                occTree.addTreeSelectionListener(new TreeSelectionListener() {
                public void valueChanged(TreeSelectionEvent ev) {
                TreePath[] occPaths = occTree.getSelectionPaths();
```



```
                if(occPaths!=null) {
                        ArrayList<String> selectedOccs = getSelectedOccs(occPaths);
                        if(selectedOccs.size()!=0) {
                                for(String occ : rule.getRuleGraphics().keySet())
                                if(selectedOccs.contains(occ)) electOccurrence(occ,true);
                                else selectOccurrence(occ,false);
                } } } }
                private ArrayList<String> getSelectedOccs(TreePath[] occPaths) {
                        ArrayList<String> occs = new ArrayList<String>();
                        for(TreePath occPath : occPaths) {
                                DefaultMutableTreeNode occNode = (DefaultMutableTreeNode)
                                                occPath.getLastPathComponent();
                                occs.add(occNode.toString());
                        } return occs;
                } });
                btnGroup.addActionListener(new ActionListener() {
                        public void actionPerformed(ActionEvent e) {
                          TreePath[] paths = occTree.getSelectionPaths();
                          if(paths.length>1) {
                                ArrayList<Object[]> soccs = new ArrayList<Object[]>();
                        ComplexOccGraphic coccBefore = new ComplexOccGraphic(true);
                        ComplexOccGraphic coccAfter = new ComplexOccGraphic(true);
                        DefaultTreeModel model = (DefaultTreeModel) occTree.getModel();
                                for(TreePath occPath : paths) {
                                        DefaultMutableTreeNode occNode =
                                 (DefaultMutableTreeNode) occPath.getLastPathComponent();
                                if(occNode != null) {
                                        String occ = occNode.toString();
                                            OccGraphic occG =
                                                 rule.getOccurrenceGraphic(occ);
                                            if(occG!=null) {
                                         MovableLayeredPane occBefore = occG.occBefore();
                                         MovableLayeredPane occAfter = occG.occAfter();
                                                model.removeNodeFromParent(occNode);
                                                occTree.repaint();
                                                if(occBefore!=null) {
                                                        occBefore.setMovable(false);
                                                        if(occAfter==null)
                                                           occRole = Role.CONSUMED;
                                coccBefore.addOccurrence((OccurrenceOval) occBefore);
                                                }
                                                if(occAfter!=null) {
                                                        occAfter.setMovable(false);
                                                        if(occBefore==null)
                                                occRole =  Role.PRODUCED;
                                                        else
                                                        occRole = Role.CONTROL;
                                coccAfter.addOccurrence((OccurrenceOval) occAfter);
                                                }
                                      Object[] soccspec = rule.getOccSpec(occ, occRole);
                                                   soccs.add(soccspec);
                                                rule.removeOccurrence(occ, occRole);
                                          } }  }
                                        panelWest.addToCell(coccBefore);
                                        panelEast.addToCell(coccAfter);
                                        String locname =
                                        comboBoxLocation.getSelectedItem().toString();
                                        Object[] coccspec = {"cocc",soccs,locname};
                        String occId = kb.mkCOccLongName(coccspec);
                        boolean added = false;
                        added = rule.addOccurrence(coccspec, occId, Role.CONTROL);
                        if(!added) lblMessages.setText("Occurrences already grouped...");
                         else {
```

```
                                        lblMessages.setText("");
                                        rule.saveOccurrenceGraphic(occId, new
                                                OccGraphic(coccBefore,coccAfter));
                                        addToTree(occId, Role.CONTROL.id()); } } } });
//creating the Before cell panel with cell and nucleus membranes
Membrane cellMembraneBefore =
new Membrane(this.getClass().getResource("images/stripCell.png"),new Point(0,101),2220,23);
Membrane nucleusMembraneBefore = new
Membrane(this.getClass().getResource("images/stripNucleus.png"),new Point(0,275),2220,23);
panelWest = new CellPanel(" Before", 1000, 380, cellMembraneBefore, nucleusMembraneBefore);
//creating the After cell panel with cell and nucleus membranes
Membrane cellMembraneAfter =
new Membrane(this.getClass().getResource("images/stripCell.png"),new Point(0,101),2220,23);
Membrane nucleusMembraneAfter = new
Membrane(this.getClass().getResource("images/stripNucleus.png"),new Point(0,275),2220,23);
panelEast = new CellPanel(" After", 1000, 380, cellMembraneAfter, nucleusMembraneAfter);
                cbFindRule.addActionListener (new ActionListener () {
                public void actionPerformed(ActionEvent e) {
                    if(cbFindRule.getSelectedItem().equals("< New Rule >")) refreshAll();
                    else {
                         String ruleid = cbFindRule.getSelectedItem().toString();
                         refreshAll();
                         editMode = EDIT;
                         txtRuleName.setEnabled(false);
                         Object[] rulespec = kb.ruleSpecById(ruleid);
                         rulespec[2] = createOccMap((ArrayList<Object[]>) rulespec[2]);
                         rulespec[3] = createOccMap((ArrayList<Object[]>) rulespec[3]);
                         rulespec[4] = createOccMap((ArrayList<Object[]>) rulespec[4]);
                         if(rulespec != null) {
                                 rule = new Rule(rulespec);
                                 txtRuleName.setText(rule.getRuleId());
                                  txtDescription.setText(rule.getDescription());
                                  if(kbGraphics.containsKey(rule.getRuleId())) {
                                   rule.setRuleGraphics(kbGraphics.get(rule.getRuleId()));
                                                drawRuleGraphics(); }
                                  else createDefaultGraphics();
                         else refreshAll();
                } } });
                String[] proteins = getData(kb.entitySpecs(),1,-1);
                Arrays.sort(proteins, new stringComparator());
                comboBoxProtein = new JComboBox(proteins);
                addModificationLabels();
                String[] locations = getData(kb.locSpecs(),1,2);
                Arrays.sort(locations, new stringComparator());
                comboBoxLocation = new JComboBox(locations);
                rdbtnConsumed.addActionListener(new ActionListener() {
                        public void actionPerformed(ActionEvent e) {
                                occRole = Role.CONSUMED;
                                occSample.setRole(occRole);
                                repaintModLabels(); }});
                btnAdd.addActionListener(new ActionListener() {
                        public void actionPerformed(ActionEvent e) {
        if(comboBoxProtein.getSelectedItem().toString().equals("") ||
comboBoxLocation.getSelectedItem().toString().equals(""))
                                lblMessages.setText("Please select entity and location...");
                                else {
        String ename = comboBoxProtein.getSelectedItem().toString();
        String locname = comboBoxLocation.getSelectedItem().toString();
        ArrayList<String> occModnames = new ArrayList<String>();
        occModnames.addAll(modnames);
        Object[] soccspec = {"socc",ename,occModnames,locname};
        String occId = kb.mkSOccLongName(soccspec); boolean added = false;
```



```
        added = rule.addOccurrence(soccspec, occId, occRole);
        if(!added) lblMessages.setText("Occurrence already exists...");
        else {
                OccurrenceOval occ = new OccurrenceOval(ename, occRole, mods, true);
                OccurrenceOval occ2 = null;
                switch(occRole) {
                  case CONSUMED: panelWest.addToCell(occ); break;
                  case PRODUCED: occ2 = occ; occ = null; panelEast.addToCell(occ2); break;
                  case CONTROL: occ2 = new OccurrenceOval(ename, occRole, mods, true);
                        panelWest.addToCell(occ); panelEast.addToCell(occ2); break;
                }
                rule.saveOccurrenceGraphic(occId, new OccGraphic(occ,occ2));
                addToTree(occId, occRole.id());
                } } } });
                occSample = new OccurrenceOval("sample", occRole, mods, false);
        protected HashMap<String,Object[]> createOccMap(ArrayList<Object[]> occArray) {
                HashMap<String,Object[]> occMap = new HashMap<String,Object[]>();
                for(Object[] occspec : occArray){
                        String occId = null;
                        if(occspec[0].equals("socc")) occId = kb.mkSOccLongName(occspec);
                        else occId = kb.mkCOccLongName(occspec);
                        occMap.put(occId, occspec);
                } return occMap;
        }
        protected void selectOccurrence(String occ, boolean selectValue) {
                    OccGraphic occG = rule.getOccurrenceGraphic(occ);
                    if(occG!=null) {
                         MovableLayeredPane occBefore = occG.occBefore();
                         MovableLayeredPane occAfter = occG.occAfter();
                         if(occBefore!=null) occBefore.select(selectValue);
                         if(occAfter!=null) occAfter.select(selectValue);  }}
        protected String getModFullName(String m) {
                String fullName = "";
                for(Iterator<String> it = abbrevMap.keySet().iterator();it.hasNext();)
                 { fullName = it.next(); if(m.equals(abbrevMap.get(fullName))) break; }
                return fullName;
        }
        private void addToTree(String occId, int i) {
                MutableTreeNode node = new DefaultMutableTreeNode(occId);
                DefaultTreeModel model = (DefaultTreeModel) occTree.getModel();

                MutableTreeNode root = (MutableTreeNode) model.getRoot();
                MutableTreeNode parent = (MutableTreeNode) root.getChildAt(i);
                int index = parent.getChildCount();
                model.insertNodeInto(node, parent, index);
                TreeNode[] nodesBefore = model.getPathToRoot(node);
                TreePath nodePath = new TreePath(nodesBefore);
                occTree.setSelectionPath(nodePath);
                occTree.scrollPathToVisible(nodePath);
        }
        private void setupTree() {
                occTree.setModel(new DefaultTreeModel(
                                new DefaultMutableTreeNode("Occurrences") {
                                add(new DefaultMutableTreeNode("Consumed"));
                                add(new DefaultMutableTreeNode("Produced"));
                                add(new DefaultMutableTreeNode("Controls")); } ));
                occTree.repaint(); }
        private String[] getData(ArrayList<Object[]> specs, int dataColumn, int decsColumn)
        {
                ArrayList<String> nameList = new ArrayList<String>();
                for(Object[] spec:specs)
                        if(decsColumn==-1) nameList.add((String) spec[dataColumn]);
                        else nameList.add((String) spec[dataColumn]);
```

```
        return nameList.toArray(new String[nameList.size()]);
}
private void save() {
        rule.setRuleId(txtRuleName.getText());
        rule.setDescription(txtDescription.getText());
        saveKBGraphics();
        kb.addRule(rule.rulespec());
}
private void saveKBGraphics() {
        String ruleid = rule.getRuleId();
        if(kbGraphics.containsKey(ruleid))
                kbGraphics.remove(ruleid);
        kbGraphics.put(ruleid, rule.getRuleGraphics());
        try{FileOutputStream fileStream = new
FileOutputStream(kb.getKBname()+".ser");
                ObjectOutputStream objStream = new
ObjectOutputStream(fileStream);
                objStream.writeObject(kbGraphics);
                objStream.close();
                fileStream.close();
        } catch(IOException ex) { //error }
        }
private void createDefaultGraphics() {
        createOccGraphics(rule.getConsumed(),Role.CONSUMED);
        createOccGraphics(rule.getProduced(),Role.PRODUCED);
        createOccGraphics(rule.getControls(),Role.CONTROL);
        if(kbGraphics.containsKey(rule.getRuleId()))
                kbGraphics.remove(rule.getRuleId());
        kbGraphics.put(rule.getRuleId(), rule.getRuleGraphics());
}
private void createOccGraphics(ArrayList<Object[]> occs, Role role) {
        for(Object[] occspec : occs){
                if(occspec[0].equals("socc")) {
                        String occId = kb.mkSOccLongName(occspec);
                        rule.saveOccurrenceGraphic(occId,
createSoccGraphic(occId, occspec, role));
                        addToTree(occId, role.id());
                } else if (occspec[0].equals("cocc")) {
                        String locname = (String) occspec[2];
                        ArrayList<Object[]> soccs = new ArrayList<Object[]>();
                        soccs.addAll((ArrayList<Object[]>)occspec[1]);
                        String coccId = kb.mkCOccLongName(occspec);
                        ComplexOccGraphic coccBefore = new
ComplexOccGraphic(true);
                        ComplexOccGraphic coccAfter = new
ComplexOccGraphic(true);
                        for(Object[] soccspec : soccs) {
                                String soccId = kb.mkSOccLongName(occspec);
                                OccGraphic occG = createSoccGraphic(soccId,
soccspec, role);
                       MovableLayeredPane occBefore = occG.occBefore();
                       MovableLayeredPane occAfter = occG.occAfter();
                       if(occBefore!=null) {
                        occBefore.setMovable(false);
                       coccBefore.addOccurrence((OccurrenceOval) occBefore);
                       }
                       if(occAfter!=null) {
                        occAfter.setMovable(false);
                        coccAfter.addOccurrence((OccurrenceOval) occAfter);
                       }
                        } panelWest.addToCell(coccBefore);
                        panelEast.addToCell(coccAfter);
                rule.saveOccurrenceGraphic(coccId, new
```



```
        OccGraphic(coccBefore,coccAfter));
                        addToTree(coccId, role.id());
                } } }
        private OccGraphic createSoccGraphic(String occId, Object[] occspec, Role role) {
                String ename = (String) occspec[1];
                //String locname = (String) occspec[3];
                modnames.clear();
                mods.clear();
                modnames.addAll((ArrayList<String>)occspec[2]);
                for(String mod : modnames)
                        mods.add(abbrevMap.get(mod));
                OccurrenceOval occ = new OccurrenceOval(ename, role, mods, true);
                OccurrenceOval occ2 = null;
                switch(role) {
                        case CONSUMED: panelWest.addToCell(occ); break;
                        case PRODUCED: occ2 = occ; occ = null; panelEast.addToCell(occ2);
        break;
                        case CONTROL: occ2 = new OccurrenceOval(ename, role, mods, true);
                                        panelWest.addToCell(occ);
        panelEast.addToCell(occ2); break;
                } return new OccGraphic(occ,occ2);
        }
        private void drawRuleGraphics() {
                Role role = Role.CONSUMED;
                for(Iterator<String> it = rule.getRuleGraphics().keySet().iterator();
        it.hasNext();) {
                        String occId = it.next();
                        OccGraphic occ = rule.getRuleGraphics().get(occId);
                        if(occ.occBefore()!=null && occ.occAfter()==null) {
                                role = Role.CONSUMED;
                                panelWest.addToCell(occ.occBefore());
                        } else if(occ.occBefore()==null && occ.occAfter()!=null) {
                                role = Role.PRODUCED;
                                panelEast.addToCell(occ.occAfter());
                        } else if(occ.occBefore()!=null && occ.occAfter()!=null) {
                                role = Role.CONTROL;
                                panelWest.addToCell(occ.occBefore());
                                panelEast.addToCell(occ.occAfter());
                        }
                        addToTree(occId, role.id());
        }}
}
//-----------------------------------------------------------------------------------------
public enum Role {
        CONSUMED(0), PRODUCED(1), CONTROL(2);
        private int id;
        Role(int id) { this.id = id; }
        public int id() { return id; }
}
//-----------------------------------------------------------------------------------------
public enum Position {
        EAST(1), NORTH(3), WEST(5), SOUTH(7), NORTH_EAST(2), NORTH_WEST(4),
        SOUTH_WEST(6), SOUTH_EAST(8);
        private int id;
        Position(int id) { this.id = id; }
        public int id() { return id; }
        public static Position getPosition(int id) {
                for(Position p : Position.values())
                        if(p.id()==id) return p;
                return EAST;
        }
}
```